JavaFX

Working with JavaFX UI Components

Release 8

E47848-02

August 2014

In this tutorial, you learn how to build user interfaces in your JavaFX applications with the UI components available through the JavaFX API.

ORACLE

JavaFX Working with JavaFX UI Components Release 8

E47848-02

Primary Author: Alla Redko, Irina Fedortsova

Primary Author:

Contents

5 Toggle Button

6 Checkbox

7 Choice Box

8 Text Field

9 Password Field

10 Scroll Bar

11 Scroll Pane

12 List View

13 Table View

22 Tooltip

23 Titled Pane and Accordion

24 Menu

25 Color Picker

26 Date Picker

27 Pagination Control

28 File Chooser

29 Customization of UI Controls

30 UI Controls on the Embedded Platforms

Part III Working with JavaFX Charts

31 Pie Chart

32 Line Chart

33 Area Chart

34 Bubble Chart

35 Scatter Chart

Part VI Source Code for the UI Components Tutorials

A UI Control Samples

B Chart Samples

C CSS Samples

D Text Samples

Preface

This preface describes the document accessibility features and conventions used in this tutorial - *Working with JavaFX UI Components*.

Audience

This document is intended for JavaFX developers.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents in the JavaFX documentation set:

- *Getting Started with JavaFX*
- *Working with Layouts in JavaFX*
- *Mastering FXML*
- *Handling Events*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.

Convention	Meaning
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Part I

About This Tutorial

This tutorial describes basic UI components available in JavaFX SDK and contains the following chapters:

- Using JavaFX UI Controls
- File Chooser
- Working with JavaFX Charts
- Skinning JavaFX Applications with CSS
- Working with Text in JavaFX Applications

Each chapter provides code samples and applications to illustrate how to use a particular component. You can find the source files of the applications and the corresponding NetBeans projects in the tutorial appendixes.

1

What Is New

This chapter introduces the new user interface (UI) features available with JavaFX SDK 8.0.

Find the following additions:

- Tree Table View

  This chapter describes the `TreeTableView` user interface component, a control designed to visualize an unlimited number of rows of data, broken out into columns.

- Date Packer

  This chapter describes `DatePicker`, a control that enables selection of a day from the given calendar.

- JavaFX UI Controls on the Embedded Platforms

  This chapter describes the specific of using JavaFX UI controls in the embedded environments.

Part II

Using JavaFX UI Controls

This tutorial covers built-in JavaFX UI controls available in the JavaFX API.

The document contains the following chapters:

- File Chooser
- Customization of UI Controls
- UI Controls on the Embedded Platforms

Each chapter provides code samples and applications to illustrate how a particular UI control functions. You can find the source files of the applications and the corresponding NetBeans projects in the table of contents.

2

Label

This chapter explains how to use the `Label` class that resides in the `javafx.scene.control` package of the JavaFX API to display a text element. Learn how to wrap a text element to fit the specific space, add a graphical image, or apply visual effects.

Figure 2–1 shows three common label usages. The label at the left is a text element with an image, the label in the center represents rotated text, and the label at the right renders wrapped text.

Figure 2–1 Sample Application with Labels

Creating a Label

The JavaFX API provides three constructors of the `Label` class for creating labels in your application, as shown in Example 2–1.

Example 2–1 Creating Labels

```
//An empty label
Label label1 = new Label();
//A label with the text element
Label label2 = new Label("Search");
//A label with the text element and graphical icon
Image image = new Image(getClass().getResourceAsStream("labels.jpg"));
Label label3 = new Label("Search", new ImageView(image));
```

Once you have created a label in your code, you can add textual and graphical content to it by using the following methods of the `Labeled` class.

- The `setText(String text)` method – specifies the text caption for the label
- `setGraphic(Node graphic)`– specifies the graphical icon

The `setTextFill` method specifies the color to paint the text element of the label. Study Example 2–2. It creates a text label, adds an icon to it, and specifies a fill color for the text.

Example 2–2 Adding an Icon and Text Fill to a Label

```
Label label1 = new Label("Search");
Image image = new Image(getClass().getResourceAsStream("labels.jpg"));
label1.setGraphic(new ImageView(image));
label1.setTextFill(Color.web("#0076a3"));
```

When this code fragment is added to the application, it produces the label shown in Figure 2–2.

Figure 2–2 Label with Icon

When defining both text and graphical content for your button, you can use the `setGraphicTextGap` method to set the gap between them.

Additionally, you can vary the position of the label content within its layout area by using the `setTextAlignment` method. You can also define the position of the graphic relative to the text by applying the `setContentDisplay` method and specifying one of the following `ContentDisplay` constant: `LFFT`, `RIGHT`, `CENTER`, `TOP`, `BOTTOM`.

Setting a Font

Compare the Search labels in Figure 2–1 and Figure 2–2. Notice that the label in Figure 2–1 has a larger font size. This is because the code fragment shown in Example 2–2 does not specify any font settings for the label. It is rendered with the default font size.

To provide a font text size other than the default for your label use the `setFont` method of the `Labeled` class. The code fragment in Example 2–3 sets the size of the label1 text to 30 points and the font name to Arial. For label2 sets the text size to 32 points and the font name to Cambria.

Example 2–3 Applying Font Settings

```
//Use a constructor of the Font class
label1.setFont(new Font("Arial", 30));
//Use the font method of the Font class
label2.setFont(Font.font("Cambria", 32));
```

Wrapping Text

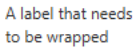
When you create a label, sometimes you must fit it within a space that is smaller than you need to render. To break up (wrap) the text so that is can fit into the layout area, set the `true` value for the `setWrapText` method, as shown in Example 2–4.

Example 2–4 Enable Text Wrapping

```
Label label3 = new Label("A label that needs to be wrapped");
label3.setWrapText(true);
```

When label3 is added to the content of an application, it is rendered as shown in Figure 2–3.

Figure 2–3 Label with Wrapped Text



Suppose that the layout area of the label is limited not only by its width, but also by its height. You can specify the behavior of a label when it is impossible to render the entire required text string. Use the `setTextOverrun` method of the `Labeled` class and one of the available `OverrunStyle` types to define how to process the part of the text string that cannot be rendered properly. See the API documentation for more information about `OverrunStyle` types.

Applying Effects

Although a label is static content and cannot be edited, you can apply visual effects or transformations to it. The code fragment in Example 2–5 rotates label2 270 degrees and translates its position vertically.

Example 2–5 Rotating a Label

```
Label label2 = new Label ("Values");
label2.setFont(new Font("Cambria", 32));
label2.setRotate(270);
label2.setTranslateY(50);
```

Rotation and translation are typical transformations available in the JavaFX API. Additionally, you can set up an effect that zooms (magnifies) the label when a user hovers the mouse cursor over it.

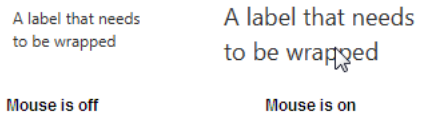
The code fragment shown in Example 2–6 applies the zoom effect to label3. When the `MOUSE_ENTERED` event is fired on the label, the scaling factor of 1.5 is set for the `setScaleX` and `setScaleY` methods. When a user moves the mouse cursor off of the label and the `MOUSE_EXITED` event occurs, the scale factor is set to 1.0, and the label is rendered in its original size.

Example 2–6 Applying the Zoom Effect

```
label3.setOnMouseEntered((MouseEvent e) -> {
    label3.setScaleX(1.5);
    label3.setScaleY(1.5);
});

label3.setOnMouseExited((MouseEvent e) -> {
    label3.setScaleX(1);
    label3.setScaleY(1);
});
```

Figure 2–4 shows the two states of label3.

Figure 2–4 Zooming a Label



Related API Documentation

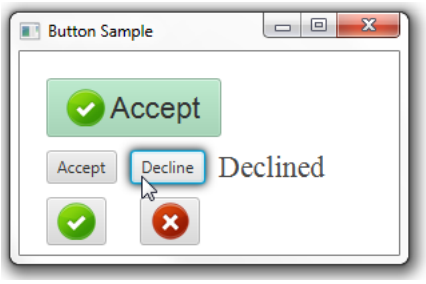
- `Label`
- `Labeled`

3

Button

The `Button` class available through the JavaFX API enables developers to process an action when a user clicks a button. The `Button` class is an extension of the `Labeled` class. It can display text, an image, or both. Figure 3–1 shows buttons with various effects. In this chapter you will learn how to create each of these button types.

Figure 3–1 Types of Buttons



Creating a Button

You can create a `Button` control in a JavaFX application by using three constructors of the Button class as shown on Example 3–1.

Example 3–1 Creating a Button

```
//A button with an empty text caption.
Button button1 = new Button();
//A button with the specified text caption.
Button button2 = new Button("Accept");
//A button with the specified text caption and icon.
Image imageOk = new Image(getClass().getResourceAsStream("ok.png"));
Button button3 = new Button("Accept", new ImageView(imageOk));
```

Because the `Button` class extends the `Labeled` class, you can use the following methods to specify content for a button that does not have an icon or text caption:

- The `setText(String text)` method – specifies the text caption for the button
- The `setGraphic(Node graphic)` method – specifies the graphical icon

Example 3–2 shows how to create a button with an icon but without a text caption.

Example 3–2 Adding an Icon to a Button

```
Image imageDecline = new Image(getClass().getResourceAsStream("not.png"));
```

```
Button button5 = new Button();
button5.setGraphic(new ImageView(imageDecline));
```

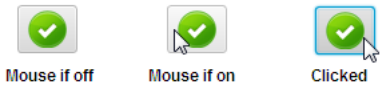
When added to the application, this code fragment produces the button shown in Figure 3–2.

Figure 3–2 Button with Icon

In Example 3–2 and Figure 3–2, the icon is an `ImageView` object. However, you can use other graphical objects, for example, shapes that reside in the `javafx.scene.shape` package. When defining both text and graphical content for your button, you can use the `setGraphicTextGap` method to set the gap between them.

The default skin of the `Button` class distinguishes the following visual states of the button. Figure 3–3 shows the default states of a button with an icon.

Figure 3–3 Button States



Assigning an Action

The primary function of each button is to produce an action when it is clicked. Use the `setOnAction` method of the `Button` class to define what will happen when a user clicks the button. Example 3–3 shows a code fragment that defines an action for button2.

Example 3–3 Defining an Action for a Button

```
button2.setOnAction((ActionEvent e) -> {
    label.setText("Accepted");
});
```

Example 3–3 shows how to process an `ActionEvent`, so that when a user presses button2 the text caption for a label is set to "Accepted."

You can use the `Button` class to set as many event-handling methods as you need to cause the specific behavior or apply visual effects.

Applying Effects

Because the `Button` class extends the `Node` class, you can apply any of the effects in the `javafx.scene.effect` package to enhance the visual appearance of the button. In Example 3–4, the `DropShadow` effect is applied to button3 when the `onMouseEntered` event occurs.

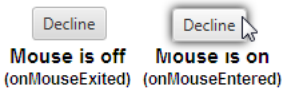
Example 3–4 Applying the DropShadow Effect

```
DropShadow shadow = new DropShadow();
//Adding the shadow when the mouse cursor is on
button3.addEventHandler(MouseEvent.MOUSE_ENTERED, (MouseEvent e) -> {
```

```
    button3.setEffect(shadow);
});

//Removing the shadow when the mouse cursor is off
button3.addEventHandler(MouseEvent.MOUSE_EXITED, (MouseEvent e) -> {
    button3.setEffect(null);
});
```

Figure 3–4 shows the states of button3 when the mouse cursor is on it and when it is off.

Figure 3–4 Button with Drop Shadow



Styling a Button

The next step to enhance the visual appearance of a button is to apply CSS styles defined by the `Skin` class. Using CSS in JavaFX 2 applications is similar to using CSS in HTML, because each case is based on the same CSS specification.

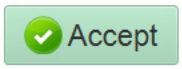
You can define styles in a separate CSS file and enable them in the application by using the `getStyleClass` method. This method is inherited from the `Node` class and is available for all UI controls. Example 3–5 and Figure 3–5 demonstrate this approach.

Example 3–5 Styling a Button

```
//Code added to the CSS file
.button1{
    -fx-font: 22 arial;
    -fx-base: #b6e7c9;
}
//Code in the ButtonSample.java file
button1.getStyleClass().add("button1");
```

The `-fx-font` property sets the font name and size for button1. The `-fx-base` property overrides the default color applied to the button. As the result, button1 is light green with lager text size, as shown in Figure 3–5.

Figure 3–5 Button Styled with CSS



Related API Documentation

- `Button`
- `Labeled`

4

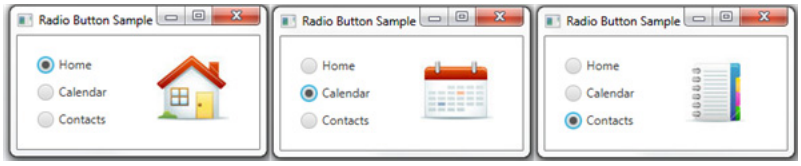
Radio Button

This chapter discusses the radio button control and the `RadioButton` class, a specialized implementation of the `ToggleButton` class.

A radio button control can be either selected or deselected. Typically radio buttons are combined into a group where only one button at a time can be selected. This behavior distinguishes them from toggle buttons, because all toggle buttons in a group can be in a deselected state.

Figure 4–1 shows three screen captures of the `RadioButton` sample, in which three radio buttons are added to a group.

Figure 4–1 RadioButton Sample



Study the following paragraphs to learn more about how to implement radio buttons in your applications.

Creating a Radio Button

The `RadioButton` class available in the `javafx.scene.control` package of the JavaFX SDK provides two constructors with which you can create a radio button. Example 4–1 shows two radio buttons. The constructor with no parameters is used to create rb1. The text caption for this radio button is set by using the `setText` method. The text caption for rb2 is defined within the corresponding constructor.

Example 4–1 Creating Radio Buttons

```
//A radio button with an empty string for its label
RadioButton rb1 = new RadioButton();
//Setting a text label
rb1.setText("Home");
//A radio button with the specified label
RadioButton rb2 = new RadioButton("Calendar");
```

You can explicitly make a radio button selected by using the `setSelected` method and specifying its value as `true`. If you need to check whether a particular radio button was selected by a user, apply the `isSelected` method.

Because the `RadioButton` class is an extension of the `Labeled` class, you can specify not only a text caption, but also an image. Use the `setGraphic` method to specify an image. Example 4–2 demonstrates how to implement a graphical radio button in your application.

Example 4–2 Creating a Graphical Radio Button

```
Image image = new Image(getClass().getResourceAsStream("ok.jpg"));
RadioButton rb = new RadioButton("Agree");
rb.setGraphic(new ImageView(image));
```

Adding Radio Buttons to Groups

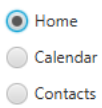
Radio buttons are typically used in a group to present several mutually exclusive options. The `ToggleGroup` object provides references to all radio buttons that are associated with it and manages them so that only one of the radio buttons can be selected at a time. Example 4–3 creates a toggle group, creates three radio buttons, adds each radio button to the toggle group, and specifies which button should be selected when the application starts.

Example 4–3 Creating a Group of Radio Buttons

```
final ToggleGroup group = new ToggleGroup();

RadioButton rb1 = new RadioButton("Home");
rb1.setToggleGroup(group);
rb1.setSelected(true);

RadioButton rb2 = new RadioButton("Calendar");
rb2.setToggleGroup(group);

RadioButton rb3 = new RadioButton("Contacts");
rb3.setToggleGroup(group);
```

When these radio buttons are laid out by using the layout containers and added to the content of the application, the output should resemble Figure 4–2.

Figure 4–2 Three Radio Buttons Combined in a Group



Processing Events for Radio Buttons

Typically, the application performs an action when one of the radio buttons in the group is selected. Review the code fragment in Example 4–4 to learn how to change an icon according to which radio button is selected.

Example 4–4 Processing Action for Radio Buttons

```
ImageView image = new ImageView();

rb1.setUserData("Home")
rb2.setUserData("Calendar");
```

```
rb3.setUserData("Contacts");

final ToggleGroup group = new ToggleGroup();

group.selectedToggleProperty().addListener(
    (ObservableValue<? extends Toggle> ov, Toggle old_toggle,
    Toggle new_toggle) -> {
        if (group.getSelectedToggle() != null) {
            final Image image = new Image(
                getClass().getResourceAsStream(
                group.getSelectedToggle().getUserData().toString() +
                ".jpg"));
        icon.setImage(image);
    }
});
```

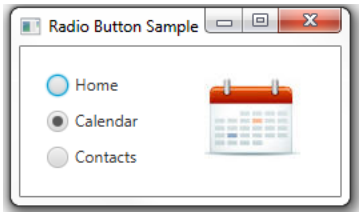
The user data was assigned for each radio button. The `ChangeListener<Toggle>` object checks a selected toggle in the group. It uses the `getSelectedToggle` method to identify which radio button is currently selected and extracts its user data by calling the `getUserData` method. Then the user data is applied to construct an image file name to load.

For example, when rb3 is selected, the `getSelectedToggle` method returns "rb3," and the `getUserData` method returns "Contacts." Therefore, the `getResourceAsStream` method receives the value "Contacts.jpg." The application output is shown in Figure 4–1.

Requesting Focus for a Radio Button

In the group of radio buttons, the first button initially has the focus by default. If you apply the `setSelected` method to the second radio button in the group, you should expect the result shown in Figure 4–3.

Figure 4–3 Default Focus Settings



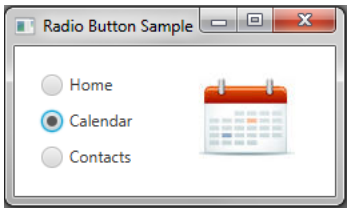
The second radio button is selected, and the first button remains in focus. Use the `requestFocus` function to change the focus, as shown in Example 4–5.

Example 4–5 Requesting Focus for the Second Radio Button

```
rb2.setSelected(true);
rb2.requestFocus();
```

When applied, this code produces the result shown in Figure 4–4.

Figure 4–4 Setting Focus for the Selected Radio Button



Related API Documentation

- `RadioButton`
- `Labeled`
- `ToggleGroup`

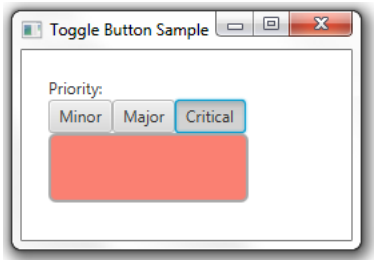
5

Toggle Button

In this chapter, you learn about the `ToggleButton` class, another type of buttons available through the JavaFX API.

Two or more toggle buttons can be combined into a group where only one button at a time can be selected, or where no selection is required. Figure 5–1 is a screen capture of an application that combines three toggle buttons in a group. The application paints the rectangle with a specific color according to on which toggle button is selected.

Figure 5–1 Three Toggle Buttons



Creating a Toggle Button

You can create a toggle button in your application by using any of the three constructors of the `ToggleButton` class, as shown in Example 5–1.

Example 5–1 Creating Toggle Buttons

```
//A toggle button without any caption or icon
ToggleButton tb1 = new ToggleButton();
//A toggle button with a text caption
ToggleButton tb2 = new ToggleButton("Press me");
//A toggle button with a text caption and an icon
Image image = new Image(getClass().getResourceAsStream("icon.png"));
ToggleButton tb3 = new ToggleButton ("Press me", new ImageView(image));
```

The `ToggleButton` class is an extension of the `Labeled` class, so you can specify a text caption, an image, or both image and text. You can use the `setText` and `setGraphic` methods of the `Labeled` class to specify textual and graphical content for a toggle button.

Once you have defined toggle buttons in your code, you can combine them in a group and set a specific behavior.

Adding Toggle Buttons to a Group

The implementation of the `ToggleButton` class is very similar to the implementation the `RadioButton` class. However, unlike radio buttons, toggle buttons in a toggle group do not attempt to force the selection at least one button in the group. That is, clicking the selected toggle button causes it to become deselected, clicking the selected radio button in the group has no effect.

Take a moment to study the code fragment Example 5–2.

Example 5–2 Combining Toggle Buttons in a Group

```
final ToggleGroup group = new ToggleGroup();

ToggleButton tb1 = new ToggleButton("Minor");
tb1.setToggleGroup(group);
tb1.setSelected(true);

ToggleButton tb2 = new ToggleButton("Major");
tb2.setToggleGroup(group);

ToggleButton tb3 = new ToggleButton("Critical");
tb3.setToggleGroup(group);
```

Example 5–2 creates three toggle buttons and adds them to the toggle group. The `setSelected` method is called for the tb1 toggle button so that it is selected when the application starts. However, you can deselect the Minor toggle button so that no toggle buttons are selected in the group at startup, as shown in Figure 5–2.

Figure 5–2 Three Toggle Buttons in a Group

Typically, you use a group of toggle buttons to assign a specific behavior for each button. The next section explains how to use these toggle buttons to alter the color of a rectangle.

Setting the Behavior

The `setUserData` method inherited by the `ToggleButton` class from the `Node` class helps you to associate any selected option with a particular value. In Example 5–3, the user data indicates which color should be used to paint the rectangle.

Example 5–3 Setting User Data for the Toggle Buttons

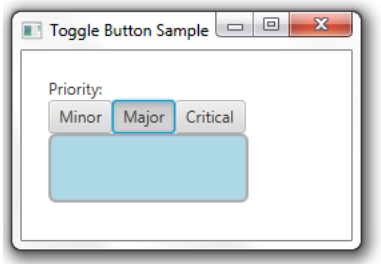
```
tb1.setUserData(Color.LIGHTGREEN);
tb2.setUserData(Color.LIGHTBLUE);
tb3.setUserData(Color.SALMON);

Rectangle rect = new Rectangle();
rect.setHeight(50);
rect.setFill(Color.WHITE);
rect.setStroke(Color.DARKGRAY);
rect.setStrokeWidth(2);
rect.setArcHeight(10);
```

```
rect.setArcWidth(10);

final ToggleGroup group = new ToggleGroup();

group.selectedToggleProperty().addListener
    (ObservableValue<? extends Toggle> ov,
    Toggle toggle, Toggle new_toggle) -> {
    if (new_toggle == null)
        rect.setFill(Color.WHITE);
    else
        rect.setFill((Color) group.getSelectedToggle().getUserData());
});

rect.setWidth(hbox.getWidth());
```

The `ChangeListener<Toggle>` object checks a selected toggle in the group. If none of the toggle buttons is selected, the rectangle is painted with the white color. If one of the toggle button is selected, consecutive calls of the `getSelectedToggle` and `getUserData` methods return a color to paint the rectangle.

For example, if a user selects the tb2 toggle button, the `setSelectedToggle().getUserData()` call returns `Color.LIGHTBLUE`. The result is shown in Figure 5–3.

Figure 5–3 Using Toggle Buttons to Paint a Rectangle



See the ToggleButtonSample.java file to examine the complete code of the application.

Styling Toggle Buttons

You can enhance this application by applying CSS styles to the toggle buttons. Using CSS in JavaFX applications is similar to using CSS in HTML, because each case is based on the same CSS specification.

First, you declare the styles of the toggle buttons in the ControlStyle.css file as shown in Example 5–4.

Example 5–4 Declaring Alternative Colors of the Toggle Button

```
.toggle-button1{
    -fx-base: lightgreen;
}

.toggle-button2{
    -fx-base: lightblue;
}

.toggle-button3{
    -fx-base: salmon;
```

```
}
```

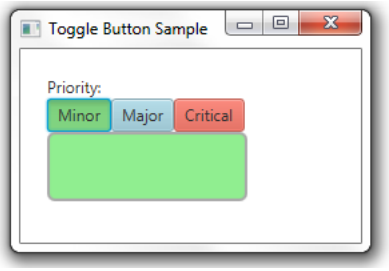
Second, you enable the styles in the ToggleButtonSample application. See how this is implemented in Example 5–5.

Example 5–5 Applying CSS Styles to Toggle Buttons

```
scene.getStylesheets().add("togglebuttonsample/ControlStyle.css");

tb1.getStyleClass().add("toggle-button1");
tb2.getStyleClass().add("toggle-button2");
tb3.getStyleClass().add("toggle-button3");
```

When added to the application code these lines change the visual appearance of the toggle buttons as shown in Figure 5–4.

Figure 5–4 Painted Toggle Buttons



You might want to try other CSS properties of the `ToggleButton` class or apply animation, transformations, and visual effects available in the JavaFX API.

Related API Documentation

- `ToggleButton`
- `ToggleGroup`

6

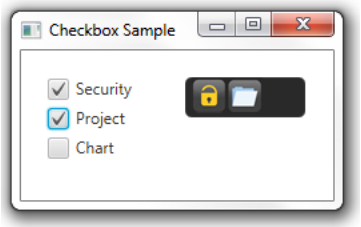
Checkbox

This chapter teaches how to add checkboxes to your JavaFX applications.

Although checkboxes look similar to radio buttons, they cannot be combined into toggle groups to enable the selection of many options at one time. See the Radio Button and Toggle Button chapters for more information.

Figure 6–1 shows a screen capture of an application in which three checkboxes are used to enable or disable icons in an application toolbar.

Figure 6–1 Checkbox Sample



Creating Checkboxes

Example 6–1 creates two simple checkboxes.

Example 6–1 Creating Checkboxes

```
//A checkbox without a caption
CheckBox cb1 = new CheckBox();
//A checkbox with a string caption
CheckBox cb2 = new CheckBox("Second");

cb1.setText("First");
cb1.setSelected(true);
```

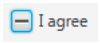
Once you have created a checkbox, you can modify it by using methods available through the JavaFX APIs. In Example 6–1 the `setText` method defines the text caption of the c1 checkbox. The `setSelected` method is set to `true` so that the cb1 checkbox is selected when the application is started.

Defining a State

The checkbox can be either defined or undefined. When it is defined, you can select or deselect it. However, when the checkbox is undefined, it cannot be selected or

deselected. Use a combination of the `setSelected` and `setIndeterminate` methods of the `CheckBox` class to specify the state of the checkbox. Table 6–1 shows three states of a checkbox based on its `INDETERMINATE` and `SELECTED` properties.

Table 6–1 States of a Checkbox

Property Values	Checkbox Appearance
`INDETERMINATE` = false `SELECTED` = false	
`INDETERMINATE` =false `SELECTED` = true	
`INDETERMINATE` = true `SELECTED` = true/false	

You might need enabling three states for checkboxes in your application when they represent UI elements that can be in mixed states, for example, "Yes", "No", "Not Applicable." The `allowIndeterminate` property of the `CheckBox` object determines whether the checkbox should cycle through all three states: selected, deselected, and undefined. If the variable is `true`, the control will cycle through all the three states. If it is `false`, the control will cycle through the selected and deselected states. The application described in the next section constructs three checkboxes and enables only two states for them.

Setting the Behavior

The code fragment in Example 6–2 creates three checkboxes, such that if a checkbox is selected, the corresponding icon appears in a toolbar.

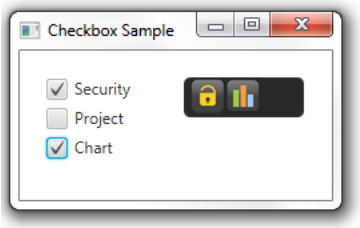
Example 6–2 Setting the Behavior for the Checkboxes

```
final String[] names = new String[]{"Security", "Project", "Chart"};
final Image[] images = new Image[names.length];
final ImageView[] icons = new ImageView[names.length];
final CheckBox[] cbs = new CheckBox[names.length];

for (int i = 0; i < names.length; i++) {
    final Image image = images[i] =
        new Image(getClass().getResourceAsStream(names[i] + ".png"));
    final ImageView icon = icons[i] = new ImageView();
    final CheckBox cb = cbs[i] = new CheckBox(names[i]);
    cb.selectedProperty().addListener(
        (ObservableValue<? extends Boolean> ov,
            Boolean old_val, Boolean new_val) -> {
                icon.setImage(new_val ? image : null);
    });
}
```

The `names` array uses a `for` loop to create an array of checkboxes and a corresponding array of icons. For example, cbs[0], the first checkbox, is assigned the "Security" text caption. At the same time, image[0] receives "Security.png" as a file name for the `getResourceStream` method when an image for the first icon is created. If a particular checkbox is selected, the corresponding image is assigned to the icon. If a checkbox is deselected, the icon receives a `null` image and the icon is not rendered.

Figure 6–2 shows an application when the Security and Chart checkboxes are selected and the Project checkbox is deselected.

Figure 6–2 Checkbox Application in Action



Related API Documentation

- `CheckBox`
- JavaFX CSS Specification

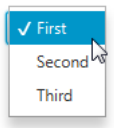
7

Choice Box

This chapter describes choice boxes, the UI controls that provide support for quickly selecting between a few options.

Use the `ChoiceBox` class to add choice boxes to your JavaFX applications. Its simple implementation is shown in Figure 7–1.

Figure 7–1 Creating a Choice Box with Three Items



Creating a Choice Box

Example 7–1 creates a choice box with three items.

Example 7–1 Creating a Choice Box

```
ChoiceBox cb = new ChoiceBox(FXCollections.observableArrayList(
    "First", "Second", "Third")
);
```

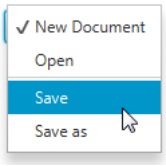
Example 7–1 shows a list of items created and populated within a constructor of the `ChoiceBox` class. The list items are specified by using an observable array. Alternatively, you can use an empty constructor of the class and set the list items by using the `setItems` method shown in Example 7–2.

Example 7–2 Choice Box with Text Elements and a Separator

```
ChoiceBox cb = new ChoiceBox();
cb.setItems(FXCollections.observableArrayList(
    "New Document", "Open ",
    new Separator(), "Save", "Save as")
);
```

Note that a choice box can contain not only text elements, but other objects as well. A `Separator` control is used in Example 7–2 to separate the items. When this code fragment is integrated into the application, it produces the output shown in Figure 7–2.

Figure 7–2 Menu Created by Using a Choice Box



In real-life applications, the choice boxes are used to build multiple-choice lists.

Setting the Behavior for a Choice Box

The application shown in Figure 7–3 provides a multiple-choice box with five options. When a particular language is selected, the corresponding greeting is rendered.

Figure 7–3 Multiple-Choice List

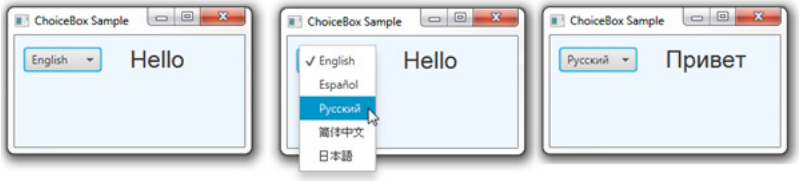


Figure 7–4 provides a code fragment to illustrate how an item selected from the choice box defines which greeting should be rendered.

Figure 7–4 Selecting a Choice Box Item

```
final String[] greetings = new String[]{"Hello", "Hola", "Привет", "你好",
    "こんにちは"};
final ChoiceBox cb = new ChoiceBox(FXCollections.observableArrayList(
    "English", "Español", "Русский", "简体中文", "日本語")
);

cb.getSelectionModel().selectedIndexProperty().addListener(
    (ObservableValue<? extends Number> ov,
         Number old_val, Number new_val) -> {
             label.setText(greetings[new_val.intValue()]);
 });
```

A `ChangeListener<Number>` object detects the index of the currently selected item by consecutive calls of the `getSelectionModel` and `selectedIndexProperty` methods. The `getSelectionModel` method returns the selected item, and the `selectedIndexProperty` method returns the `SELECTED_INDEX` property of the cb choice box. As a result, the integer value as an index defines an element of the greetings array and specifies a `String` text value for the label. If, for example, a user selects the second item, which corresponds to Spanish, the `SELECTED_INDEX` is equal to 1 and "Hola" is selected from the greetings array. Thus, the label renders "Hola."

You can make the `ChoiceBox` control more informative by assigning a tooltip to it. A tooltip is a UI control that is available in the `javafx.scene.control` package. A tooltip can be applied to any of the JavaFX UI controls.

Applying a Tooltip

The `Tooltip` class provides a prefabricated tooltip that can be easily applied to a choice box (or any other control) by calling the `setTooltip` method shown in Example 7–3.

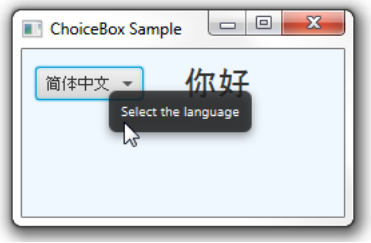
Example 7–3 Adding a Tooltip to a Choice Box

```
cb.setTooltip(new Tooltip("Select the language"));
```

Typically a user defines the text of the tooltip within a constructor of the `Tooltip` class. However, if the logic of your application requires UI to set the text dynamically, you can apply a tooltip by using an empty constructor and then assign the text to it by using the `setText` method.

After the tooltip is applied to the cb choice box, a user who positions the cursor over the choice box sees the image shown in Figure 7–5.

Figure 7–5 Choice Box with the Applied Tooltip



To further enhance your application, you can style the choice box with the CSS properties or apply visual effects or transformations.

Related API Documentation

- `ChoiceBox`
- `Tooltip`

8

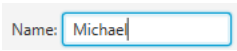
Text Field

This chapter discusses the capabilities of the text field control.

The `TextField` class implements a UI control that accepts and displays text input. It provides capabilities to receive text input from a user. Along with another text input control, `PasswordField`, this class extends the `TextInput` class, a super class for all the text controls available through the JavaFX API.

Figure 8–1 shows a typical text field with a label.

Figure 8–1 Label and Text Field



Creating a Text Field

In Example 8–1, a text field is used in combination with a label to indicate the type of content that should be typed in the field.

Example 8–1 Creating a Text Field

```
Label label1 = new Label("Name:");
TextField textField = new TextField ();
HBox hb = new HBox();
hb.getChildren().addAll(label1, textField);
hb.setSpacing(10);
```

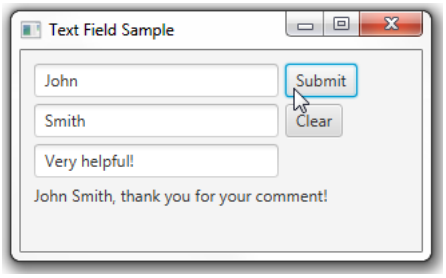
You can create an empty text field as shown in Example 8–1 or a text field with a particular text data in it. To create a text field with the predefined text, use the following constructor of the `TextField` class: `TextField("Hello World!")`. You can obtain the value of a text field at any time by calling the `getText` method.

You can apply the `setPrefColumnCount` method of the `TextInput` class to set the size of the text field, defined as the maximum number of characters it can display at one time.

Building the UI with Text Fields

Typically, the `TextField` objects are used in forms to create several text fields. The application in Figure 8–2 displays three text fields and processes the data that a user enters in them.

Figure 8–2 TextFieldSample Application



The code fragment in Example 8–2 creates the three text fields and two buttons, and adds them to the application's scene by using the `GridPane` container. This container is particularly handy when you need to implement a flexible layout for your UI controls.

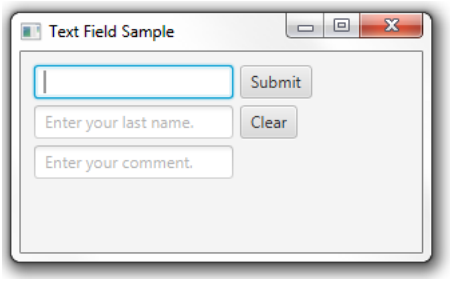
Example 8–2 Adding Text Fields to the Application

```
//Creating a GridPane container
GridPane grid = new GridPane();
grid.setPadding(new Insets(10, 10, 10, 10));
grid.setVgap(5);
grid.setHgap(5);

//Defining the Name text field
final TextField name = new TextField();
name.setPromptText("Enter your first name.");
GridPane.setConstraints(name, 0, 0);
grid.getChildren().add(name);

//Defining the Last Name text field
final TextField lastName = new TextField();
lastName.setPromptText("Enter your last name.");
GridPane.setConstraints(lastName, 0, 1);
grid.getChildren().add(lastName);

//Defining the Comment text field
final TextField comment = new TextField();
comment.setPromptText("Enter your comment.");
GridPane.setConstraints(comment, 0, 2);
grid.getChildren().add(comment);

//Defining the Submit button
Button submit = new Button("Submit");
GridPane.setConstraints(submit, 1, 0);
grid.getChildren().add(submit);

//Defining the Clear button
Button clear = new Button("Clear");
GridPane.setConstraints(clear, 1, 1);
grid.getChildren().add(clear);
```

Take a moment to study the code fragment. The `name`, `lastName`, and `comment` text fields are created by using empty constructors of the `TextField` class. Unlike Example 8–1, labels do not accompany the text fields in this code fragment. Instead, prompt captions notify users what type of data to enter in the text fields. The `setPromptText` method defines the string that appears in the text field when the application is started. When Example 8–2 is added to the application, it produces the

output shown in Figure 8–3.

Figure 8–3 Three Text Fields with the Prompt Messages



The difference between the prompt text and the text entered in the text field is that the prompt text cannot be obtained through the `getText` method.

In real-life applications, data entered into the text fields is processed according to an application's logic as required by a specific business task. The next section explains how to use text fields to evaluate the entered data and generate a response to a user.

Processing Text Field Data

As mentioned earlier, the text data entered by a user into the text fields can be obtained by the `getText` method of the `TextInput` class.

Study Example 8–3 to learn how to process the data of the `TextField` object.

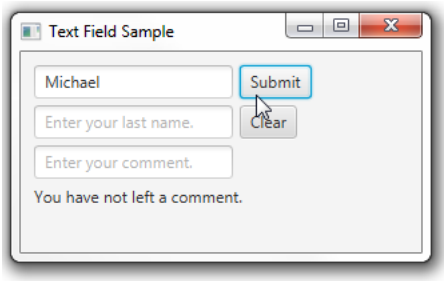
Example 8–3 Defining Actions for the Submit and Clear Buttons

```
//Adding a Label
final Label label = new Label();
GridPane.setConstraints(label, 0, 3);
GridPane.setColumnSpan(label, 2);
grid.getChildren().add(label);

submit.setOnAction((ActionEvent e) -> {
    if (
        (comment.getText() != null && !comment.getText().isEmpty())
    ) {
    label.setText(name.getText() + " " +
        lastName.getText() + ", "
        + "thank you for your comment!");
    } else {
        label.setText("You have not left a comment.");
    }
});

clear.setOnAction((ActionEvent e) -> {
    name.clear();
    lastName.clear();
    comment.clear();
    label.setText(null);
});
```

The `Label` control added to the `GridPane` container renders an application's response to users. When a user clicks the Submit button, the `setOnAction` method checks the `comment` text field. If it contains a nonempty string, a thank-you message is rendered.

Otherwise, the application notifies a user that the comment message has not been left yet, as shown in Figure 8–4.

Figure 8–4 The Comment Text Field Left Blank



When a user clicks the Clear button, the content is erased in all three text fields.

Review some helpful methods that you can use with text fields.

- `copy()` – transfers the currently selected range in the text to the clipboard, leaving the current selection.
- `cut()` – transfers the currently selected range in the text to the clipboard, removing the current selection.
- `selectAll()` - selects all text in the text input.
- `paste()` – transfers the contents in the clipboard into this text, replacing the current selection.

Related API Documentation

- `TextField`
- `TextInputControl`

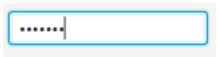
9

Password Field

In this chapter, you learn about yet another type of the text control, the password field.

The `PasswordField` class implements a specialized text field. The characters typed by a user are hidden by displaying an echo string. Figure 9–1 shows a password field with an entered password.

Figure 9–1 Password Field



Creating a Password Field

An entry-level task is to create a password field by using the code in Example 9–1.

Example 9–1 Creating a Password Field

```
PasswordField passwordField = new PasswordField();
passwordField.setPromptText("Your password");
```

For your user interface, you can accompany the password field with a prompt message or you can add a notifying label. As with the `TextField` class, the `PasswordField` class provides the `setText` method to render a text string in the control when the application is started. However, the string specified in the `setText` method is hidden by the echo characters in the password field. By default, the echo character is a dot. Figure 9–2 shows the password field with the predefined text in it.

Figure 9–2 Password Field with the Set Text

The value typed in the password field can be obtained through the `getText` method. You can process this value in your application and set the authentication logic as appropriate.

Evaluating the Password

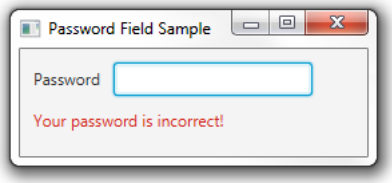
Take a moment to review in Example 9–2 the implementation of a password field that you can apply in your user interface.

Example 9–2 Implementing the Authentication Logic

```
final Label message = new Label("");

VBox vb = new VBox();
vb.setPadding(new Insets(10, 0, 0, 10));
vb.setSpacing(10);
HBox hb = new HBox();
hb.setSpacing(10);
hb.setAlignment(Pos.CENTER_LEFT);

Label label = new Label("Password");
final PasswordField pb = new PasswordField();

pb.setOnAction((ActionEvent e) -> {
    if (!pb.getText().equals("T2f$Ay!")) {
        message.setText("Your password is incorrect!");
        message.setTextFill(Color.rgb(210, 39, 30));
    } else {
        message.setText("Your password has been confirmed");
        message.setTextFill(Color.rgb(21, 117, 84));
    }
    pb.clear();
});

hb.getChildren().addAll(label, pb);
vb.getChildren().addAll(hb, message);
```

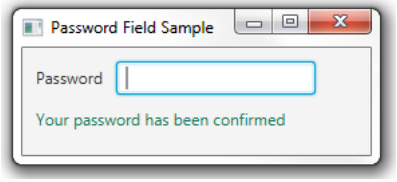
The authentication logic of the password field is defined by using the `setOnAction` method. This method is called when a password is committed and it processes the typed value. If the typed value is different from the required password, the corresponding message appears in red as shown in Figure 9–3.

Figure 9–3 Password is Incorrect



If the typed value satisfies the predefined criteria, the confirmation message appears as shown in Figure 9–4.

Figure 9–4 Password is Correct



For security reasons, it is good practice to clear the password field after the value is typed. In Example 9–2, an empty string is set for the passwordField after the authentication is performed.

Related API Documentation

- `PasswordField`
- `TextInputControl`

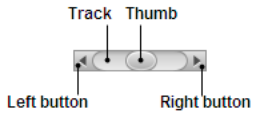
10

Scroll Bar

This chapter explains how to create scrollable panes by using the scroll bar control.

The `ScrollBar` class enables you to create scrollable panes and views in your application. Figure 10–1 shows the three areas of a scroll bar: the thumb, the right and left buttons (or down and up buttons), and the track.

Figure 10–1 Elements of the scroll bar



Creating a Scroll Bar

Take a moment to review the code fragment in Example 10–1.

Example 10–1 Simple Scroll Bar

```
ScrollBar sc = new ScrollBar();
sc.setMin(0);
sc.setMax(100);
sc.setValue(50);
```

The `setMin` and `setMax` methods define the minimum and maximum values represented by the scroll bar. When a user moves the thumb, the value of the scroll bar changes. In Example 10–1, the value equals 50, so when the application starts, the thumb is in the center of the scroll bar. By default, the scroll bar is oriented horizontally. However, you can set the vertical orientation by using the `setOrientation` method.

The user can click the left or right button (down or up button for the vertical orientation) to scroll by a unit increment. The `UNIT_INCREMENT` property specifies the amount by which the scroll bar is adjusted when a button is clicked. Another option is clicking within the track by a block increment. The `BLOCK_INCREMENT` property defines the amount by which the scroll bar is adjusted when the track of the bar is clicked.

In your application, you can use one of several scroll bars to scroll through graphical content that exceeds the borders of the available space.

Using a Scroll Bar in Your Application

Examine the scroll bar in action. The application shown in Example 10–2 implements a scrollable scene to view the images. The task of this application is to enable users to view the content of the vertical box, which is longer than the scene's height.

Example 10–2 Scrolling Through Multiple Images

```
import javafx.application.Application;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Orientation;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.ScrollBar;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class ScrollBarSample extends Application {

    final ScrollBar sc = new ScrollBar();
    final Image[] images = new Image[5];
    final ImageView[] pics = new ImageView[5];
    final VBox vb = new VBox();
    DropShadow shadow = new DropShadow();

    @Override
    public void start(Stage stage) {
        Group root = new Group();
        Scene scene = new Scene(root, 180, 180);
        scene.setFill(Color.BLACK);
        stage.setScene(scene);
        stage.setTitle("Scrollbar");
        root.getChildren().addAll(vb, sc);

        shadow.setColor(Color.GREY);
        shadow.setOffsetX(2);
        shadow.setOffsetY(2);

        vb.setLayoutX(5);
        vb.setSpacing(10);

        sc.setLayoutX(scene.getWidth()-sc.getWidth());
        sc.setMin(0);
        sc.setOrientation(Orientation.VERTICAL);
        sc.setPrefHeight(180);
        sc.setMax(360);

        for (int i = 0; i < 5; i++) {
            final Image image = images[i] =
                new Image(getClass().getResourceAsStream(
                    "fw" +(i+1)+ ".jpg")
                );
            final ImageView pic = pics[i] =
                new ImageView(images[i]);
            pic.setEffect(shadow);
```

```
            vb.getChildren().add(pics[i]);
        }

        sc.valueProperty().addListener((ObservableValue<? extends Number> ov,
            Number old_val, Number new_val) -> {
                vb.setLayoutY(-new_val.doubleValue());
        });

        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

The first lines of the code add a vertical box with the images and a scroll bar to the scene.

The `Y` coordinate of the vertical box changes when the `VALUE` property of the scroll bar is changed, so that each time the thumb is moved, or either a button or the track is clicked, the vertical box moves, as shown in Example 10–3.

Example 10–3 Implementing the Scrolling of the Vertical Box

```
sc.valueProperty().addListener((ObservableValue<? extends Number> ov,
    Number old_val, Number new_val) -> {
        vb.setLayoutY(-new_val.doubleValue());
});
```

Compiling and running this application produces the output shown in Figure 10–2.

Figure 10–2 Scroll Bar Sample

This application shows one typical use of the `ScrollBar` class. You can also customize this class to create a scroll area within a scene. As for every UI control and every node, the scroll bar can be styled to change its appearance from the default implementation.

Related API Documentation

- `ScrollBar`
- JavaFX CSS Specification

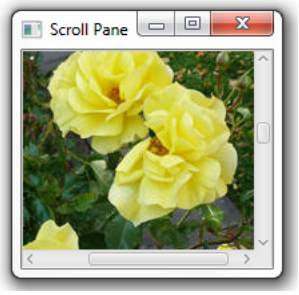
11

Scroll Pane

In this chapter, you learn how to build scroll panes in your JavaFX applications.

Scroll panes provide a scrollable view of UI elements. This control enables the user to scroll the content by panning the viewport or by using scroll bars. A scroll pane with the default settings and the added image is shown in Figure 11–1.

Figure 11–1 Scroll Pane



Creating a Scroll Pane

Example 11–1 shows how to create this scroll pane in your application.

Example 11–1 Using a Scroll Pane to View an Image

```
Image roses = new Image(getClass().getResourceAsStream("roses.jpg"));
ScrollPane sp = new ScrollPane();
sp.setContent(new ImageView(roses));
```

The `setContent` method defines the node that is used as the content of this scroll pane. You can specify only one node. To create a scroll view with more than one component, use layout containers or the `Group` class. You can also specify the `true` value for the `setPannable` method to preview the image by clicking it and moving the mouse cursor. The position of the scroll bars changes accordingly.

Setting the Scroll Bar Policy for a Scroll Pane

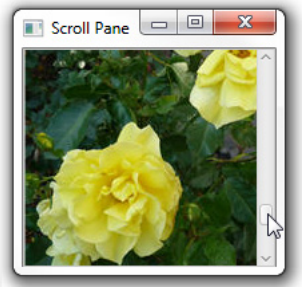
The `ScrollPane` class provides a policy to determine when to display scroll bars: always, never, or only when they are needed. Use the `setHbarPolicy` and `setVbarPolicy` methods to specify the scroll bar policy for the horizontal and vertical scroll bars respectively. Thus, in Example 11–2 the vertical scroll bar will appear, but not the horizontal scroll bar.

Example 11–2 Setting the Horizontal and Vertical Scroll Bar Policies

```
sp.setHbarPolicy(ScrollBarPolicy.NEVER);
sp.setVbarPolicy(ScrollBarPolicy.ALWAYS);
```

As a result, you can only scroll the image vertically, as shown in Figure 11–2.

Figure 11–2 Disabling the Horizontal Scroll Bar

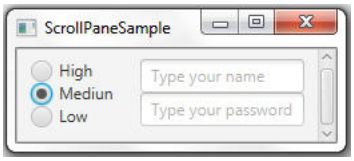


Resizing Components in the Scroll Pane

When designing a UI interface, you might need to resize the components, so that they match the width or height of the scroll pane. Set either the `setFitToWidth` or `setFitToHeight` method to `true` to match a particular dimension.

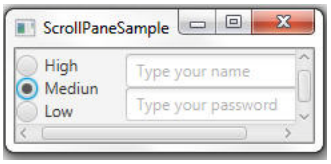
The scroll pane shown in Figure 11–3 contains radio buttons, a text box, and a password box. The size of the content exceeds the predefined size of the scroll pane and a vertical scroll bar appears. However, because the `setFitToWidth` method sets `true` for the scroll pane, the content shrinks in width and never scrolls horizontally.

Figure 11–3 Fitting the Width of the Scroll Pane



By default, both `FIT_TO_WIDTH` and `FIT_TO_HEIGHT` properties are `false`, and the resizable content keeps its original size. If you remove the `setFitToWidth` methods from the code of this application, you will see the output shown in Figure 11–4.

Figure 11–4 Default Properties for Fitting the Content



The `ScrollPane` class enables you to retrieve and set the current, minimum, and maximum values of the contents in the horizontal and vertical directions. Learn how to use them in your applications.

Sample Application with a Scroll Pane

Example 11–3 uses a scroll pane to display a vertical box with images. The `VVALUE` property of the `ScrollPane` class helps to identify the currently displayed image and to render the name of the image file.

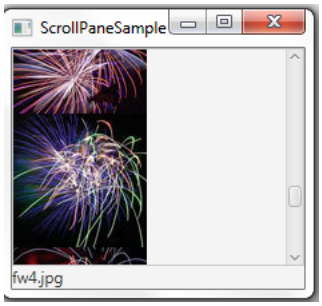
Example 11–3 Using a Scroll Pane to View Images

```
import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ScrollPane;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class Main extends Application {

    final ScrollPane sp = new ScrollPane();
    final Image[] images = new Image[5];
    final ImageView[] pics = new ImageView[5];
    final VBox vb = new VBox();
    final Label fileName = new Label();
    final String [] imageNames = new String [] {"fw1.jpg", "fw2.jpg",
        "fw3.jpg", "fw4.jpg", "fw5.jpg"};

    @Override
    public void start(Stage stage) {
        VBox box = new VBox();
        Scene scene = new Scene(box, 180, 180);
        stage.setScene(scene);
        stage.setTitle("Scroll Pane");
        box.getChildren().addAll(sp, fileName);
        VBox.setVgrow(sp, Priority.ALWAYS);

        fileName.setLayoutX(30);
        fileName.setLayoutY(160);

        for (int i = 0; i < 5; i++) {
            images[i] = new Image(getClass().getResourceAsStream(imageNames[i]));
            pics[i] = new ImageView(images[i]);
            pics[i].setFitWidth(100);
            pics[i].setPreserveRatio(true);
            vb.getChildren().add(pics[i]);
        }

        sp.setVmax(440);
        sp.setPrefSize(115, 150);
        sp.setContent(vb);
        sp.vvalueProperty().addListener((ObservableValue<? extends Number> ov,
            Number old_val, Number new_val) -> {
                fileName.setText(imageNames[(new_val.intValue() - 1)/100]);
        });
        stage.show();
    }

    public static void main(String[] args) {
```

```
        launch(args);
    }
}
```

Compiling and running this application produces the window shown in Figure 11–5.

Figure 11–5 Scrolling Images



The maximum value of the vertical scroll bar is equal to the height of the vertical box. The code fragment shown in Example 11–4 renders the name of the currently displayed image file.

Example 11–4 Tracking the Change of the Scroll Pane's Vertical Value

```
sp.vvalueProperty().addListener((ObservableValue<? extends Number> ov,
    Number old_val, Number new_val) -> {
        fileName.setText(imageNames[(new_val.intValue() - 1)/100]);
});
```

The `ImageView` object limits the image height to 100 pixels. Therefore, when `new_val.intValue() - 1` is divided by 100, the result gives the index of the current image in the `imageNames` array.

In your application, you can also vary minimum and maximum values for vertical and horizontal scroll bars, thus dynamically updating your user interface.

Related API Documentation

- `ScrollPane`
- `ScrollBar`

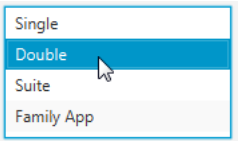
12

List View

In this chapter, you learn how to create lists in your JavaFX applications.

The `ListView` class represents a scrollable list of items. Figure 12–1 shows the list of available accommodation types in a hotel reservation system.

Figure 12–1 Simple List View



You can populate the list by defining its items with the `setItems` method. You can also create a view for the items in the list by applying the `setCellFactory` method.

Creating a List View

The code fragment in Example 12–1 implements the list with the `String` items shown in Figure 12–1.

Example 12–1 Creating a List View Control

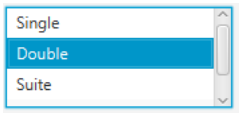
```
ListView<String> list = new ListView<>();
ObservableList<String> items =FXCollections.observableArrayList (
    "Single", "Double", "Suite", "Family App");
list.setItems(items);
```

To alter the size and height of the list view control, use the `setPrefHeight` and `setPrefWidth` methods. Example 12–2 constrains the vertical list to 100 pixels wide by 70 pixels high, which results in the list shown in Figure 12–2.

Example 12–2 Setting Height and Width for a List View

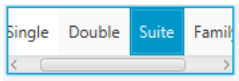
```
list.setPrefWidth(100);
list.setPrefHeight(70);
```

Figure 12–2 Resized Vertical List



You can orient a `ListView` object horizontally by setting the orientation property to `Orientation.HORIZONTAL`. This can be done as follows: `list.setOrientation(Orientation.HORIZONTAL)`. The horizontal list with the same items as in Figure 12–1 is shown in Figure 12–3.

Figure 12–3 Horizontal List View Control



At any time, you can track the selection and focus of the `ListView` object with the `SelectionModel` and `FocusModel` classes. To obtain the current state of each item, use a combination of the following methods:

- `getSelectionModel().getSelectedIndex()` – Returns the index of the currently selected items in a single-selection mode
- `getSelectionModel().getSelectedItem()` – Returns the currently selected item
- `getFocusModel().getFocusedIndex()` – Returns the index of the currently focused item
- `getFocusModel().getFocusedItem()` – Returns the currently focused item

The default `SelectionModel` used when instantiating a `ListView` is an implementation of the `MultipleSelectionModel` abstract class. However, the default value of the `selectionMode` property is `SelectionMode.SINGLE`. To enable multiple selection in a default `ListView` instance, use the following sequence of calls:

```
listView.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
```

Also note that `MultipleSelectionModel` has the `selectedItems` and `selectedIndices` properties, which are both observable lists that can be monitored to detect any multiple selections.

Populating a List View with Data

Example 12–1 shows the simplest way to populate a list view. To enhance your list, you can add data of various types by using the specific extensions of the `ListCell` class, such as `CheckBoxListCell`, `ChoiceBoxListCell`, `ComboBoxListCell`, and `TextFieldListCell`. These classes bring additional functionality to the basic list cell. Implementing cell factories for such classes enables developers to change data directly in the list view.

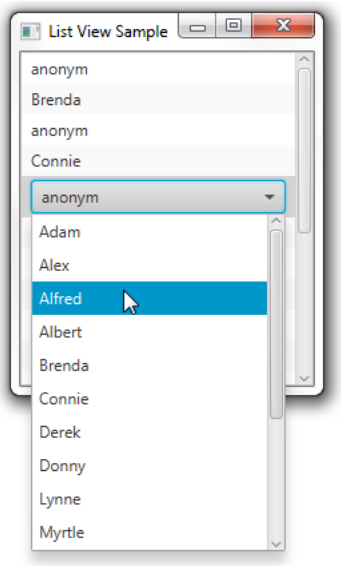
For example, the content of a list cell is not editable by default. However, the `ComboBoxListCell` class draws a combo box inside the list cell. This modification enables users to build a list of names by selecting them from a combo box, as shown in Example 12–3.

Example 12–3 Adding ComboBoxListCell Items to a List View

```
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.cell.ComboBoxListCell;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class ListViewSample extends Application {

    public static final ObservableList names =
        FXCollections.observableArrayList();
    public static final ObservableList data =
        FXCollections.observableArrayList();

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("List View Sample");

        final ListView listView = new ListView(data);
        listView.setPrefSize(200, 250);
        listView.setEditable(true);

        names.addAll(
             "Adam", "Alex", "Alfred", "Albert",
             "Brenda", "Connie", "Derek", "Donny",
             "Lynne", "Myrtle", "Rose", "Rudolph",
             "Tony", "Trudy", "Williams", "Zach"
        );

        for (int i = 0; i < 18; i++) {
            data.add("anonym");
        }

        listView.setItems(data);
        listView.setCellFactory(ComboBoxListCell.forListView(names));

        StackPane root = new StackPane();
        root.getChildren().add(listView);
        primaryStage.setScene(new Scene(root, 200, 250));
        primaryStage.show();
    }
}
```

The bold line in the example, calls the `setCellFactory` method to redefine the implementation of the list cell. When you compile and run this example, it produces the application window shown in Figure 12–4.

Figure 12–4 List View with the Combo Box Cells



Not only the cell factory mechanism enables you to apply the alternative implementations of the list cells, it can help you to totally customize the appearance of the cells.

Customizing the Content of a List View

Study the following application to learn how to generate the list items by using the cell factory. The application shown in Example 12–4 creates a list of color patterns.

Example 12–4 Creating a Cell Factory

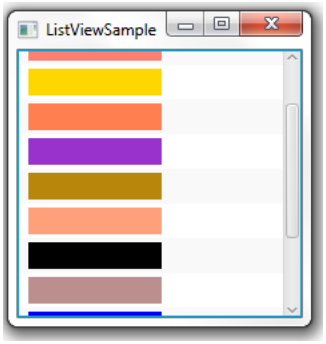
```
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.ListCell;
import javafx.scene.control.ListView;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

public class ListViewSample extends Application {

    ListView<String> list = new ListView<>();
    ObservableList<String> data = FXCollections.observableArrayList(
            "chocolate", "salmon", "gold", "coral", "darkorchid",
            "darkgoldenrod", "lightsalmon", "black", "rosybrown", "blue",
            "blueviolet", "brown");

    @Override
    public void start(Stage stage) {
        VBox box = new VBox();
        Scene scene = new Scene(box, 200, 200);
        stage.setScene(scene);
        stage.setTitle("ListViewSample");
```

```
        box.getChildren().addAll(list);
        VBox.setVgrow(list, Priority.ALWAYS);

        list.setItems(data);

        list.setCellFactory((ListView<String> l) -> new ColorRectCell());

        stage.show();
    }

    static class ColorRectCell extends ListCell<String> {
        @Override
        public void updateItem(String item, boolean empty) {
            super.updateItem(item, empty);
            Rectangle rect = new Rectangle(100, 20);
            if (item != null) {
                rect.setFill(Color.web(item));
                setGraphic(rect);
            }
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

The cell factory produces `ListCell` objects. Every cell is associated with a single data item and renders a single "row" of the list view. The content that the cell represents through the `setGraphic` method can include other controls, text, shapes, or images. In this application, the list cell shows rectangles.

Compiling and running the application produces the window shown in Figure 12–5.

Figure 12–5 List of Color Patterns



You can scroll through the list, selecting and deselecting any of its items. You can also extend this application to fill the text label with the color pattern as shown in the next section.

Processing the List Item Selection

Modify the application code as shown in Example 12–5 to enable processing of the event when a particular list item is selected.

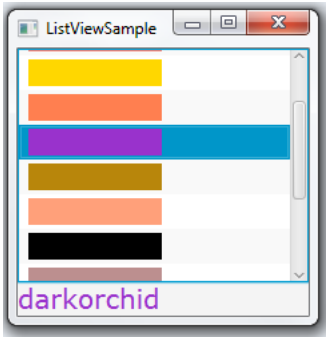
Example 12–5 Processing Events for a List Item

```
import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ListCell;
import javafx.scene.control.ListView;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class ListViewSample extends Application {

    ListView<String> list = new ListView<>();
    ObservableList<String> data = FXCollections.observableArrayList(
            "chocolate", "salmon", "gold", "coral", "darkorchid",
            "darkgoldenrod", "lightsalmon", "black", "rosybrown", "blue",
            "blueviolet", "brown");
    final Label label = new Label();

    @Override
    public void start(Stage stage) {
        VBox box = new VBox();
        Scene scene = new Scene(box, 200, 200);
        stage.setScene(scene);
        stage.setTitle("ListViewSample");
        box.getChildren().addAll(list, label);
        VBox.setVgrow(list, Priority.ALWAYS);

        label.setLayoutX(10);
        label.setLayoutY(115);
        label.setFont(Font.font("Verdana", 20));

        list.setItems(data);

        list.setCellFactory((ListView<String> l) -> new ColorRectCell());

        list.getSelectionModel().selectedItemProperty().addListener(
            (ObservableValue<? extends String> ov, String old_val,
                String new_val) -> {
                    label.setText(new_val);
                    label.setTextFill(Color.web(new_val));
        });
        stage.show();
    }

    static class ColorRectCell extends ListCell<String> {
        @Override
        public void updateItem(String item, boolean empty) {
            super.updateItem(item, empty);
            Rectangle rect = new Rectangle(100, 20);
            if (item != null) {
                rect.setFill(Color.web(item));
                setGraphic(rect);
            } else {
```

```
                setGraphic(null);
            }
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

The `addListener` method called for the `selectedItemProperty` creates a new listener to handle changes of the selected item. If, for instance, the dark orchid item is selected, the label receives the "darkorchid" caption and is filled with the corresponding color. The output of the modified application is shown in Figure 12–6.

Figure 12–6 Selecting a Dark Orchid Color Pattern



Related Documentation

- `ListView`
- `ListCell`
- `ComboBoxListCell`
- Customization of UI Controls

13

Table View

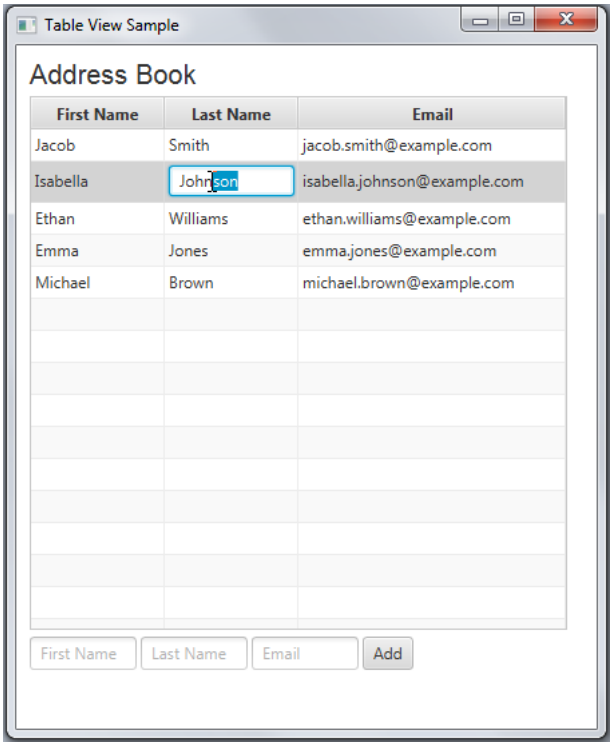
In this chapter, you learn how to perform basic operations with tables in JavaFX applications, such as adding a table, populating the table with data, and editing table rows.

Several classes in the JavaFX SDK API are designed to represent data in a tabular form. The most important classes for creating tables in JavaFX applications are `TableView`, `TableColumn`, and `TableCell`. You can populate a table by implementing the data model and by applying a cell factory.

The table classes provide built-in capabilities to sort data in columns and to resize columns when necessary.

Figure 13–1 shows a typical table representing contact information from an address book.

Figure 13–1 Table Sample



Creating a Table

The code fragment in Example 13–1 creates an empty table with three columns and adds it to the application scene.

Example 13–1 Adding a Table

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class TableViewSample extends Application {

    private final TableView table = new TableView();
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Table View Sample");
        stage.setWidth(300);
        stage.setHeight(500);

        final Label label = new Label("Address Book");
        label.setFont(new Font("Arial", 20));

        table.setEditable(true);

        TableColumn firstNameCol = new TableColumn("First Name");
        TableColumn lastNameCol = new TableColumn("Last Name");
        TableColumn emailCol = new TableColumn("Email");

        table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);

        final VBox vbox = new VBox();
        vbox.setSpacing(5);
        vbox.setPadding(new Insets(10, 0, 0, 10));
        vbox.getChildren().addAll(label, table);

        ((Group) scene.getRoot()).getChildren().addAll(vbox);

        stage.setScene(scene);
        stage.show();
    }
}
```

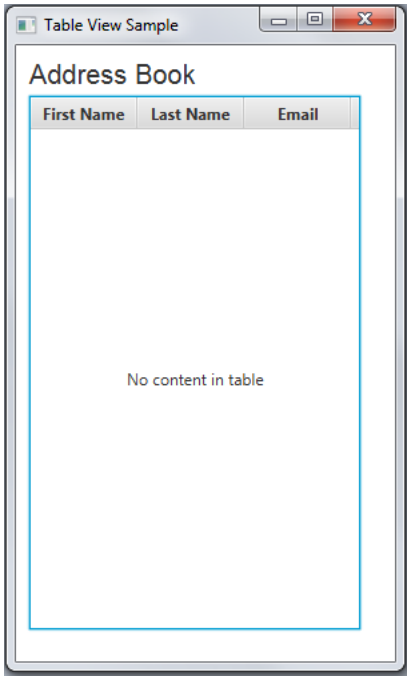
The table control is created by instantiating the `TableView` class. In Example 13–1, it is added to the `VBox` layout container, however, you can add it directly to the application scene.

Example 13–1 defines three columns to store the following information in an address book: a contact's first name and last name, and an email address. The columns are created by using the `TableColumn` class.

The `getColumns` method of the `TableView` class adds the previously created columns to the table. In your applications, you can use this method to dynamically add and remove columns.

Compiling and running this application produces the output shown in Figure 13–2.

Figure 13–2 Table Without Data



You can manage visibility of the columns by calling the `setVisible` method. For example, if the logic of your application requires hiding user email addresses, you can implement this task as follows: `emailCol.setVisible(false)`.

When the structure of your data requires a more complicated representation, you can create nested columns.

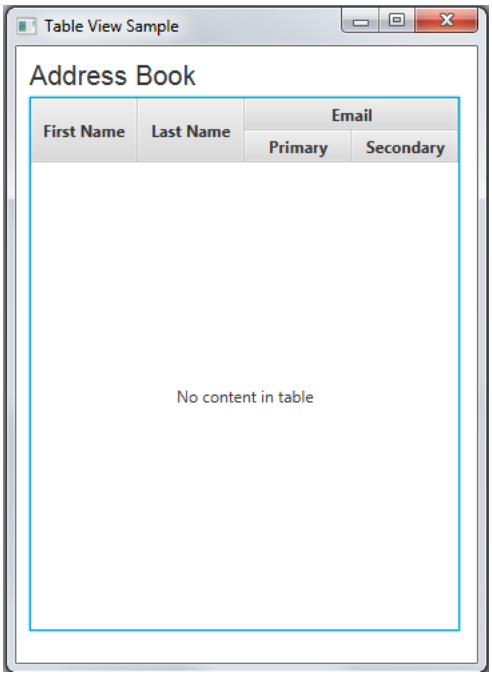
For example, suppose that the contacts in the address book have two email accounts. Then you need two columns to show the primary and the secondary email addresses. Create two subcolumns, and call the `getColumns` method on `emailCol` as shown in Example 13–2.

Example 13–2 Creating Nested Columns

```
TableColumn firstEmailCol = new TableColumn("Primary");
TableColumn secondEmailCol = new TableColumn("Secondary");

emailCol.getColumns().addAll(firstEmailCol, secondEmailCol);
```

After you have added these lines to Example 13–1, and compiled and run the application code, the table appears as shown in Figure 13–3.

Figure 13–3 Table with Nested Columns



Although the table is added to the application, the standard caption "No content in table" appears, because no data is defined. Instead of showing this caption, you can use the `setPlaceholder` method to specify a `Node` object to appear in an empty table.

Defining the Data Model

When you create a table in a JavaFX application, it is a best practice to implement a class that defines the data model and provides methods and fields to further work with the table. Example 13–3 creates the `Person` class to define data in an address book.

Example 13–3 Creating the Person Class

```
public static class Person {
    private final SimpleStringProperty firstName;
    private final SimpleStringProperty lastName;
    private final SimpleStringProperty email;

    private Person(String fName, String lName, String email) {
        this.firstName = new SimpleStringProperty(fName);
        this.lastName = new SimpleStringProperty(lName);
        this.email = new SimpleStringProperty(email);
    }

    public String getFirstName() {
        return firstName.get();
    }
    public void setFirstName(String fName) {
        firstName.set(fName);
    }

    public String getLastName() {
        return lastName.get();
```

```
    }
    public void setLastName(String fName) {
        lastName.set(fName);
    }

    public String getEmail() {
        return email.get();
    }
    public void setEmail(String fName) {
        email.set(fName);
    }

}
```

The `firstName`, `lastName`, and `email` string properties are created to enable the referencing of a particular data element.

Additionally, the `get` and `set` methods are provided for each data element. Thus, for example, the `getFirstName` method returns the value of the `firstName` property, and the `setFirstName` method specifies a value for this property.

When the data model is outlined in the `Person` class, you can create an `ObservableList` array and define as many data rows as you would like to show in your table. The code fragment in Example 13–4 implements this task.

Example 13–4 Defining Table Data in an Observable List

```
final ObservableList<Person> data = FXCollections.observableArrayList(
    new Person("Jacob", "Smith", "jacob.smith@example.com"),
    new Person("Isabella", "Johnson", "isabella.johnson@example.com"),
    new Person("Ethan", "Williams", "ethan.williams@example.com"),
    new Person("Emma", "Jones", "emma.jones@example.com"),
    new Person("Michael", "Brown", "michael.brown@example.com")
);
```

The next step is to associate the data with the table columns. You can do this through the properties defined for each data element, as shown in Example 13–5.

Example 13–5 Setting Data Properties to Columns

```
firstNameCol.setCellValueFactory(
    new PropertyValueFactory<>("firstName")
);
lastNameCol.setCellValueFactory(
    new PropertyValueFactory<>("lastName")
);
emailCol.setCellValueFactory(
    new PropertyValueFactory<>("email")
);
```

The `setCellValueFactory` method specifies a cell factory for each column. The cell factories are implemented by using the `PropertyValueFactory` class, which uses the `firstName`, `lastName`, and `email` properties of the table columns as references to the corresponding methods of the `Person` class.

When the data model is defined, and the data is added and associated with the columns, you can add the data to the table by using the `setItems` method of the `TableView` class: `table.setItems(data)`.

Because the `ObservableList` object enables the tracking of any changes to its elements, the `TableView` content automatically updates whenever the data changes.

Examine the application code shown in Example 13–6.

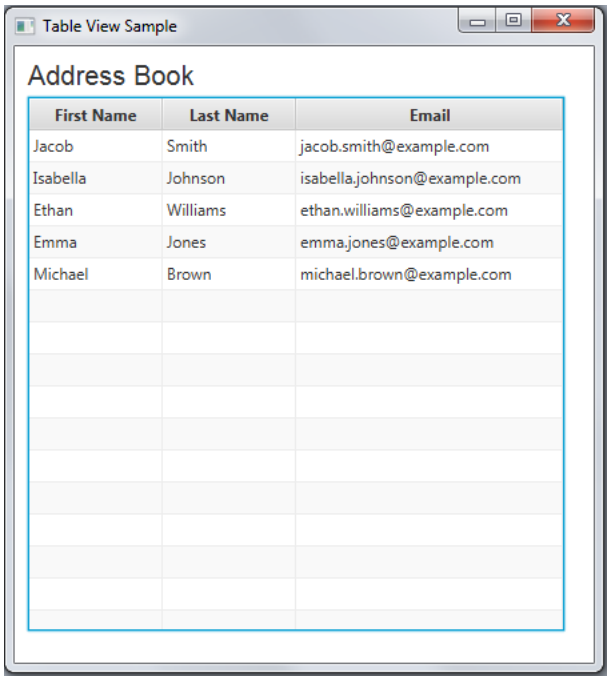
Example 13–6 Creating a Table and Adding Data to It

```
import javafx.application.Application;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class TableViewSample extends Application {

    private final TableView<Person> table = new TableView<>();
    private final ObservableList<Person> data =
        FXCollections.observableArrayList(
            new Person("Jacob", "Smith", "jacob.smith@example.com"),
            new Person("Isabella", "Johnson", "isabella.johnson@example.com"),
            new Person("Ethan", "Williams", "ethan.williams@example.com"),
            new Person("Emma", "Jones", "emma.jones@example.com"),
            new Person("Michael", "Brown", "michael.brown@example.com")
        );

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Table View Sample");
        stage.setWidth(450);
        stage.setHeight(500);

        final Label label = new Label("Address Book");
        label.setFont(new Font("Arial", 20));

        table.setEditable(true);

        TableColumn firstNameCol = new TableColumn("First Name");
        firstNameCol.setMinWidth(100);
        firstNameCol.setCellValueFactory(
                new PropertyValueFactory<>("firstName"));

        TableColumn lastNameCol = new TableColumn("Last Name");
        lastNameCol.setMinWidth(100);
        lastNameCol.setCellValueFactory(
                new PropertyValueFactory<>("lastName"));

        TableColumn emailCol = new TableColumn("Email");
        emailCol.setMinWidth(200);
        emailCol.setCellValueFactory(
```

```
                new PropertyValueFactory<>("email"));

        table.setItems(data);
        table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);

        final VBox vbox = new VBox();
        vbox.setSpacing(5);
        vbox.setPadding(new Insets(10, 0, 0, 10));
        vbox.getChildren().addAll(label, table);

        ((Group) scene.getRoot()).getChildren().addAll(vbox);

        stage.setScene(scene);
        stage.show();
    }

    public static class Person {

        private final SimpleStringProperty firstName;
        private final SimpleStringProperty lastName;
        private final SimpleStringProperty email;

        private Person(String fName, String lName, String email) {
            this.firstName = new SimpleStringProperty(fName);
            this.lastName = new SimpleStringProperty(lName);
            this.email = new SimpleStringProperty(email);
        }

        public String getFirstName() {
            return firstName.get();
        }

        public void setFirstName(String fName) {
            firstName.set(fName);
        }

        public String getLastName() {
            return lastName.get();
        }

        public void setLastName(String fName) {
            lastName.set(fName);
        }

        public String getEmail() {
            return email.get();
        }

        public void setEmail(String fName) {
            email.set(fName);
        }
    }
}
```

When you compile and run this application code, the table shown in Figure 13–4 appears.

Figure 13–4 Table Populated with Data



Adding New Rows

The table in Figure 13–4 contains five rows of data, which cannot be modified so far.

You can use text fields to enter new values into the First Name, Last Name, and Email columns. The Text Field control enables your application to receive text input from a user. Example 13–7 creates three text fields, defines the prompt text for each field, and creates the Add button.

Example 13–7 Using Text Fields to Enter New Items in the Table

```
final TextField addFirstName = new TextField();
addFirstName.setPromptText("First Name");
addFirstName.setMaxWidth(firstNameCol.getPrefWidth());

final TextField addLastName = new TextField();
addLastName.setMaxWidth(lastNameCol.getPrefWidth());
addLastName.setPromptText("Last Name");

final TextField addEmail = new TextField();
addEmail.setMaxWidth(emailCol.getPrefWidth());
addEmail.setPromptText("Email");

final Button addButton = new Button("Add");
addButton.setOnAction((ActionEvent e) -> {
    data.add(new Person(
        addFirstName.getText(),
        addLastName.getText(),
        addEmail.getText()
    ));
    addFirstName.clear();
    addLastName.clear();
    addEmail.clear();
});
```

When a user clicks the Add button, the values entered in the text fields are included in a `Person` constructor and added to the `data` observable list. Thus, the new entry with contact information appears in the table.

Examine the application code shown in Example 13–8.

Example 13–8 Table with the Text Fields to Enter New Items

```
import javafx.application.Application;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class TableViewSample extends Application {

    private final TableView<Person> table = new TableView<>();
    private final ObservableList<Person> data =
            FXCollections.observableArrayList(
            new Person("Jacob", "Smith", "jacob.smith@example.com"),
            new Person("Isabella", "Johnson", "isabella.johnson@example.com"),
            new Person("Ethan", "Williams", "ethan.williams@example.com"),
            new Person("Emma", "Jones", "emma.jones@example.com"),
            new Person("Michael", "Brown", "michael.brown@example.com"));
    final HBox hb = new HBox();

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Table View Sample");
        stage.setWidth(450);
        stage.setHeight(550);

        final Label label = new Label("Address Book");
        label.setFont(new Font("Arial", 20));

        table.setEditable(true);

        TableColumn firstNameCol = new TableColumn("First Name");
        firstNameCol.setMinWidth(100);
        firstNameCol.setCellValueFactory(
                new PropertyValueFactory<>("firstName"));
```

```
        TableColumn lastNameCol = new TableColumn("Last Name");
        lastNameCol.setMinWidth(100);
        lastNameCol.setCellValueFactory(
                new PropertyValueFactory<>("lastName"));

        TableColumn emailCol = new TableColumn("Email");
        emailCol.setMinWidth(200);
        emailCol.setCellValueFactory(
                new PropertyValueFactory<>("email"));

        table.setItems(data);
        table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);

        final TextField addFirstName = new TextField();
        addFirstName.setPromptText("First Name");
        addFirstName.setMaxWidth(firstNameCol.getPrefWidth());
        final TextField addLastName = new TextField();
        addLastName.setMaxWidth(lastNameCol.getPrefWidth());
        addLastName.setPromptText("Last Name");
        final TextField addEmail = new TextField();
        addEmail.setMaxWidth(emailCol.getPrefWidth());
        addEmail.setPromptText("Email");

        final Button addButton = new Button("Add");
        addButton.setOnAction((ActionEvent e) -> {
            data.add(new Person(
                    addFirstName.getText(),
                    addLastName.getText(),
                    addEmail.getText()));
            addFirstName.clear();
            addLastName.clear();
            addEmail.clear();
        });

        hb.getChildren().addAll(addFirstName, addLastName, addEmail, addButton);
        hb.setSpacing(3);

        final VBox vbox = new VBox();
        vbox.setSpacing(5);
        vbox.setPadding(new Insets(10, 0, 0, 10));
        vbox.getChildren().addAll(label, table, hb);

        ((Group) scene.getRoot()).getChildren().addAll(vbox);

        stage.setScene(scene);
        stage.show();
    }

    public static class Person {

        private final SimpleStringProperty firstName;
        private final SimpleStringProperty lastName;
        private final SimpleStringProperty email;

        private Person(String fName, String lName, String email) {
            this.firstName = new SimpleStringProperty(fName);
            this.lastName = new SimpleStringProperty(lName);
            this.email = new SimpleStringProperty(email);
        }
```

```
        public String getFirstName() {
            return firstName.get();
        }

        public void setFirstName(String fName) {
            firstName.set(fName);
        }

        public String getLastName() {
            return lastName.get();
        }

        public void setLastName(String fName) {
            lastName.set(fName);
        }

        public String getEmail() {
            return email.get();
        }

        public void setEmail(String fName) {
            email.set(fName);
        }
    }
}
```

This application does not provide any filters to check if, for example, an email address was entered in an incorrect format. You can provide such functionality when you develop your own application.

The current implementation also does not check to determine if the empty values are entered. If no values are provided, clicking the Add button inserts an empty row in the table.

Figure 13–5 demonstrates how a user adds a new row of data.

Figure 13–5 Adding Contact Information to the Address Book

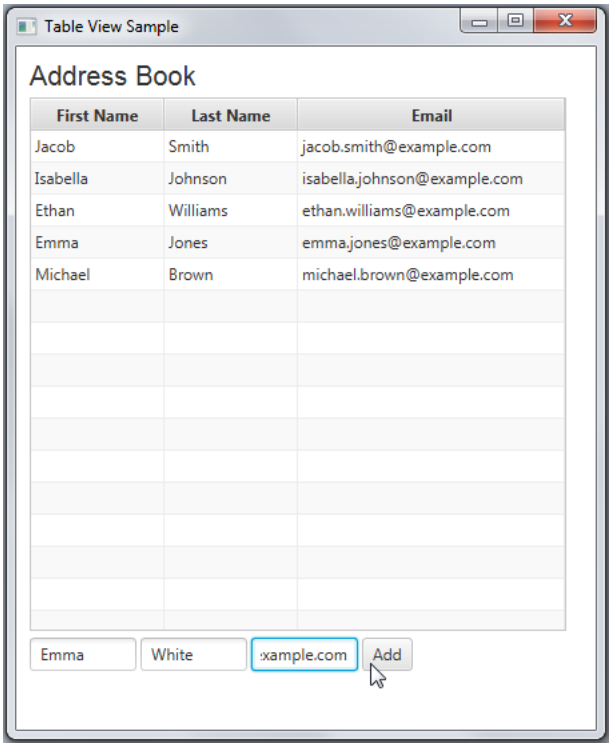
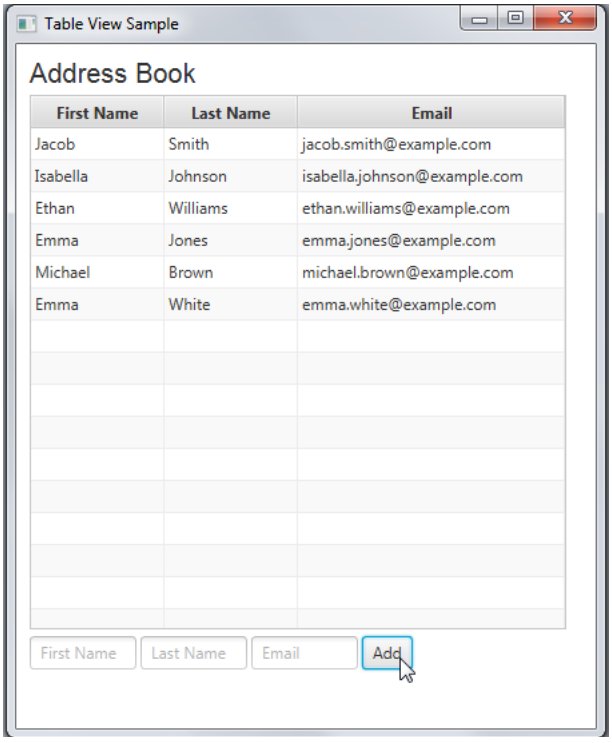


Figure 13–6 shows the table after the Add button is clicked. The contact details of Emma White now appear in the table.

Figure 13–6 Newly Added Entry



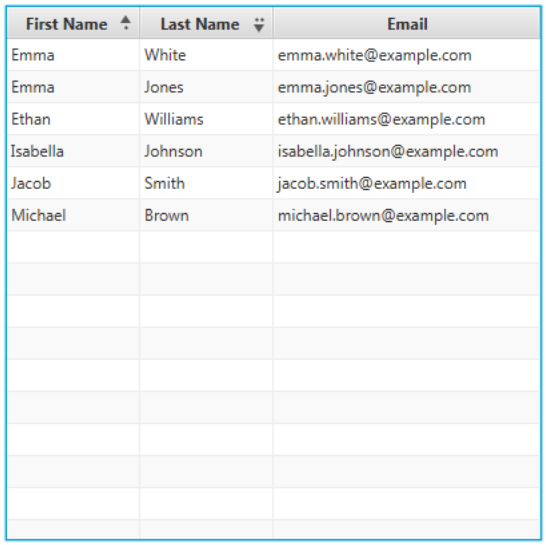
Sorting Data in Columns

The `TableView` class provides built-in capabilities to sort data in columns. Users can alter the order of data by clicking column headers. The first click enables the ascending sorting order, the second click enables descending sorting order, and the third click disables sorting. By default, no sorting is applied.

Users can sort multiple columns in a table and specify the priority of each column in the sort operation. To sort multiple columns, the user presses the Shift key while clicking the header of each column to be sorted.

In Figure 13–7, an ascending sort order is applied to the first names, while last names are sorted in a descending order. Note that the first column has priority over the second column.

Figure 13–7 Sorting Multiple Columns



First Name	Last Name	Email
Emma	White	emma.white@example.com
Emma	Jones	emma.jones@example.com
Ethan	Williams	ethan.williams@example.com
Isabella	Johnson	isabella.johnson@example.com
Jacob	Smith	jacob.smith@example.com
Michael	Brown	michael.brown@example.com

As the application developer, you can set sorting preferences for each column in your application by applying the `setSortType` method. You can specify both ascending and descending type. For example, use the following code line to set the descending type of sorting for the emailCol column:

```
emailCol.setSortType(TableColumn.SortType.DESCENDING);.
```

You can also specify which columns to sort by adding and removing `TableColumn` instances from the `TableView.sortOrder` observable list. The order of columns in this list represents the sort priority (for example, the zero item has higher priority than the first item).

To prohibit sorting of data, call the `setSortable(false)` method on the column.

Editing Data in the Table

The `TableView` class not only renders tabular data, but it also provides capabilities to edit it. Use the `setEditable` method to enable editing of the table content.

Use the `setCellFactory` method to reimplement the table cell as a text field with the help of the `TextFieldTableCell` class. The `setOnEditCommit` method processes editing and assigns the updated value to the corresponding table cell. Example 13–9 shows

how to apply these methods to process cell editing in the First Name, Last Name, and Email columns.

Example 13–9 Implementing Cell Editing

```
firstNameCol.setCellFactory(TextFieldTableCell.<Person>forTableColumn());
firstNameCol.setOnEditCommit(
    (CellEditEvent<Person, String> t) -> {
        ((Person) t.getTableView().getItems().get(
            t.getTablePosition().getRow())
            ).setFirstName(t.getNewValue());
});

lastNameCol.setCellFactory(TextFieldTableCell.<Person>forTableColumn());
lastNameCol.setOnEditCommit(
    (CellEditEvent<Person, String> t) -> {
        ((Person) t.getTableView().getItems().get(
            t.getTablePosition().getRow())
            ).setLastName(t.getNewValue());
});

emailCol.setCellFactory(TextFieldTableCell.<Person>forTableColumn());
emailCol.setOnEditCommit(
(CellEditEvent<Person, String> t) -> {
    ((Person) t.getTableView().getItems().get(
        t.getTablePosition().getRow())
        ).setEmail(t.getNewValue());
});
```

The complete code of the application shown in Example 13–10.

Example 13–10 TableViewSample with Enabled Cell Editing

```
import javafx.application.Application;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableColumn.CellEditEvent;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.control.cell.TextFieldTableCell;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class TableViewSample extends Application {

    private final TableView<Person> table = new TableView<>();
    private final ObservableList<Person> data =
            FXCollections.observableArrayList(
            new Person("Jacob", "Smith", "jacob.smith@example.com"),
```

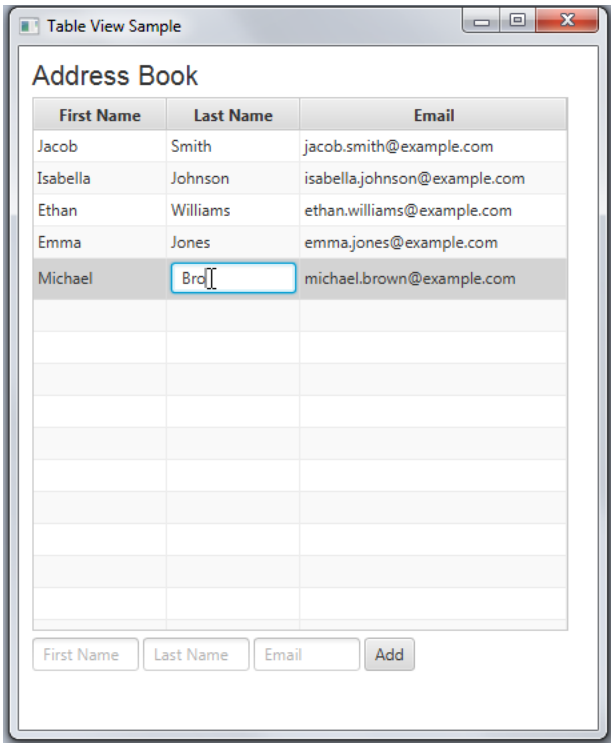
```
        new Person("Isabella", "Johnson", "isabella.johnson@example.com"),
        new Person("Ethan", "Williams", "ethan.williams@example.com"),
        new Person("Emma", "Jones", "emma.jones@example.com"),
        new Person("Michael", "Brown", "michael.brown@example.com"));
final HBox hb = new HBox();

public static void main(String[] args) {
    launch(args);
}

@Override
public void start(Stage stage) {
    Scene scene = new Scene(new Group());
    stage.setTitle("Table View Sample");
    stage.setWidth(450);
    stage.setHeight(550);

    final Label label = new Label("Address Book");
    label.setFont(new Font("Arial", 20));

    table.setEditable(true);

    TableColumn<Person, String> firstNameCol =
        new TableColumn<>("First Name");
    firstNameCol.setMinWidth(100);
    firstNameCol.setCellValueFactory(
        new PropertyValueFactory<>("firstName"));

    firstNameCol.setCellFactory(TextFieldTableCell.<Person>forTableColumn());
    firstNameCol.setOnEditCommit(
        (CellEditEvent<Person, String> t) -> {
            ((Person) t.getTableView().getItems().get(
                    t.getTablePosition().getRow())
                    ).setFirstName(t.getNewValue());
    });


    TableColumn<Person, String> lastNameCol =
        new TableColumn<>("Last Name");
    lastNameCol.setMinWidth(100);
    lastNameCol.setCellValueFactory(
        new PropertyValueFactory<>("lastName"));
   lastNameCol.setCellFactory(TextFieldTableCell.<Person>forTableColumn());
   lastNameCol.setOnEditCommit(
        (CellEditEvent<Person, String> t) -> {
            ((Person) t.getTableView().getItems().get(
                    t.getTablePosition().getRow())
                    ).setLastName(t.getNewValue());
    });

    TableColumn<Person, String> emailCol = new TableColumn<>("Email");
    emailCol.setMinWidth(200);
    emailCol.setCellValueFactory(
        new PropertyValueFactory<>("email"));
   emailCol.setCellFactory(TextFieldTableCell.<Person>forTableColumn());
    emailCol.setOnEditCommit(
        (CellEditEvent<Person, String> t) -> {
            ((Person) t.getTableView().getItems().get(
                    t.getTablePosition().getRow())
                    ).setEmail(t.getNewValue());
```

```
        });

        table.setItems(data);
        table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);

        final TextField addFirstName = new TextField();
        addFirstName.setPromptText("First Name");
        addFirstName.setMaxWidth(firstNameCol.getPrefWidth());
        final TextField addLastName = new TextField();
        addLastName.setMaxWidth(lastNameCol.getPrefWidth());
        addLastName.setPromptText("Last Name");
        final TextField addEmail = new TextField();
        addEmail.setMaxWidth(emailCol.getPrefWidth());
        addEmail.setPromptText("Email");

        final Button addButton = new Button("Add");
        addButton.setOnAction((ActionEvent e) -> {
            data.add(new Person(
                    addFirstName.getText(),
                    addLastName.getText(),
                    addEmail.getText()));
            addFirstName.clear();
            addLastName.clear();
            addEmail.clear();
        });

        hb.getChildren().addAll(addFirstName, addLastName, addEmail, addButton);
        hb.setSpacing(3);

        final VBox vbox = new VBox();
        vbox.setSpacing(5);
        vbox.setPadding(new Insets(10, 0, 0, 10));
        vbox.getChildren().addAll(label, table, hb);

        ((Group) scene.getRoot()).getChildren().addAll(vbox);

        stage.setScene(scene);
        stage.show();
    }

    public static class Person {

        private final SimpleStringProperty firstName;
        private final SimpleStringProperty lastName;
        private final SimpleStringProperty email;

        private Person(String fName, String lName, String email) {
            this.firstName = new SimpleStringProperty(fName);
            this.lastName = new SimpleStringProperty(lName);
            this.email = new SimpleStringProperty(email);
        }

        public String getFirstName() {
            return firstName.get();
        }

        public void setFirstName(String fName) {
            firstName.set(fName);
        }
```

```
        public String getLastName() {
            return lastName.get();
        }

        public void setLastName(String fName) {
            lastName.set(fName);
        }

        public String getEmail() {
            return email.get();
        }

        public void setEmail(String fName) {
            email.set(fName);
        }
    }
}
```

In Figure 13–8, the user is editing the last name of Michael Brown. To edit a table cell, the user enters the new value in the cell, and then presses the Enter key. The cell is not modified until the Enter key is pressed. This behavior is determined by the implementation of the `TextField` class.

Figure 13–8 Editing a Table Cell



Note that the default implementation of the `TextField` control requires that users press the Enter key to commit the edit. You can redefine the `TextField` behavior to commit the edit on the focus change, which is an expected user experience. Try the modified code in to implement such an alternative behavior.

Example 13–11 Alternative Solution Of Cell Editing

```
import javafx.application.Application;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TableCell;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableColumn.CellEditEvent;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;
import javafx.util.Callback;

public class TableViewSample extends Application {

    private final TableView<Person> table = new TableView<>();
    private final ObservableList<Person> data =
            FXCollections.observableArrayList(
            new Person("Jacob", "Smith", "jacob.smith@example.com"),
            new Person("Isabella", "Johnson", "isabella.johnson@example.com"),
            new Person("Ethan", "Williams", "ethan.williams@example.com"),
            new Person("Emma", "Jones", "emma.jones@example.com"),
            new Person("Michael", "Brown", "michael.brown@example.com"));
    final HBox hb = new HBox();

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Table View Sample");
        stage.setWidth(450);
        stage.setHeight(550);

        final Label label = new Label("Address Book");
        label.setFont(new Font("Arial", 20));

        table.setEditable(true);

        Callback<TableColumn<Person, String>,
            TableCell<Person, String>> cellFactory
                = (TableColumn<Person, String> p) -> new EditingCell();

        TableColumn<Person, String> firstNameCol =
            new TableColumn<>("First Name");
        TableColumn<Person, String> lastNameCol =
            new TableColumn<>("Last Name");
```

```
TableColumn<Person, String> emailCol =
    new TableColumn<>("Email");

firstNameCol.setMinWidth(100);
firstNameCol.setCellValueFactory(
    new PropertyValueFactory<>("firstName"));
firstNameCol.setCellFactory(cellFactory);
firstNameCol.setOnEditCommit(
    (CellEditEvent<Person, String> t) -> {
        ((Person) t.getTableView().getItems().get(
                t.getTablePosition().getRow())
                ).setFirstName(t.getNewValue());
});


lastNameCol.setMinWidth(100);
lastNameCol.setCellValueFactory(
    new PropertyValueFactory<>("lastName"));
lastNameCol.setCellFactory(cellFactory);
lastNameCol.setOnEditCommit(
    (CellEditEvent<Person, String> t) -> {
        ((Person) t.getTableView().getItems().get(
                t.getTablePosition().getRow())
                ).setLastName(t.getNewValue());
});

emailCol.setMinWidth(200);
emailCol.setCellValueFactory(
    new PropertyValueFactory<>("email"));
emailCol.setCellFactory(cellFactory);
emailCol.setOnEditCommit(
    (CellEditEvent<Person, String> t) -> {
        ((Person) t.getTableView().getItems().get(
                t.getTablePosition().getRow())
                ).setEmail(t.getNewValue());
});

table.setItems(data);
table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);

final TextField addFirstName = new TextField();
addFirstName.setPromptText("First Name");
addFirstName.setMaxWidth(firstNameCol.getPrefWidth());
final TextField addLastName = new TextField();
addLastName.setMaxWidth(lastNameCol.getPrefWidth());
addLastName.setPromptText("Last Name");
final TextField addEmail = new TextField();
addEmail.setMaxWidth(emailCol.getPrefWidth());
addEmail.setPromptText("Email");

final Button addButton = new Button("Add");
addButton.setOnAction((ActionEvent e) -> {
    data.add(new Person(
            addFirstName.getText(),
            addLastName.getText(),
            addEmail.getText()));
    addFirstName.clear();
    addLastName.clear();
    addEmail.clear();
});
```

```
        hb.getChildren().addAll(addFirstName, addLastName, addEmail, addButton);
        hb.setSpacing(3);

        final VBox vbox = new VBox();
        vbox.setSpacing(5);
        vbox.setPadding(new Insets(10, 0, 0, 10));
        vbox.getChildren().addAll(label, table, hb);

        ((Group) scene.getRoot()).getChildren().addAll(vbox);

        stage.setScene(scene);
        stage.show();
    }

    public static class Person {

        private final SimpleStringProperty firstName;
        private final SimpleStringProperty lastName;
        private final SimpleStringProperty email;

        private Person(String fName, String lName, String email) {
            this.firstName = new SimpleStringProperty(fName);
            this.lastName = new SimpleStringProperty(lName);
            this.email = new SimpleStringProperty(email);
        }

        public String getFirstName() {
            return firstName.get();
        }

        public void setFirstName(String fName) {
            firstName.set(fName);
        }

        public String getLastName() {
            return lastName.get();
        }

        public void setLastName(String fName) {
            lastName.set(fName);
        }

        public String getEmail() {
            return email.get();
        }

        public void setEmail(String fName) {
            email.set(fName);
        }
    }

    class EditingCell extends TableCell<Person, String> {

        private TextField textField;

        public EditingCell() {
        }

        @Override
```

```
        public void startEdit() {
            if (!isEmpty()) {
                super.startEdit();
                createTextField();
                setText(null);
                setGraphic(textField);
                textField.selectAll();
            }
        }

        @Override
        public void cancelEdit() {
            super.cancelEdit();

            setText((String) getItem());
            setGraphic(null);
        }

        @Override
        public void updateItem(String item, boolean empty) {
            super.updateItem(item, empty);

            if (empty) {
                setText(null);
                setGraphic(null);
            } else {
                if (isEditing()) {
                    if (textField != null) {
                        textField.setText(getString());
                    }
                    setText(null);
                    setGraphic(textField);
                } else {
                    setText(getString());
                    setGraphic(null);
                }
            }
        }

        private void createTextField() {
            textField = new TextField(getString());
            textField.setMinWidth(this.getWidth() - this.getGraphicTextGap()* 2);
            textField.focusedProperty().addListener(
                (ObservableValue<? extends Boolean> arg0,
                Boolean arg1, Boolean arg2) -> {
                    if (!arg2) {
                        commitEdit(textField.getText());
                    }
            });
        }

        private String getString() {
            return getItem() == null ? "" : getItem().toString();
        }
    }
}
```

Note that this approach might become redundant in future releases as the `TextFieldTableCell` implementation is being evolved to provide better user experience.

Adding Maps of Data to the Table

You can add the `Map` data to the table. Use the `MapValueFactory` class as shown in Example 13–12 to display the map of student IDs in the table.

Example 13–12 Adding Map Data to the Table

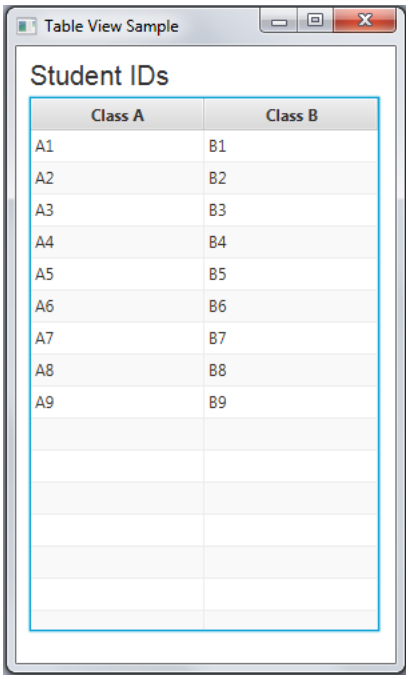
```
import java.util.HashMap;
import java.util.Map;
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TableCell;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.MapValueFactory;
import javafx.scene.control.cell.TextFieldTableCell;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;
import javafx.util.Callback;
import javafx.util.StringConverter;

public class TableViewSample extends Application {

    public static final String Column1MapKey = "A";
    public static final String Column2MapKey = "B";

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Table View Sample");
        stage.setWidth(300);
        stage.setHeight(500);

        final Label label = new Label("Student IDs");
        label.setFont(new Font("Arial", 20));

        TableColumn<Map, String> firstDataColumn = new TableColumn<>("Class A");
        TableColumn<Map, String> secondDataColumn = new TableColumn<>("Class B");

        firstDataColumn.setCellValueFactory(new MapValueFactory(Column1MapKey));
        firstDataColumn.setMinWidth(130);
        secondDataColumn.setCellValueFactory(new MapValueFactory(Column2MapKey));
        secondDataColumn.setMinWidth(130);

        TableView tableView = new TableView<>(generateDataInMap());

        tableView.setEditable(true);
        tableView.getSelectionModel().setCellSelectionEnabled(true);
        tableView.getColumns().setAll(firstDataColumn, secondDataColumn);
        Callback<TableColumn<Map, String>, TableCell<Map, String>>
```

```
            cellFactoryForMap = (TableColumn<Map, String> p) ->
                new TextFieldTableCell(new StringConverter() {
                    @Override
                        public String toString(Object t) {
                        return t.toString();
                    }
                    @Override
                    public Object fromString(String string) {
                        return string;
                    }
            });
        firstDataColumn.setCellFactory(cellFactoryForMap);
        secondDataColumn.setCellFactory(cellFactoryForMap);

        final VBox vbox = new VBox();

        vbox.setSpacing(5);
        vbox.setPadding(new Insets(10, 0, 0, 10));
        vbox.getChildren().addAll(label, tableView);

        ((Group) scene.getRoot()).getChildren().addAll(vbox);

        stage.setScene(scene);

        stage.show();
    }

    private ObservableList<Map> generateDataInMap() {
        int max = 10;
        ObservableList<Map> allData = FXCollections.observableArrayList();
        for (int i = 1; i < max; i++) {
            Map<String, String> dataRow = new HashMap<>();

            String value1 = "A" + i;
            String value2 = "B" + i;

            dataRow.put(Column1MapKey, value1);
            dataRow.put(Column2MapKey, value2);

            allData.add(dataRow);
        }
        return allData;
    }
}
```

The `MapValueFactory` class implements the `Callback` interface, and it is designed specifically to be used within cell factories of table columns. In Example 13–12, the dataRow hash map presents a single row in the `TableView` object. The map has two `String` keys: Column1MapKey and Column2MapKey to map the corresponding values in the first and second columns. The `setCellValueFactory` method called for the table columns populates them with data matching with a particular key, so that the first column contains values that correspond to the "A" key and second column contains the values that correspond to the "B" key.

When you compile and run this application, it produces the output shown in Figure 13–9.

Figure 13–9 TableVIew with Map Data



`TreeTableView` is the further extension of the tabular data controls in JavaFX. Refer to the Tree Table View chapter for more information about this UI control.

Related API Documentation

- `TableView`
- `TableColumn`
- `TableCell`
- `TextField`
- `TextFieldTableCell`
- `TreeTableView`
- `MapValueFactory`
- `Button`

14

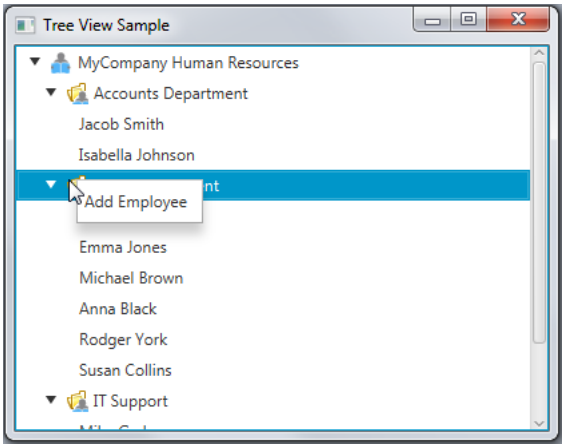
Tree View

In this chapter you can learn how to build tree structures in your JavaFX application, add items to the tree views, process events, and customize the tree cells by implementing and applying cell factories.

The `TreeView` class of the `javafx.scene.control` package provides a view of hierarchical structures. In each tree the highest object in the hierarchy is called the "root." The root contains several child items, which can have children as well. An item without children is called "leaf."

Figure 14–1 shows a screen capture of an application with a tree view.

Figure 14–1 Tree View Sample



Creating Tree Views

When you build a tree structure in your JavaFX applications, you typically need to instantiate the `TreeView` class, define several `TreeItem` objects, make one of the tree items the root, add the root to the tree view and other tree items to the root.

You can accompany each tree item with a graphical icon by using the corresponding constructor of the `TreeItem` class or by calling the `setGraphic` method. The recommended size for icons is 16x16, but in fact, any `Node` object can be set as the icon and it will be fully interactive.

Example 14–1 is an implementation of a simple tree view with the root and five leaves.

Example 14–1 Creating a Tree View

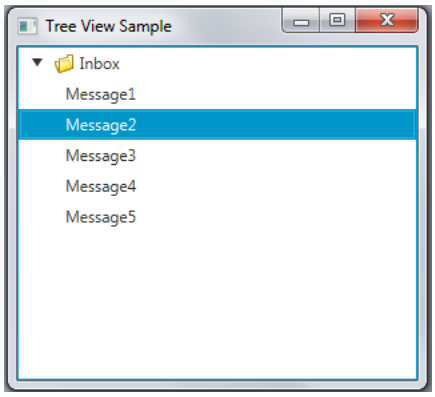
```
import javafx.application.Application;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class TreeViewSample extends Application {

    private final Node rootIcon = new ImageView(
        new Image(getClass().getResourceAsStream("folder_16.png"))
    );

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Tree View Sample");

        TreeItem<String> rootItem = new TreeItem<> ("Inbox", rootIcon);
        rootItem.setExpanded(true);
        for (int i = 1; i < 6; i++) {
            TreeItem<String> item = new TreeItem<> ("Message" + i);
            rootItem.getChildren().add(item);
        }
        TreeView<String> tree = new TreeView<> (rootItem);
        StackPane root = new StackPane();
        root.getChildren().add(tree);
        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();
    }
}
```

All the tree items created within the `for` loop are added to the root item by calling the `getChildren` and `add` methods. You can also use the `addAll` method instead of the `add` method to include all the previously created tree items at once.

You can specify the root of the tree within the constructor of the `TreeView` class when you create a new `TreeView` object as shown in Example 14–1, or you can set it by calling the `setRoot` method of the `TreeView` class.

The `setExpanded` method called on the root item defines the initial appearance of the tree view item. By default, all `TreeItem` instances are collapsed, and must be manually expanded if required. Pass the `true` value to the `setExpanded` method, so that the root tree item looks expanded when the application starts, as shown in Figure 14–2.

Figure 14–2 Tree View with Five Tree Items



Example 14–1 creates a simple tree view with the `String` items. However, a tree structure can contain items of different types. Use the following generic notation of the `TreeItem` constructor to define application-specific data represented by a tree item: `TreeItem<T> (T value)`. The `T` value can specify any object, such as UI controls or custom components.

Unlike the `TreeView` class, the `TreeItem` class does not extend the `Node` class. Therefore, you cannot apply any visual effects or add menus to the tree items. Use the cell factory mechanism to overcome this obstacle and define as much custom behavior for the tree items as your application requires.

Implementing Cell Factories

The cell factory mechanism is used for generating `TreeCell` instances to represent a single `TreeItem` in the `TreeView`. Using cell factories is particularly helpful when your application operates with an excessive amount of data that is changed dynamically or added on demand.

Consider an application that visualizes human resources data of a given company, and enables users to modify employee details and add new employees.

Example 14–2 creates the `Employee` class and arranges employees in groups according to their departments.

Example 14–2 Creating a Model of the Human Resources Tree View

```
import java.util.Arrays;
import java.util.List;
import javafx.application.Application;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

import javafx.beans.property.SimpleStringProperty;
import javafx.scene.layout.VBox;

public class TreeViewSample extends Application {
```

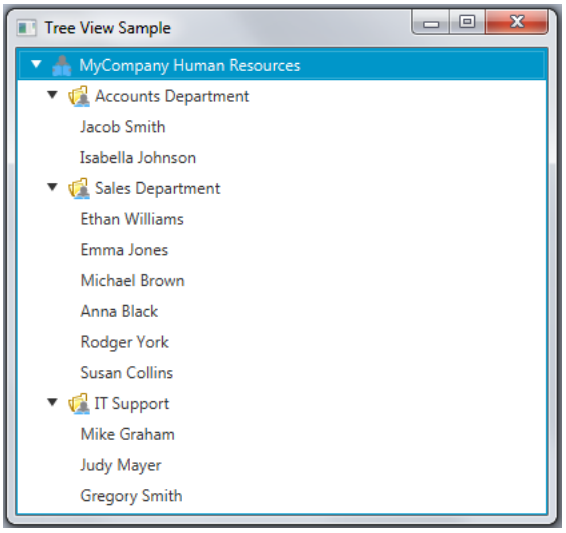
```
private final Node rootIcon =
    new ImageView(new Image(getClass().getResourceAsStream("root.png")));
private final Image depIcon =
    new Image(getClass().getResourceAsStream("department.png"));
List<Employee> employees = Arrays.<Employee>asList(
        new Employee("Jacob Smith", "Accounts Department"),
        new Employee("Isabella Johnson", "Accounts Department"),
        new Employee("Ethan Williams", "Sales Department"),
        new Employee("Emma Jones", "Sales Department"),
        new Employee("Michael Brown", "Sales Department"),
        new Employee("Anna Black", "Sales Department"),
        new Employee("Rodger York", "Sales Department"),
        new Employee("Susan Collins", "Sales Department"),
        new Employee("Mike Graham", "IT Support"),
        new Employee("Judy Mayer", "IT Support"),
        new Employee("Gregory Smith", "IT Support"));
TreeItem<String> rootNode;

public static void main(String[] args) {
    launch(args);
}

public TreeViewSample() {
    this.rootNode = new TreeItem<>("MyCompany Human Resources", rootIcon);
}

@Override
public void start(Stage stage) {
    rootNode.setExpanded(true);
    for (Employee employee : employees) {
        TreeItem<String> empLeaf = new TreeItem<>(employee.getName());
        boolean found = false;
        for (TreeItem<String> depNode : rootNode.getChildren()) {
            if (depNode.getValue().contentEquals(employee.getDepartment())){
                depNode.getChildren().add(empLeaf);
                found = true;
                break;
            }
        }
        if (!found) {
            TreeItem<String> depNode = new TreeItem<>(
                employee.getDepartment(),
                new ImageView(depIcon)
            );
            rootNode.getChildren().add(depNode);
            depNode.getChildren().add(empLeaf);
        }
    }

    stage.setTitle("Tree View Sample");
    VBox box = new VBox();
    final Scene scene = new Scene(box, 400, 300);
    scene.setFill(Color.LIGHTGRAY);

    TreeView<String> treeView = new TreeView<>(rootNode);

    box.getChildren().add(treeView);
    stage.setScene(scene);
    stage.show();
}
```

```
    public static class Employee {

        private final SimpleStringProperty name;
        private final SimpleStringProperty department;

        private Employee(String name, String department) {
            this.name = new SimpleStringProperty(name);
            this.department = new SimpleStringProperty(department);
        }

        public String getName() {
            return name.get();
        }

        public void setName(String fName) {
            name.set(fName);
        }

        public String getDepartment() {
            return department.get();
        }

        public void setDepartment(String fName) {
            department.set(fName);
        }
    }
}
```

Each `Employee` object in Example 14–2 has two properties: `name` and `department`. `TreeItem` objects corresponding to the employees are referred as tree leaves, whereas the tree items corresponding to the departments are referred to as tree items with children. The name of the new department to be created is retrieved from an `Employee` object by calling the `getDepartment` method.

When you compile and run this application, it creates the window shown in Figure 14–3.

Figure 14–3 List of Employees in the Tree View Sample Application



With Example 14–2, you can preview the tree view and its items, but you cannot change the existing items or add any new items. Example 14–3 shows a modified version of the application with the cell factory implemented. The modified application enables you to change the name of an employee.

Example 14–3 Implementing a Cell Factory

```
import java.util.Arrays;
import java.util.List;
import javafx.application.Application;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.control.TreeCell;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.beans.property.SimpleStringProperty;
import javafx.scene.layout.VBox;

public class TreeViewSample extends Application {

    private final Node rootIcon =
        new ImageView(new Image(getClass().getResourceAsStream("root.png")));
    private final Image depIcon =
        new Image(getClass().getResourceAsStream("department.png"));
    List<Employee> employees = Arrays.<Employee>asList(
            new Employee("Jacob Smith", "Accounts Department"),
            new Employee("Isabella Johnson", "Accounts Department"),
            new Employee("Ethan Williams", "Sales Department"),
            new Employee("Emma Jones", "Sales Department"),
            new Employee("Michael Brown", "Sales Department"),
            new Employee("Anna Black", "Sales Department"),
            new Employee("Rodger York", "Sales Department"),
            new Employee("Susan Collins", "Sales Department"),
            new Employee("Mike Graham", "IT Support"),
            new Employee("Judy Mayer", "IT Support"),
            new Employee("Gregory Smith", "IT Support"));
    TreeItem<String> rootNode;

    public static void main(String[] args) {
        Application.launch(args);
    }

    public TreeViewSample() {
        this.rootNode = new TreeItem<>("MyCompany Human Resources", rootIcon);
    }

    @Override
    public void start(Stage stage) {
        rootNode.setExpanded(true);
        for (Employee employee : employees) {
            TreeItem<String> empLeaf = new TreeItem<>(employee.getName());
            boolean found = false;
            for (TreeItem<String> depNode : rootNode.getChildren()) {
```

```
                if (depNode.getValue().contentEquals(employee.getDepartment())){
                    depNode.getChildren().add(empLeaf);
                    found = true;
                    break;
                }
            }
            if (!found) {
                TreeItem<String> depNode = new TreeItem<>(
                    employee.getDepartment(),
                    new ImageView(depIcon)
                );
                rootNode.getChildren().add(depNode);
                depNode.getChildren().add(empLeaf);
            }
        }

        stage.setTitle("Tree View Sample");
        VBox box = new VBox();
        final Scene scene = new Scene(box, 400, 300);
        scene.setFill(Color.LIGHTGRAY);

        TreeView<String> treeView = new TreeView<>(rootNode);
        treeView.setEditable(true);
        treeView.setCellFactory((TreeView<String> p) ->
            new TextFieldTreeCellImpl());

        box.getChildren().add(treeView);
        stage.setScene(scene);
        stage.show();
    }

    private final class TextFieldTreeCellImpl extends TreeCell<String> {

        private TextField textField;

        public TextFieldTreeCellImpl() {
        }

        @Override
        public void startEdit() {
            super.startEdit();

            if (textField == null) {
                createTextField();
            }
            setText(null);
            setGraphic(textField);
            textField.selectAll();
        }

        @Override
        public void cancelEdit() {
            super.cancelEdit();
            setText((String) getItem());
            setGraphic(getTreeItem().getGraphic());
        }

        @Override
        public void updateItem(String item, boolean empty) {
            super.updateItem(item, empty);
```

```
            if (empty) {
                setText(null);
                setGraphic(null);
            } else {
                if (isEditing()) {
                    if (textField != null) {
                        textField.setText(getString());
                    }
                    setText(null);
                    setGraphic(textField);
                } else {
                    setText(getString());
                    setGraphic(getTreeItem().getGraphic());
                }
            }
        }

        private void createTextField() {
            textField = new TextField(getString());
            textField.setOnKeyReleased((KeyEvent t) -> {
                if (t.getCode() == KeyCode.ENTER) {
                    commitEdit(textField.getText());
                } else if (t.getCode() == KeyCode.ESCAPE) {
                    cancelEdit();
                }
            });
        }

        private String getString() {
            return getItem() == null ? "" : getItem().toString();
        }
    }

    public static class Employee {

        private final SimpleStringProperty name;
        private final SimpleStringProperty department;

        private Employee(String name, String department) {
            this.name = new SimpleStringProperty(name);
            this.department = new SimpleStringProperty(department);
        }

        public String getName() {
            return name.get();
        }

        public void setName(String fName) {
            name.set(fName);
        }

        public String getDepartment() {
            return department.get();
        }

        public void setDepartment(String fName) {
            department.set(fName);
        }
    }
```

```
}
```

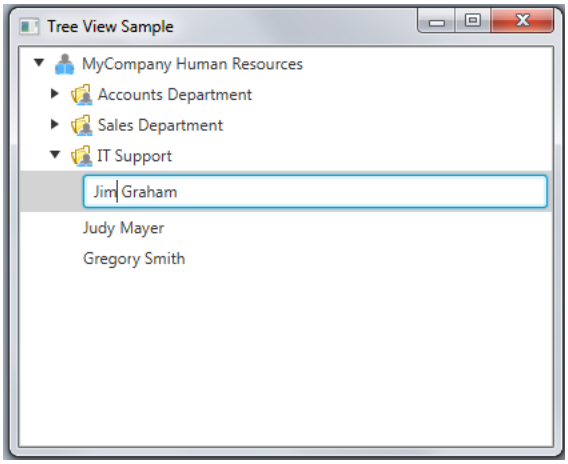
The `setCellFactory` method called on the `treeView` object overrides the `TreeCell` implementation and redefines the tree items as specified in the `TextFieldTreeCellImpl` class.

The `TextFieldTreeCellImpl` class creates a `TextField` object for each tree item and provides the methods to process editing events.

Note that you must explicitly call the `setEditable(true)` method on the tree view to enable editing all its items.

Compile and run the application in Example 14–3. Then try to click the employees in the tree and change their names. Figure 14–4 captures the moment of editing a tree item in the IT Support department.

Figure 14–4 Changing an Employee Name



Adding New Tree Items on Demand

Modify the Tree View Sample application so that a human resources representative can add new employees. Use the bold code lines of Example 14–4 for your reference. These lines add a context menu to the tree items that correspond to the departments. When the Add Employee menu item is selected, the new tree item is added as a leaf to the current department.

Use the `isLeaf` method to distinguish between department tree items and employee tree items.

Example 14–4 Adding New Tree Items

```
import java.util.Arrays;
import java.util.List;
import javafx.application.Application;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.control.TreeCell;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
```

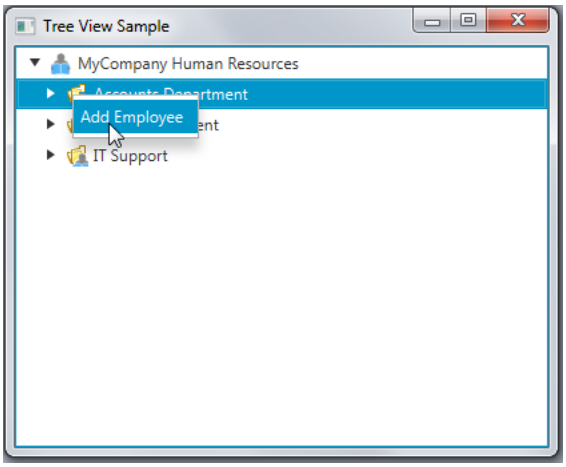
```
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.beans.property.SimpleStringProperty;
import javafx.event.ActionEvent;
import javafx.scene.control.ContextMenu;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.VBox;

public class TreeViewSample extends Application {

    private final Node rootIcon =
        new ImageView(new Image(getClass().getResourceAsStream("root.png")));
    private final Image depIcon =
        new Image(getClass().getResourceAsStream("department.png"));
    List<Employee> employees = Arrays.<Employee>asList(
            new Employee("Jacob Smith", "Accounts Department"),
            new Employee("Isabella Johnson", "Accounts Department"),
            new Employee("Ethan Williams", "Sales Department"),
            new Employee("Emma Jones", "Sales Department"),
            new Employee("Michael Brown", "Sales Department"),
            new Employee("Anna Black", "Sales Department"),
            new Employee("Rodger York", "Sales Department"),
            new Employee("Susan Collins", "Sales Department"),
            new Employee("Mike Graham", "IT Support"),
            new Employee("Judy Mayer", "IT Support"),
            new Employee("Gregory Smith", "IT Support"));
    TreeItem<String> rootNode =
        new TreeItem<>("MyCompany Human Resources", rootIcon);

    public static void main(String[] args) {
        Application.launch(args);
    }

    @Override
    public void start(Stage stage) {
        rootNode.setExpanded(true);
        for (Employee employee : employees) {
            TreeItem<String> empLeaf = new TreeItem<>(employee.getName());
            boolean found = false;
            for (TreeItem<String> depNode : rootNode.getChildren()) {
                if (depNode.getValue().contentEquals(employee.getDepartment())){
                    depNode.getChildren().add(empLeaf);
                    found = true;
                    break;
                }
            }
            if (!found) {
                TreeItem depNode = new TreeItem(employee.getDepartment(),
                    new ImageView(depIcon)
                );
                rootNode.getChildren().add(depNode);
                depNode.getChildren().add(empLeaf);
            }
        }

        stage.setTitle("Tree View Sample");
        VBox box = new VBox();
        final Scene scene = new Scene(box, 400, 300);
```

```
        scene.setFill(Color.LIGHTGRAY);

        TreeView<String> treeView = new TreeView<>(rootNode);
        treeView.setEditable(true);
        treeView.setCellFactory((TreeView<String> p) ->
            new TextFieldTreeCellImpl());

        box.getChildren().add(treeView);
        stage.setScene(scene);
        stage.show();
    }

    private final class TextFieldTreeCellImpl extends TreeCell<String> {

        private TextField textField;
        private final ContextMenu addMenu = new ContextMenu();

        public TextFieldTreeCellImpl() {
            MenuItem addMenuItem = new MenuItem("Add Employee");
            addMenu.getItems().add(addMenuItem);
            addMenuItem.setOnAction((ActionEvent t) -> {
                TreeItem newEmployee =
                    new TreeItem<>("New Employee");
                getTreeItem().getChildren().add(newEmployee);
            });
        }

        @Override
        public void startEdit() {
            super.startEdit();

            if (textField == null) {
                createTextField();
            }
            setText(null);
            setGraphic(textField);
            textField.selectAll();
        }

        @Override
        public void cancelEdit() {
            super.cancelEdit();

            setText((String) getItem());
            setGraphic(getTreeItem().getGraphic());
        }

        @Override
        public void updateItem(String item, boolean empty) {
            super.updateItem(item, empty);

            if (empty) {
                setText(null);
                setGraphic(null);
            } else {
                if (isEditing()) {
                    if (textField != null) {
                        textField.setText(getString());
                    }
                    setText(null);
```

```
                    setGraphic(textField);
                } else {
                    setText(getString());
                    setGraphic(getTreeItem().getGraphic());
                    if (
                        !getTreeItem().isLeaf()&&getTreeItem().getParent()!= null
                    ){
                        setContextMenu(addMenu);
                    }
                }
            }
        }

        private void createTextField() {
            textField = new TextField(getString());
            textField.setOnKeyReleased((KeyEvent t) -> {
                if (t.getCode() == KeyCode.ENTER) {
                    commitEdit(textField.getText());
                } else if (t.getCode() == KeyCode.ESCAPE) {
                    cancelEdit();
                }
            });

        }

        private String getString() {
            return getItem() == null ? "" : getItem().toString();
        }
    }

    public static class Employee {

        private final SimpleStringProperty name;
        private final SimpleStringProperty department;

        private Employee(String name, String department) {
            this.name = new SimpleStringProperty(name);
            this.department = new SimpleStringProperty(department);
        }

        public String getName() {
            return name.get();
        }

        public void setName(String fName) {
            name.set(fName);
        }

        public String getDepartment() {
            return department.get();
        }

        public void setDepartment(String fName) {
            department.set(fName);
        }
    }
}
```

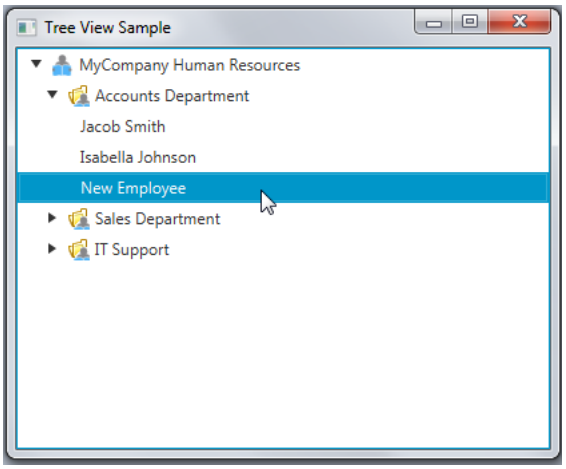
Compile and run the application. Then select a department in the tree structure and right-click it. The context menu appears, as shown in Figure 14–5.

Figure 14–5 Context Menu for Adding New Employees



When you select the Add Employee menu item from the context menu, the new record is added to the current department. Figure 14–6 shows a new tree item added to the Accounts Department.

Figure 14–6 Newly Added Employee



Because editing is enabled for the tree items, you can change the default "New Employee" value to the appropriate name.

Using Tree Cell Editors

You can use the following tree cell editors available in the API: `CheckBoxTreeCell`, `ChoiceBoxTreeCell`, `ComboBoxTreeCell`, `TextFieldTreeCell`. There classes extend the `TreeCell` implementation to render a particular control inside the cell.

Example 14–5 demonstrates the use of the `CheckBoxTreeCell` class in the UI that builds a hierarchical structure of checkboxes.

Example 14–5 Using the CheckBoxTreeCell Class

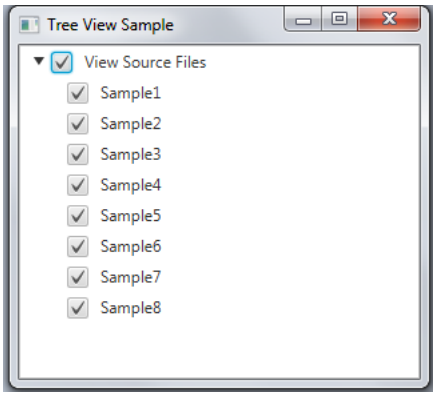
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
```

```
import javafx.scene.control.cell.CheckBoxTreeCell;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class TreeViewSample extends Application {

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Tree View Sample");

        CheckBoxTreeItem<String> rootItem =
            new CheckBoxTreeItem<>("View Source Files");
        rootItem.setExpanded(true);

        final TreeView tree = new TreeView(rootItem);
        tree.setEditable(true);

        tree.setCellFactory(CheckBoxTreeCell.<String>forTreeView());
        for (int i = 0; i < 8; i++) {
            final CheckBoxTreeItem<String> checkBoxTreeItem =
                new CheckBoxTreeItem<>("Sample" + (i+1));
                    rootItem.getChildren().add(checkBoxTreeItem);
        }

        tree.setRoot(rootItem);
        tree.setShowRoot(true);

        StackPane root = new StackPane();
        root.getChildren().add(tree);
        primaryStage.setScene(new Scene(root, 300, 250));
        primaryStage.show();
    }
}
```

Example 14–5 builds the tree view by using the `CheckBoxTreeItem` class instead of the TreeItem. The `CheckBoxTreeItem` class was specifically designed to support the selected, unselected, and indeterminate states in tree structures. A `CheckBoxTreeItem` instance can be independent or dependent. If a `CheckBoxTreeItem` instance is independent, any changes to its selection state do not impact its parent and child `CheckBoxTreeItem` instances. By default, all the `ChechBoxTreeItem` instances are dependent.

Compile and run Example 14–5, then select the View Source Files item. You should see the output shown in Figure 14–7, where all the child items are selected.

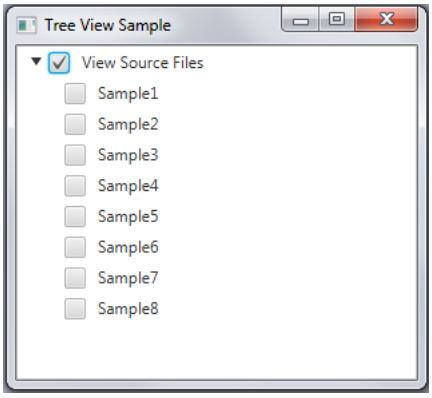
Figure 14–7 Dependent CheckBoxTreeItem



To make a `CheckBoxTreeItem` instance independent, use the `setIndependent` method: `rootItem.setIndependent(true);`.

When you run the TreeViewSample application, its behavior should change as shown in Figure 14–8.

Figure 14–8 Independent CheckBoxTreeItem



The `TreeTableView` control provides additional capabilities to present tree structures in table forms. See the Tree Table View chapter for more information about these controls.

Related Documentation

- `TreeView`
- `TreeItem`
- `TreeCell`
- `TreeTableView`
- `Cell`
- `TextField`
- `CheckBoxTreeCell`
- `CheckBoxTreeItem`
- Customization of UI Controls

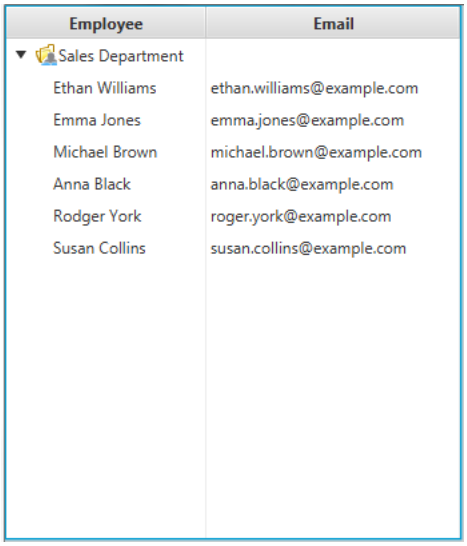
15

Tree Table View

This chapter describes the `TreeTableView` user interface component, which is a control designed to help you to visualize an unlimited hierarchy of data, presented in columns.

The `TreeTableView` component has much in common with the `TreeView` and `TableView` controls: it combines and extends some aspects of their functionality.

Figure 15–1 Sample of a TreeTableView Component



Creating a TreeTableView control

A basic implementation of a `TreeTableView` component in your application can be done by the following steps:

1. Create tree items.
2. Create the root element.
3. Add the tree items to the root element.
4. Create one or several columns.
5. Define cell content.
6. Create a tree table view.

7. Assign columns to the tree table view.

In some cases, you can omit or hide the root element. Example 15–1 implements these steps to create a simple tree table view control.

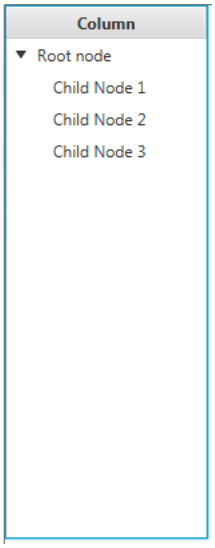
Example 15–1 TreeTableView with One Column

```
import javafx.application.Application;
import javafx.beans.property.ReadOnlyStringWrapper;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.TreeTableColumn;
import javafx.scene.control.TreeTableColumn.CellDataFeatures;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeTableView;
import javafx.stage.Stage;

public class TreeTableViewSample extends Application {

    public static void main(String[] args) {
        Application.launch(args);
    }

    @Override
    public void start(Stage stage) {
        stage.setTitle("Tree Table View Samples");
        final Scene scene = new Scene(new Group(), 200, 400);
        Group sceneRoot = (Group)scene.getRoot();

        //Creating tree items
        final TreeItem<String> childNode1 = new TreeItem<>("Child Node 1");
        final TreeItem<String> childNode2 = new TreeItem<>("Child Node 2");
        final TreeItem<String> childNode3 = new TreeItem<>("Child Node 3");

        //Creating the root element
        final TreeItem<String> root = new TreeItem<>("Root node");
        root.setExpanded(true);

        //Adding tree items to the root
        root.getChildren().setAll(childNode1, childNode2, childNode3);

        //Creating a column
        TreeTableColumn<String,String> column = new TreeTableColumn<>("Column");
        column.setPrefWidth(150);

        //Defining cell content
        column.setCellValueFactory((CellDataFeatures<String, String> p) ->
            new ReadOnlyStringWrapper(p.getValue().getValue()));

        //Creating a tree table view
        final TreeTableView<String> treeTableView = new TreeTableView<>(root);
        treeTableView.getColumns().add(column);
        treeTableView.setPrefWidth(152);
        treeTableView.setShowRoot(true);
        sceneRoot.getChildren().add(treeTableView);
        stage.setScene(scene);
        stage.show();
    }
}
```

The cell factory defined for the column specifies the content to be placed into this column for each tree item.

When you compile and run Example 15–1, it produces the output shown in Figure 15–2.

Figure 15–2 Simple TreeTableView Component



Example 15–1 creates a simplified tree view component with one column, whereas in your application you often operate with the extended set of data. The next section shows you how to represent the personal data of the Sales Department employees from the Tree View use case.

Adding Several Columns

Extend the use case described in the Tree View chapter and provide more information about each employee of the Sales Department. Example 15–2 shows you how to add columns.

Example 15–2 Creating a TreeTableView Component With Two Columns

```
import java.util.Arrays;
import java.util.List;
import javafx.application.Application;
import javafx.beans.property.ReadOnlyStringWrapper;
import javafx.beans.property.SimpleStringProperty;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.TreeTableColumn;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeTableView;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class TreeTableViewSample extends Application {

    List<Employee> employees = Arrays.<Employee>asList(
```

```
        new Employee("Ethan Williams", "ethan.williams@example.com"),
        new Employee("Emma Jones", "emma.jones@example.com"),
        new Employee("Michael Brown", "michael.brown@example.com"),
        new Employee("Anna Black", "anna.black@example.com"),
        new Employee("Rodger York", "roger.york@example.com"),
        new Employee("Susan Collins", "susan.collins@example.com"));

    private final ImageView depIcon = new ImageView (
            new Image(getClass().getResourceAsStream("department.png"))
    );

    final TreeItem<Employee> root =
        new TreeItem<>(new Employee("Sales Department", ""), depIcon);
    public static void main(String[] args) {
        Application.launch(TreeTableViewSample.class, args);
    }

    @Override
    public void start(Stage stage) {
        root.setExpanded(true);
        employees.stream().forEach((employee) -> {
            root.getChildren().add(new TreeItem<>(employee));
        });
        stage.setTitle("Tree Table View Sample");
        final Scene scene = new Scene(new Group(), 400, 400);
        scene.setFill(Color.LIGHTGRAY);
        Group sceneRoot = (Group) scene.getRoot();

        TreeTableColumn<Employee, String> empColumn =
            new TreeTableColumn<>("Employee");
        empColumn.setPrefWidth(150);
        empColumn.setCellValueFactory(
            (TreeTableColumn.CellDataFeatures<Employee, String> param) ->
            new ReadOnlyStringWrapper(param.getValue().getValue().getName())
        );

        TreeTableColumn<Employee, String> emailColumn =
            new TreeTableColumn<>("Email");
        emailColumn.setPrefWidth(190);
        emailColumn.setCellValueFactory(
            (TreeTableColumn.CellDataFeatures<Employee, String> param) ->
            new ReadOnlyStringWrapper(param.getValue().getValue().getEmail())
        );

        TreeTableView<Employee> treeTableView = new TreeTableView<>(root);
        treeTableView.getColumns().setAll(empColumn, emailColumn);
        sceneRoot.getChildren().add(treeTableView);
        stage.setScene(scene);
        stage.show();
    }

    public class Employee {

        private SimpleStringProperty name;
        private SimpleStringProperty email;
        public SimpleStringProperty nameProperty() {
            if (name == null) {
                name = new SimpleStringProperty(this, "name");
            }
            return name;
```

```
        }
        public SimpleStringProperty emailProperty() {
            if (email == null) {
                email = new SimpleStringProperty(this, "email");
            }
            return email;
        }
        private Employee(String name, String email) {
            this.name = new SimpleStringProperty(name);
            this.email = new SimpleStringProperty(email);
        }
        public String getName() {
            return name.get();
        }
        public void setName(String fName) {
            name.set(fName);
        }
        public String getEmail() {
            return email.get();
        }
        public void setEmail(String fName) {
            email.set(fName);
        }
    }
}
```

In Example 15–2, you modify the structure of data in the `Employee` class and define two properties corresponding to an employee's name and email address. The `TreeTableView` component provides two columns to represent them, respectively.

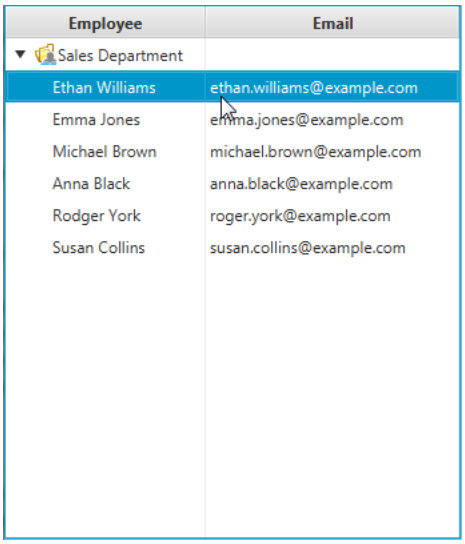
Inspect the code fragment shown in Example 15–3. It creates cell factories for two columns. The `setCellValueFactory` methods define `TreeItem` content for each column, so that the Employee column contains the `Employee.name` property values, and the Email column contains the `Employee.email` property values.

Example 15–3 Implementing Cell Factories for Employee and Email Columns

```
        //Cell factory for the data the Empoyee column
        empColumn.setCellValueFactory(
            (TreeTableColumn.CellDataFeatures<Employee, String> param) ->
            new ReadOnlyStringWrapper(param.getValue().getValue().getName())
        );
        //Cell factory for the data in the Email column
        emailColumn.setCellValueFactory(
            (TreeTableColumn.CellDataFeatures<Employee, String> param) ->
            new ReadOnlyStringWrapper(param.getValue().getValue().getEmail())
        );
```

When you compile and run the application from Example 15–2, the output shown in Figure 15–3 appears.

Figure 15–3 TreeTableView Component with Employee and Email Columns

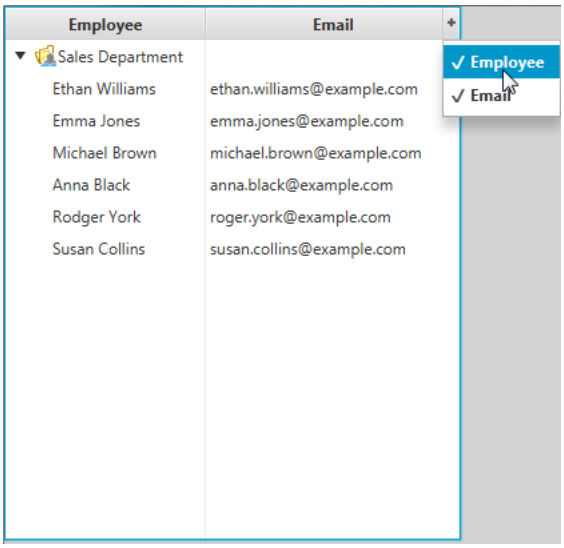


The default style of the tree table forces all tree elements to be expanded and all table columns to be shown. However, you can change these and other default settings by using the corresponding properties and methods of the `TreeTableView` and `TreeTableColumn` classes.

Altering Visual Appearance

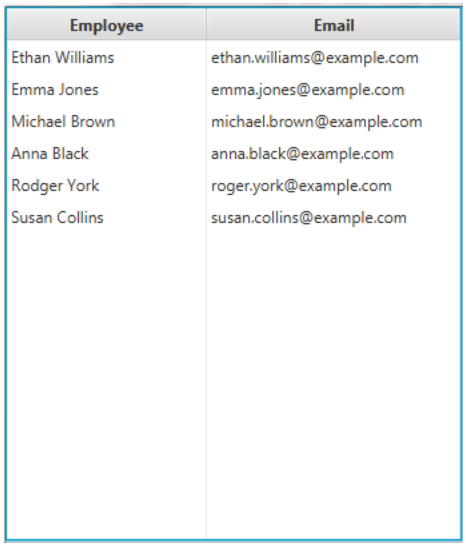
You can enable showing the table menu button, so that users of your application will be able to toggle the visibility of the table columns. Call the `treeTableView.setTableMenuButtonVisible(true);` method for the `treeTableView` object. The method adds the "+" button to the table header. Figure 15–4 captures the moment when a user hides the Employee column.

Figure 15–4 TreeTableView With the Enabled Table Menu Button



You can also control the visibility of the root tree item by using the `setShowRoot` method of the `TreeTableView` class: `treeTableView.setShowRoot(false);`. The `false` value, passed to it, hides the root as shown in Figure 15–5.

Figure 15–5 TreeTableView Control Without the Root Tree Item



You can sort data in columns by clicking the column headers and switching the sorting modes from ascending to descending and vice versa. In addition to the built-in sorting support, data sorting is available through the `TreeTableColumn` and `TreeTableView` properties listed in Example 15–4.

Example 15–4 Sorting Mode Settings

```
//Setting the descending mode of sorting for the Email column
emailColumn.setSortType(TreeTableColumn.SortType.DESCENDING);
//Setting the ascending mode of sorting for the Email column
emailColumn.setSortType(TreeTableColumn.SortType.ASCENDING);
//Applying the sorting mode to all tree items
treeTableView.setSortMode(TreeSortMode.ALL_DESCENDANTS);
//Applying the sorting mode only to the first-level nodes
treeTableView.setSortMode(TreeSortMode.ONLY_FIRST_LEVEL);
```

Managing Selection Mode

The default selection model implemented for the `TreeTableView` class is defined in the `MultipleSelectionModel` abstract class that specifies the default value as `SelectionMode.SINGLE`. To enable multiple selection of the tree items and cells, use a combination of the `setSelectionModel` and `setCellSelectionEnabled` methods. See Table 15–1 for more information about available values and the resulting behavior.

Table 15–1 Selection Model Settings

setSelectionModel	setCellSelectionEnabled	Behavior
`SelectionMode.SINGLE`	`false`	Enables selection of a single row in the table.

Table 15–1 (Cont.) Selection Model Settings

setSelectionModel	setCellSelectionEnabled	Behavior
`SelectionMode.SINGLE`	`true`	Enables selection of a single cell in the table.
`SelectionMode.MULTUPLE`	`false`	Enables selection of several rows in the table.
`SelectionMode.MULTIPLE`	`true`	Enables selection of several cells in several rows.

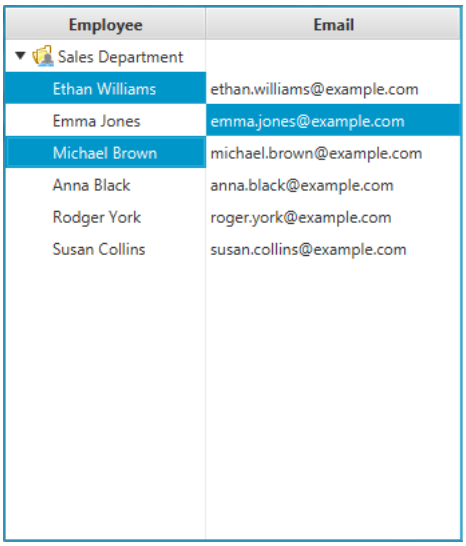
For example, you can implement a selection of several cells in several rows for the `TreeTableView` component as shown in Example 15–5.

Example 15–5 Enabling Multiple Selection for the Cells

```
treeTableView.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
treeeTableView.getSelectionModel().setCellSelectionEnabled(true);
```

Figure 15–6 shows the cells that you can select after the lines in Example 15–5 are added to the code of the TreeTableViewSample application.

Figure 15–6 Multiple Selection of Table Cells



Because the `TreeTableView` control encapsulates all features of table view and tree view, you might want to see the Table View and Tree View chapters for more information about additional capabilities.

Related Documentation

- `TreeView`
- `TreeItem`
- `TreeCell`
- `Cell`
- `TableView`
- `TableColumn`

- `TreeTableView`
- `TreeTableColumn`
- `TreeTableRow`
- `TreeTableCell`
- Customization of UI Controls

16

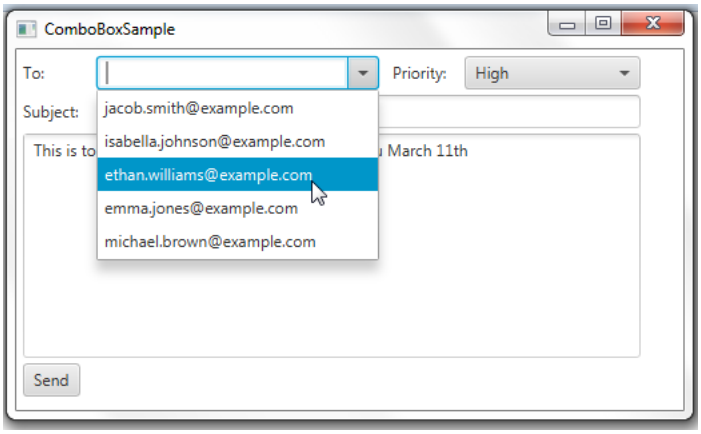
Combo Box

This chapter explains how to use combo boxes in your JavaFX application. It discusses editable and uneditable combo boxes, teaches you how to track changes in the editable combo boxes and handle events on them, and explains how to use cell factories to alter the default implementation of a combo box.

A combo box is a typical element of a user interface that enables users to choose one of several options. A combo box is helpful when the number of items to show exceeds some limit, because it can add scrolling to the drop down list, unlike a choice box. If the number of items does not exceed a certain limit, developers can decide whether a combo box or a choice box better suits their needs.

You can create a combo box in the JavaFX application by using the `ComboBox` class of the JavaFX API. Figure 16–1 shows an application with two combo boxes.

Figure 16–1 Application with Two Combo Boxes



Creating Combo Boxes

When creating a combo box, you must instantiate the `ComboBox` class and define the items as an observable list, just like other UI controls such as `ChoiceBox`, `ListView`, and `TableView`. Example 16–1 sets the items within a constructor.

Example 16–1 Creating a Combo Box with an Observable List

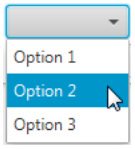
```
ObservableList<String> options =
    FXCollections.observableArrayList(
```

```
        "Option 1",
        "Option 2",
        "Option 3"
    );
final ComboBox comboBox = new ComboBox(options);
```

Another possibility is to create a combo box by using an empty constructor and call the `setItems` method on it, as follows: `comboBox.setItems(options);`

When the combo box is added to the application scene, it appears in the user interface as shown in Figure 16–2.

Figure 16–2 Combo Box with Three Items



At any time, you can supplement the list of items with new values. Example 16–2 implements this task by adding three more items to the comboBox control.

Example 16–2 Adding Items to a Combo Box

```
comboBox.getItems().addAll(
    "Option 4",
    "Option 5",
    "Option 6"
);
```

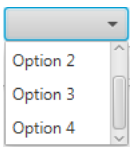
The `ComboBox` class provides handy properties and methods to use with combo boxes.

You can use the `setValue` method to specify the item selected in the combo box. When you call the `setValue` method on the `ComboBox` object, the selected item of the `selectionModel` property changes to this value even if the value is not in the combo box items list. If the items list then changes to include this value, the corresponding item becomes selected.

Similarly, you can obtain the value of the selected item by calling the `getValue` method. When a user selects an item, the selected item of the `selectionModel` property and the combo box `value` property are both updated to the new value.

You can also restrict the number of visible rows in the `ComboBox` drop down list when it is displayed. The following code line enables the display of three items for the comboBox control: `comboBox.setVisibleRowCount(3)` As the result of calling this method, the number of visible rows is limited to three, and a scroll bar appears (as shown in Figure 16–3).

Figure 16–3 Setting the Number of Visible Rows for a Combo Box



Although the `ComboBox` class has a generic notation and enables users to populate it with items of various types, do not use `Node` (or any subclass) as the type. Because the scene graph concept implies that only one `Node` object can be in one place of the application scene, the selected item is removed from the `ComboBox` list of items. When the selection changes, the previously selected item returns to the list and the new selection is removed. To prevent this situation, use the cell factory mechanism and the solution described in the API documentation. The cell factory mechanism is particularly helpful when you need to change the initial behavior or appearance of the `ComboBox` object.

The ComboBoxSample application is designed to illustrate how to use combo boxes in a typical email interface. Example 16–3 creates a such an interface, in which two combo boxes are used to select the email recipient and the priority of the message.

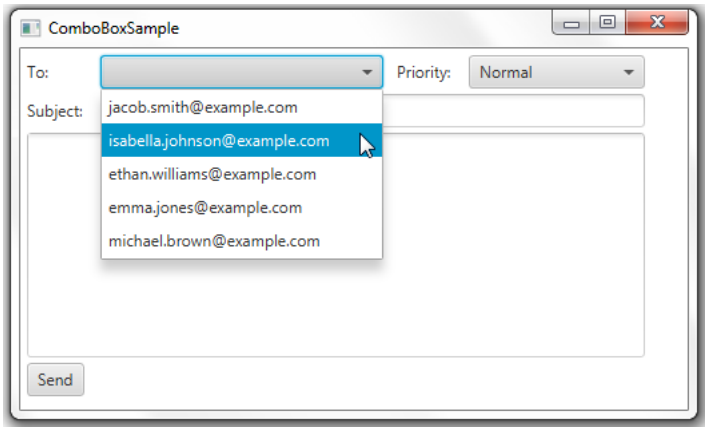
Example 16–3 Creating Combo Boxes and Adding Them to the Scene

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class ComboBoxSample extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    final Button button = new Button ("Send");
    final Label notification = new Label ();
    final TextField subject = new TextField("");
    final TextArea text = new TextArea ("");

    String address = " ";

    @Override public void start(Stage stage) {
        stage.setTitle("ComboBoxSample");
        Scene scene = new Scene(new Group(), 500, 270);

        final ComboBox emailComboBox = new ComboBox();
        emailComboBox.getItems().addAll(
            "jacob.smith@example.com",
            "isabella.johnson@example.com",
            "ethan.williams@example.com",
            "emma.jones@example.com",
            "michael.brown@example.com"
        );

        final ComboBox priorityComboBox = new ComboBox();
        priorityComboBox.getItems().addAll(
            "Highest",
            "High",
            "Normal",
            "Low",
            "Lowest"
        );

        priorityComboBox.setValue("Normal");
```

```
        GridPane grid = new GridPane();
        grid.setVgap(4);
        grid.setHgap(10);
        grid.setPadding(new Insets(5, 5, 5, 5));
        grid.add(new Label("To: "), 0, 0);
        grid.add(emailComboBox, 1, 0);
        grid.add(new Label("Priority: "), 2, 0);
        grid.add(priorityComboBox, 3, 0);
        grid.add(new Label("Subject: "), 0, 1);
        grid.add(subject, 1, 1, 3, 1);
        grid.add(text, 0, 2, 4, 1);
        grid.add(button, 0, 3);
        grid.add (notification, 1, 3, 3, 1);

        Group root = (Group)scene.getRoot();
        root.getChildren().add(grid);
        stage.setScene(scene);
        stage.show();
    }
}
```

Both combo boxes in Example 16–3 use the `getItems` and `addAll` methods to add items. When you compile and run this code, it produces the application window shown in Figure 16–4.

Figure 16–4 Email Recipient and Priority Combo Boxes



Editable Combo Boxes

Typically, email client applications enable users to both select recipients from the address book and type a new address. An editable combo box perfectly fits this task. Use the `setEditable(true)` method of the `ComboBox` class to make a combo box editable. With the `setPromptText` method, you can specify the text to appear in the combo box editing area when no selection is performed. Examine the modified code of the application in Example 16–4. The bold lines are the additions made to Example 16–3.

Example 16–4 Processing Newly Typed Values in an Editable Combo Box

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.geometry.Insets;
```

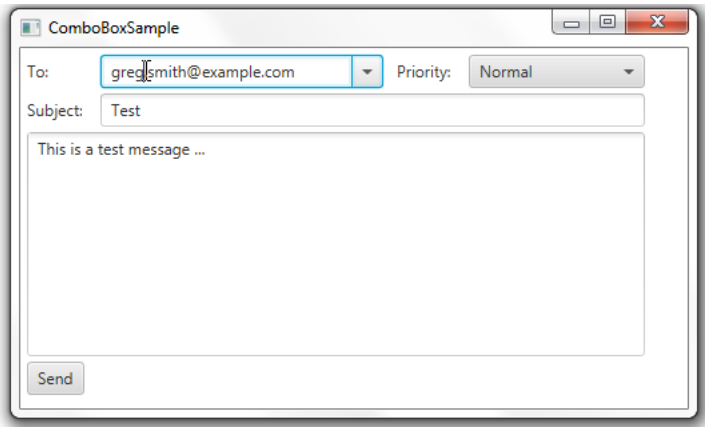
```
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class ComboBoxSample extends Application {
        public static void main(String[] args) {
        launch(args);
    }

    final Button button = new Button ("Send");
    final Label notification = new Label ();
    final TextField subject = new TextField("");
    final TextArea text = new TextArea ("");

    String address = " ";

    @Override public void start(Stage stage) {
        stage.setTitle("ComboBoxSample");
        Scene scene = new Scene(new Group(), 500, 270);

        final ComboBox emailComboBox = new ComboBox();
        emailComboBox.getItems().addAll(
            "jacob.smith@example.com",
            "isabella.johnson@example.com",
            "ethan.williams@example.com",
            "emma.jones@example.com",
            "michael.brown@example.com"
        );
        emailComboBox.setPromptText("Email address");
        emailComboBox.setEditable(true);
        emailComboBox.setOnAction((Event ev) -> {
            address =
                emailComboBox.getSelectionModel().getSelectedItem().toString();
        });

        final ComboBox priorityComboBox = new ComboBox();
        priorityComboBox.getItems().addAll(
            "Highest",
            "High",
            "Normal",
            "Low",
            "Lowest"
        );

        priorityComboBox.setValue("Normal");

        button.setOnAction((ActionEvent e) -> {
            if (emailComboBox.getValue() != null &&
                    !emailComboBox.getValue().toString().isEmpty()){
                notification.setText("Your message was successfully sent"
                        + " to " + address);
                emailComboBox.setValue(null);
                if (priorityComboBox.getValue() != null &&
                        !priorityComboBox.getValue().toString().isEmpty()){
                    priorityComboBox.setValue(null);
                }
                subject.clear();
                text.clear();
```

```
            }
            else {
                notification.setText("You have not selected a recipient!");
            }
        });

        GridPane grid = new GridPane();
        grid.setVgap(4);
        grid.setHgap(10);
        grid.setPadding(new Insets(5, 5, 5, 5));
        grid.add(new Label("To: "), 0, 0);
        grid.add(emailComboBox, 1, 0);
        grid.add(new Label("Priority: "), 2, 0);
        grid.add(priorityComboBox, 3, 0);
        grid.add(new Label("Subject: "), 0, 1);
        grid.add(subject, 1, 1, 3, 1);
        grid.add(text, 0, 2, 4, 1);
        grid.add(button, 0, 3);
        grid.add (notification, 1, 3, 3, 1);

        Group root = (Group)scene.getRoot();
        root.getChildren().add(grid);
        stage.setScene(scene);
        stage.show();
    }
}
```

Besides the ability to edit `emailComboBox`, this code fragment implements event handling for this control. The newly typed or selected value is stored in the `address` variable. When users press the Send button, the notification containing the email address is shown.

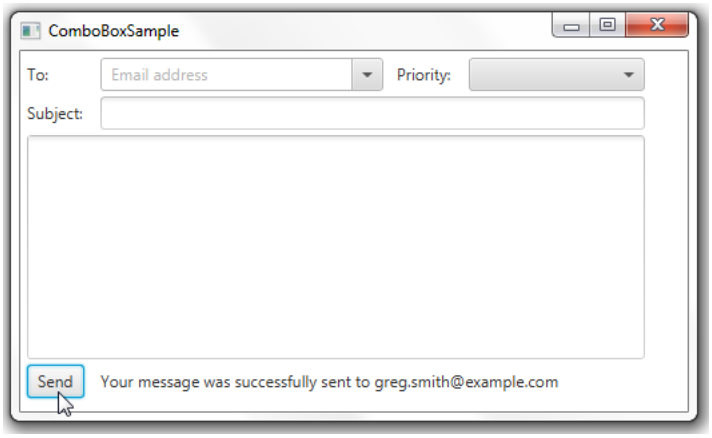
Figure 16–5 captures the moment when a user is editing the email address of Jacob Smith and changing it to greg.smith@example.com.

Figure 16–5 Editing an Email Address



When the Send button is pressed, all the controls return to their default states. The `clear` methods are called on the `TextField` and `TextArea` objects, and the `null` value is set for the combo box selected items. Figure 16–6 shows the moment after the Send button is pressed.

Figure 16–6 User Interface After the Send Button Is Pressed



Applying Cell Factories to Combo Boxes

You can use the cell factory mechanism to alter the default behavior or appearance of a combo box. Example 16–5 creates a cell factory and applies it to the priority combo box to highlight priority types with special colors.

Example 16–5 Implementing a Cell Factory for the Priority Combo Box

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.util.Callback;

public class ComboBoxSample extends Application {
        public static void main(String[] args) {
        launch(args);
    }

    final Button button = new Button ("Send");
    final Label notification = new Label ();
    final TextField subject = new TextField("");
    final TextArea text = new TextArea ("");

    String address = " ";

    @Override public void start(Stage stage) {
        stage.setTitle("ComboBoxSample");
        Scene scene = new Scene(new Group(), 500, 270);

        final ComboBox emailComboBox = new ComboBox();
        emailComboBox.getItems().addAll(
            "jacob.smith@example.com",
            "isabella.johnson@example.com",
            "ethan.williams@example.com",
            "emma.jones@example.com",
```

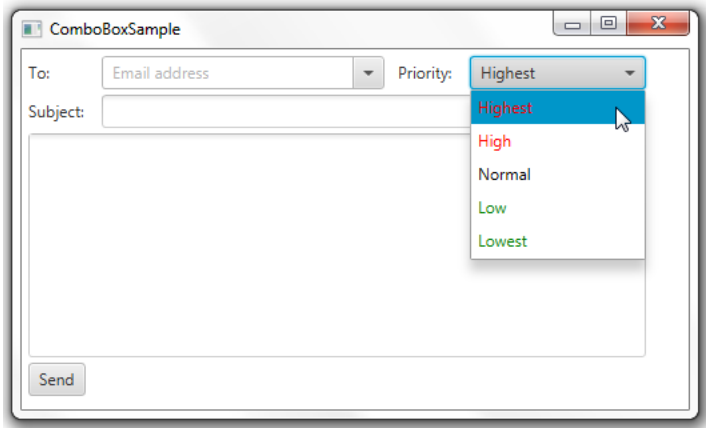
```
    "michael.brown@example.com"
);
emailComboBox.setPromptText("Email address");
emailComboBox.setEditable(true);
emailComboBox.setOnAction((Event ev) -> {
    address =
        emailComboBox.getSelectionModel().getSelectedItem().toString();
});

final ComboBox priorityComboBox = new ComboBox();
priorityComboBox.getItems().addAll(
    "Highest",
    "High",
    "Normal",
    "Low",
    "Lowest"
);

priorityComboBox.setValue("Normal");
priorityComboBox.setCellFactory(
    new Callback<ListView<String>, ListCell<String>>() {
        @Override public ListCell<String> call(ListView<String> param) {
            final ListCell<String> cell = new ListCell<String>() {
                {
                    super.setPrefWidth(100);
                }
                @Override public void updateItem(String item,
                    boolean empty) {
                        super.updateItem(item, empty);
                        if (item != null) {
                            setText(item);
                            if (item.contains("High")) {
                                setTextFill(Color.RED);
                            }
                            else if (item.contains("Low")){
                                setTextFill(Color.GREEN);
                            }
                            else {
                                setTextFill(Color.BLACK);
                            }
                        }
                        else {
                            setText(null);
                        }
                    }
        };
        return cell;
    }
});

button.setOnAction((ActionEvent e) -> {
    if (emailComboBox.getValue() != null &&
            !emailComboBox.getValue().toString().isEmpty()){
        notification.setText("Your message was successfully sent"
                + " to " + address);
        emailComboBox.setValue(null);
        if (priorityComboBox.getValue() != null &&
                !priorityComboBox.getValue().toString().isEmpty()){
            priorityComboBox.setValue(null);
        }
```

```
                subject.clear();
                text.clear();
            }
            else {
                notification.setText("You have not selected a recipient!");
            }
        })

        GridPane grid = new GridPane();
        grid.setVgap(4);
        grid.setHgap(10);
        grid.setPadding(new Insets(5, 5, 5, 5));
        grid.add(new Label("To: "), 0, 0);
        grid.add(emailComboBox, 1, 0);
        grid.add(new Label("Priority: "), 2, 0);
        grid.add(priorityComboBox, 3, 0);
        grid.add(new Label("Subject: "), 0, 1);
        grid.add(subject, 1, 1, 3, 1);
        grid.add(text, 0, 2, 4, 1);
        grid.add(button, 0, 3);
        grid.add (notification, 1, 3, 3, 1);

        Group root = (Group)scene.getRoot();
        root.getChildren().add(grid);
        stage.setScene(scene);
        stage.show();

    }
}
```

The cell factory produces `ListCell` objects. Every cell is associated with a single combo box item. The width of each combo box item is set through the `setPrefWidth` method. The `updateItem` method sets the red color for the High and Highest items, green color for the Low and Lowest items, and leaves the Normal item black.

Figure 16–7 shows the items of the priority combo box after the cell factory in Example 16–5 is applied.

Figure 16–7 Modified the Priority Combo Box



You can further enhance the appearance of the `ComboBox` control by applying CSS styles or visual effects.

Related API Documentation

- `ComboBox`
- `ComboBoxBase`
- `ListView`
- `ListCell`
- `Button`

17

Separator

This chapter explains how to use separator to organize UI components of you JavaFX applications.

The `Separator` class that is available in the JavaFX API represents a horizontal or vertical separator line. It serves to divide elements of the application user interface and does not produce any action. However, you can style it, apply visual effects to it, or even animate it. By default, the separator is horizontal. You can change its orientation by using the `setOrientation` method.

Creating a Separator

The code fragment in Example 17–1 creates one horizontal separator and one vertical separator.

Example 17–1 Vertical and Horizontal Separators

```
//Horizontal separator
Separator separator1 = new Separator();
//Vertical separator
Separator separator2 = new Separator();
separator2.setOrientation(Orientation.VERTICAL);
```

The `Separator` class is an extension of the `Node` class. Therefore, the separator inherits all the instance variables of the `Node` class.

Typically, separators are used to divide groups of the UI controls. Study the code fragment shown in Example 17–2. It separates the spring month checkboxes from the summer month checkboxes.

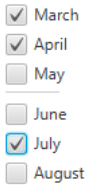
Example 17–2 Using a Separator Between Checkbox Categories

```
final String[] names = new String[]{"March", "April", "May",
    "June", "July", "August"};
final CheckBox[] cbs = new CheckBox[names.length];
final Separator separator = new Separator();
final VBox vbox = new VBox();

for (int i = 0; i < names.length; i++) {
    cbs[i] = new CheckBox(names[i]);
}

separator.setMaxWidth(40);
separator.setHalignment(HPos.LEFT);
vbox.getChildren().addAll(cbs);
vbox.setSpacing(5);
```

```
vbox.getChildren().add(3, separator);
```

When this code fragment is added to an application, it produces the controls shown in Figure 17–1.

Figure 17–1 Checkboxes and a Separator



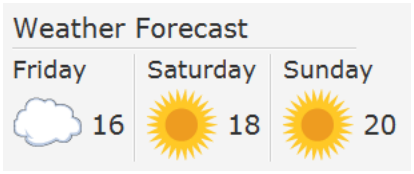
A separator occupies the full horizontal or vertical space allocated to it. The `setMaxWidth` method is applied to define a particular width. The `setValignment` method specifies the vertical position of the separator within the allocated layout space. Similarly, you can set the horizontal position of the separator line by applying the `setHalignment` method.

In Example 17–2, the separator is added to the vertical box by using a dedicated method `add(index, node)`. You can use this approach in your application to include separators after the UI is created or when the UI is dynamically changed.

Adding Separators to the UI of Your Application

As previously mentioned, separators can be used to divide groups of UI controls. You can also use them to structure a user interface. Consider the task of rendering the weather forecast data shown in Figure 17–2.

Figure 17–2 Structuring Weather Forecast Data with Separators



For the application shown in Figure 17–2, separators are used to divide `Label` and `ImageView` objects. Study the source code of this application shown in Example 17–3.

Example 17–3 Using Separators in a Weather Forecast Application

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Orientation;
import javafx.geometry.VPos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.GridPane;
```

```
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class SeparatorSample extends Application {

    Label caption = new Label("Weather Forecast");
    Label friday = new Label("Friday");
    Label saturday = new Label("Saturday");
    Label sunday = new Label("Sunday");

    @Override
    public void start(Stage stage) {
        Group root = new Group();
        Scene scene = new Scene(root, 350, 150);
        stage.setScene(scene);
        stage.setTitle("Separator Sample");

        GridPane grid = new GridPane();
        grid.setPadding(new Insets(10, 10, 10, 10));
        grid.setVgap(2);
        grid.setHgap(5);

        scene.setRoot(grid);

        Image cloudImage = new Image(getClass().getResourceAsStream("cloud.jpg"));
        Image sunImage = new Image(getClass().getResourceAsStream("sun.jpg"));

        caption.setFont(Font.font("Verdana", 20));
        GridPane.setConstraints(caption, 0, 0);
        GridPane.setColumnSpan(caption, 8);
        grid.getChildren().add(caption);

        final Separator sepHor = new Separator();
        sepHor.setValignment(VPos.CENTER);
        GridPane.setConstraints(sepHor, 0, 1);
        GridPane.setColumnSpan(sepHor, 7);
        grid.getChildren().add(sepHor);

        friday.setFont(Font.font("Verdana", 18));
        GridPane.setConstraints(friday, 0, 2);
        GridPane.setColumnSpan(friday, 2);
        grid.getChildren().add(friday);

        final Separator sepVert1 = new Separator();
        sepVert1.setOrientation(Orientation.VERTICAL);
        sepVert1.setValignment(VPos.CENTER);
        sepVert1.setPrefHeight(80);
        GridPane.setConstraints(sepVert1, 2, 2);
        GridPane.setRowSpan(sepVert1, 2);
        grid.getChildren().add(sepVert1);

        saturday.setFont(Font.font("Verdana", 18));
        GridPane.setConstraints(saturday, 3, 2);
        GridPane.setColumnSpan(saturday, 2);
        grid.getChildren().add(saturday);

        final Separator sepVert2 = new Separator();
        sepVert2.setOrientation(Orientation.VERTICAL);
        sepVert2.setValignment(VPos.CENTER);
        sepVert2.setPrefHeight(80);
```

```
            GridPane.setConstraints(sepVert2, 5, 2);
            GridPane.setRowSpan(sepVert2, 2);
            grid.getChildren().add(sepVert2);

            sunday.setFont(Font.font("Verdana", 18));
            GridPane.setConstraints(sunday, 6, 2);
            GridPane.setColumnSpan(sunday, 2);
            grid.getChildren().add(sunday);

            final ImageView cloud = new ImageView(cloudImage);
            GridPane.setConstraints(cloud, 0, 3);
            grid.getChildren().add(cloud);

            final Label t1 = new Label("16");
            t1.setFont(Font.font("Verdana", 20));
            GridPane.setConstraints(t1, 1, 3);
            grid.getChildren().add(t1);

            final ImageView sun1 = new ImageView(sunImage);
            GridPane.setConstraints(sun1, 3, 3);
            grid.getChildren().add(sun1);

            final Label t2 = new Label("18");
            t2.setFont(Font.font("Verdana", 20));
            GridPane.setConstraints(t2, 4, 3);
            grid.getChildren().add(t2);

            final ImageView sun2 = new ImageView(sunImage);
            GridPane.setConstraints(sun2, 6, 3);
            grid.getChildren().add(sun2);

            final Label t3 = new Label("20");
            t3.setFont(Font.font("Verdana", 20));
            GridPane.setConstraints(t3, 7, 3);
            grid.getChildren().add(t3);

            stage.show();
        }
        public static void main(String[] args) {
            launch(args);
        }
}
```

This application uses both horizontal and vertical separators and makes the separators span rows and columns in the `GridPane` container. In your application, you can also set the preferred length for a separator (width for a horizontal separator and height for a vertical separator), so that it can change dynamically when the user interface resizes. You can also alter the visual appearance of a separator by applying the CSS classes available for `Separator` objects.

Styling Separators

To apply the same style to all the separators in Example 17–3, you create CSS file (for example, controlStyle.css) and save this file in the same package as the main class of your application. Example 17–4 demonstrates the CSS classes that you can add to the controlStyle file.

Example 17–4 Using CSS Classes to Style Separators

```
/*controlStyle.css */
.separator .line {
    -fx-border-color: #e79423;
    -fx-border-width: 2;
}
```

You can enable the separator style in the application through the `getStylesheets` method of the `Scene` class, as shown in Example 17–5.

Example 17–5 Enabling Style Sheets in a JavaFX Application

```
scene.getStylesheets().add("separatorsample/controlStyle.css");
```

Figure 17–3 shows how the separators in the weather forecast look when the modified application is compiled and run.

Figure 17–3 Styled Separators

Related API Documentation

- `Separator`
- JavaFX CSS Specification

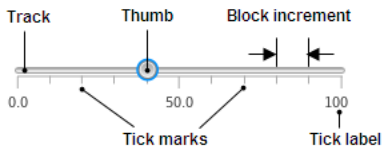
18

Slider

In this chapter, you learn how to use sliders in your JavaFX applications to display and interact with a range of numeric values.

The `Slider` control consists of a track and a draggable thumb. It can also include tick marks and tick labels that indicate numeric values of the range. Figure 18–1 shows a typical slider and indicates its main elements.

Figure 18–1 Elements of a Slider



Creating a Slider

Take a moment to review the code fragment in Example 18–1 that produces a slider shown in Figure 18–1.

Example 18–1 Creating a Slider

```
Slider slider = new Slider();
slider.setMin(0);
slider.setMax(100);
slider.setValue(40);
slider.setShowTickLabels(true);
slider.setShowTickMarks(true);
slider.setMajorTickUnit(50);
slider.setMinorTickCount(5);
slider.setBlockIncrement(10);
```

The `setMin` and `setMax` methods define, respectively, the minimum and the maximum numeric values represented by the slider. The `setValue` method specifies the current value of the slider, which is always less than the maximum value and more than the minimum value. Use this method to define the position of the thumb when the application starts.

Two Boolean methods, `setShowTickMarks` and `setShowTickLabels`, define the visual appearance of the slider. In Example 18–1, the marks and labels are enabled. Additionally, the unit distance between major tick marks is set to 50, and the number

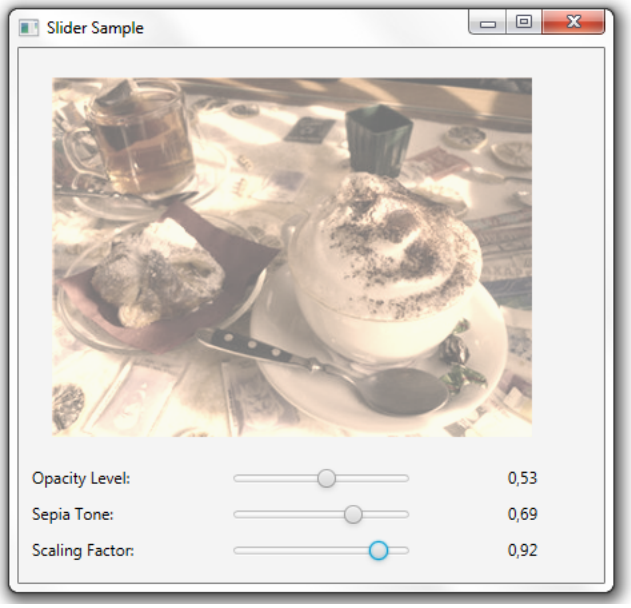
of minor ticks between any two major ticks is specified as 5. You can assign the `setSnapToTicks` method to `true` to keep the slider's value aligned with the tick marks.

The `setBlockIncrement` method defines the distance that the thumb moves when a user clicks on the track. In Example 18–1, this value is 10, which means that each time a user clicks on the track, the thumb moves 10 units toward the click location.

Using Sliders in Graphical Applications

Now examine Figure 18–2. This application uses three sliders to edit rendering characteristics of a picture. Each slider adjusts a particular visual characteristic: opacity level, sepia tone value, or scaling factor.

Figure 18–2 Three Sliders



Example 18–2 shows the source code of this application.

Example 18–2 Slider Sample

```
import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.effect.SepiaTone;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class SliderSample extends Application {

    final Slider opacityLevel = new Slider(0, 1, 1);
    final Slider sepiaTone = new Slider(0, 1, 1);
```

```
final Slider scaling = new Slider (0.5, 1, 1);
final Image image  = new Image(getClass().getResourceAsStream(
    "cappuccino.jpg")
);

final Label opacityCaption = new Label("Opacity Level:");
final Label sepiaCaption = new Label("Sepia Tone:");
final Label scalingCaption = new Label("Scaling Factor:");

final Label opacityValue = new Label(
    Double.toString(opacityLevel.getValue()));
final Label sepiaValue = new Label(
    Double.toString(sepiaTone.getValue()));
final Label scalingValue = new Label(
    Double.toString(scaling.getValue()));

final static Color textColor = Color.BLACK;
final static SepiaTone sepiaEffect = new SepiaTone();

@Override
public void start(Stage stage) {
    Group root = new Group();
    Scene scene = new Scene(root, 600, 400);
    stage.setScene(scene);
    stage.setTitle("Slider Sample");

    GridPane grid = new GridPane();
    grid.setPadding(new Insets(10, 10, 10, 10));
    grid.setVgap(10);
    grid.setHgap(70);

    final ImageView cappuccino = new ImageView (image);
    cappuccino.setEffect(sepiaEffect);
    GridPane.setConstraints(cappuccino, 0, 0);
    GridPane.setColumnSpan(cappuccino, 3);
    grid.getChildren().add(cappuccino);
    scene.setRoot(grid);

    opacityCaption.setTextFill(textColor);
    GridPane.setConstraints(opacityCaption, 0, 1);
    grid.getChildren().add(opacityCaption);


    opacityLevel.valueProperty().addListener((
        ObservableValue<? extends Number> ov,
        Number old_val, Number new_val) -> {
            cappuccino.setOpacity(new_val.doubleValue());
            opacityValue.setText(String.format("%.2f", new_val));
    });

    GridPane.setConstraints(opacityLevel, 1, 1);
    grid.getChildren().add(opacityLevel);

    opacityValue.setTextFill(textColor);
    GridPane.setConstraints(opacityValue, 2, 1);
    grid.getChildren().add(opacityValue);

    sepiaCaption.setTextFill(textColor);
    GridPane.setConstraints(sepiaCaption, 0, 2);
    grid.getChildren().add(sepiaCaption);
```

```
        sepiaTone.valueProperty().addListener((
            ObservableValue<? extends Number> ov, Number old_val,
            Number new_val) -> {
                sepiaEffect.setLevel(new_val.doubleValue());
                sepiaValue.setText(String.format("%.2f", new_val));
        });
        GridPane.setConstraints(sepiaTone, 1, 2);
        grid.getChildren().add(sepiaTone);

        sepiaValue.setTextFill(textColor);
        GridPane.setConstraints(sepiaValue, 2, 2);
        grid.getChildren().add(sepiaValue);

        scalingCaption.setTextFill(textColor);
        GridPane.setConstraints(scalingCaption, 0, 3);
        grid.getChildren().add(scalingCaption);

        scaling.valueProperty().addListener((
            ObservableValue<? extends Number> ov, Number old_val,
            Number new_val) -> {
                cappuccino.setScaleX(new_val.doubleValue());
                cappuccino.setScaleY(new_val.doubleValue());
                scalingValue.setText(String.format("%.2f", new_val));
        });
        GridPane.setConstraints(scaling, 1, 3);
        grid.getChildren().add(scaling);

        scalingValue.setTextFill(textColor);
        GridPane.setConstraints(scalingValue, 2, 3);
        grid.getChildren().add(scalingValue);

        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

The opacity property of the `ImageView` object changes in accordance with the value of the first slider, named opacityLevel. The level of the `SepiaTone` effect changes when the value of the sepiaTone slider changes. The third slider defines the scaling factor for the picture by passing to the `setScaleX` and `setScaleY` methods the current value of the slider.

The code fragment in Example 18–3 demonstrates the methods that convert the double value returned by the `getValue` method of the `Slider` class into `String`. It also applies formatting to render the slider's value as a float number with two digits after the point.

Example 18–3 Formatting the Rendered Slider's Value

```
scalingValue.setText(String.format("%.2f", new_val));
```

The next step to enhance the look and feel of a slider is to apply visual effects or CSS styles to it.

Related API Documentation

- `Slider`

- `SepiaTone`

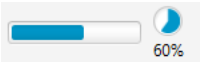
19

Progress Bar and Progress Indicator

In this chapter, you learn about the progress indicator and progress bar, the UI controls that visualize progress of any operations in your JavaFX applications.

The `ProgressIndicator` class and its direct subclass `ProgressBar` provide the capabilities to indicate that a particular task is processing and to detect how much of this work has been already done. While the `ProgressBar` class visualizes the progress as a completion bar, the `ProgressIndicator` class visualizes the progress in the form of a dynamically changing pie chart, as shown in Figure 19–1.

Figure 19–1 Progress Bar and Progress Indicator



Creating Progress Controls

Use the code fragment in Example 19–1 to insert the progress controls in your JavaFX application.

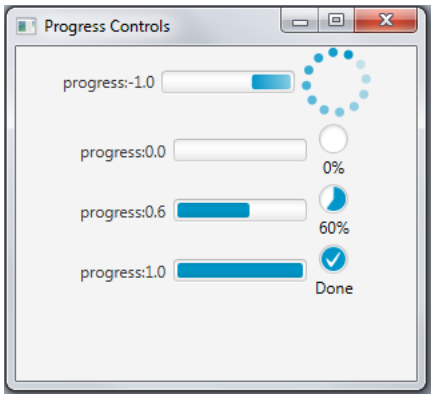
Example 19–1 Implementing the Progress Bar and Progress Indicator

```
ProgressBar pb = new ProgressBar(0.6);
ProgressIndicator pi = new ProgressIndicator(0.6);
```

You can also create the progress controls without parameters by using an empty constructor. In that case, you can assign the value by using the `setProgress` method.

Sometimes an application cannot determine the full completion time of a task. In that case, progress controls remain in indeterminate mode until the length of the task is determined. Figure 19–2 shows different states of the progress controls depending on their progress variable value.

Figure 19–2 Various States of Progress Controls



Example 19–2 shows the source code of the application shown in Figure 19–2.

Example 19–2 Enabling Different States of Progress Controls

```
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ProgressBar;
import javafx.scene.control.ProgressIndicator;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class ProgressSample extends Application {

final Float[] values = new Float[] {-1.0f, 0f, 0.6f, 1.0f};
final Label [] labels = new Label[values.length];
final ProgressBar[] pbs = new ProgressBar[values.length];
final ProgressIndicator[] pins = new ProgressIndicator[values.length];
final HBox hbs [] = new HBox [values.length];

    @Override
    public void start(Stage stage) {
        Group root = new Group();
        Scene scene = new Scene(root, 300, 250);
        stage.setScene(scene);
        stage.setTitle("Progress Controls");

        for (int i = 0; i < values.length; i++) {
            final Label label = labels[i] = new Label();
            label.setText("progress:" + values[i]);

            final ProgressBar pb = pbs[i] = new ProgressBar();
            pb.setProgress(values[i]);

            final ProgressIndicator pin = pins[i] = new ProgressIndicator();
            pin.setProgress(values[i]);
            final HBox hb = hbs[i] = new HBox();
            hb.setSpacing(5);
            hb.setAlignment(Pos.CENTER);
            hb.getChildren().addAll(label, pb, pin);
```

```
        }

        final VBox vb = new VBox();
        vb.setSpacing(5);
        vb.getChildren().addAll(hbs);
        scene.setRoot(vb);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

A positive value of the progress variable between 0 and 1 indicates the percentage of progress. For example, 0.4 corresponds to 40%. A negative value for this variable indicates that the progress is in the indeterminate mode. Use the `isIndeterminate` method to check whether the progress control is in the indeterminate mode.

Indicating Progress in Your User Interface

Example 19–2 was initially simplified to render all the possible states of the progress controls. In real-world applications, the progress value can be obtained through the value of other UI elements.

Study the code in Example 19–3 to learn how set values for the progress bar and progress indicator based on the slider position.

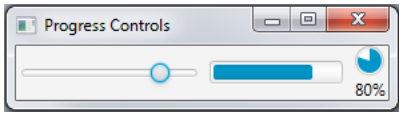
Example 19–3 Receiving the Progress Value from a Slider

```
import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.ProgressBar;
import javafx.scene.control.ProgressIndicator;
import javafx.scene.control.Slider;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class ProgressSample extends Application {

    @Override
    public void start(Stage stage) {
        Group root = new Group();
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.setTitle("Progress Controls");

        final Slider slider = new Slider();
        slider.setMin(0);
        slider.setMax(50);

        final ProgressBar pb = new ProgressBar(0);
        final ProgressIndicator pi = new ProgressIndicator(0);

        slider.valueProperty().addListener(
            (ObservableValue<? extends Number> ov, Number old_val,
```

```
            Number new_val) -> {
                pb.setProgress(new_val.doubleValue()/50);
                pi.setProgress(new_val.doubleValue()/50);
        });

        final HBox hb = new HBox();
        hb.setSpacing(5);
        hb.setAlignment(Pos.CENTER);
        hb.getChildren().addAll(slider, pb, pi);
        scene.setRoot(hb);
        stage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

When you compile and run this application, it produces the window shown in Figure 19–3.

Figure 19–3 Indicating the Progress Set by a Slider



An `ChangeListener` object determines if the slider's value is changed and computes the progress for the progress bar and progress indicator so that the values of the progress controls are in the range of 0.0 to 1.0.

Related API Documentation.

- `ProgressBar`
- `ProgressIndicator`

20

Hyperlink

This chapter tells about the `Hyperlink` control used to format text as a hyperlink.

The `Hyperlink` class represents another type of `Labeled` control. Figure 20–1 demonstrates three states of the default hyperlink implementation.

Figure 20–1 Three States of a Hyperlink Control

Creating a Hyperlink

The code fragment that produces a hyperlink is shown in Example 20–1.

Example 20–1 Typical Hyperlink

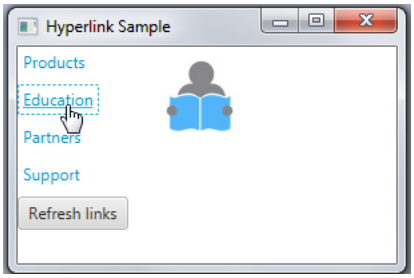
```
Hyperlink link = new Hyperlink();
link.setText("http://example.com");
link.setOnAction((ActionEvent e) -> {
    System.out.println("This link is clicked");
});
```

The `setText` instance method defines the text caption for the hyperlink. Because the `Hyperlink` class is an extension of the `Labeled` class, you can set a specific font and text fill for the hyperlink caption. The `setOnAction` method sets the specific action, which is called whenever the hyperlink is clicked, similar to how this method works for the `Button` control. In Example 20–1, this action is limited to printing the string. However, in your application, you might want to implement more common tasks.

Linking the Local Content

The application in Figure 20–2 renders images from the local directory.

Figure 20–2 Viewing Images



Review the source code of this application shown in Example 20–2.

Example 20–2 Using Hyperlinks to View Images

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class HyperlinkSample extends Application {

    final static String[] imageFiles = new String[]{
        "product.png",
        "education.png",
        "partners.png",
        "support.png"
    };
    final static String[] captions = new String[]{
        "Products",
        "Education",
        "Partners",
        "Support"
    };
    final ImageView selectedImage = new ImageView();
    final ScrollPane list = new ScrollPane();
    final Hyperlink[] hpls = new Hyperlink[captions.length];
    final Image[] images = new Image[imageFiles.length];

    public static void main(String[] args) {
        Application.launch(args);
    }

    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Hyperlink Sample");
        stage.setWidth(300);
        stage.setHeight(200);

        selectedImage.setLayoutX(100);
        selectedImage.setLayoutY(10);

        for (int i = 0; i < captions.length; i++) {
            final Hyperlink hpl = hpls[i] = new Hyperlink(captions[i]);
```

```
            final Image image = images[i] = new Image(
                getClass().getResourceAsStream(imageFiles[i])
            );
            hpl.setOnAction((ActionEvent e) -> {
                selectedImage.setImage(image);
            });
        }

        final Button button = new Button("Refresh links");
        button.setOnAction((ActionEvent e) -> {
            for (int i = 0; i < captions.length; i++) {
                hpls[i].setVisited(false);
                selectedImage.setImage(null);
            }
        });

        VBox vbox = new VBox();
        vbox.getChildren().addAll(hpls);
        vbox.getChildren().add(button);
        vbox.setSpacing(5);

        ((Group) scene.getRoot()).getChildren().addAll(vbox, selectedImage);
        stage.setScene(scene);
        stage.show();
    }
}
```

This application creates four `Hyperlink` objects within a `for` loop. The `setOnAction` method called on each hyperlink defines the behavior when a user clicks a particular hyperlink. In that case, the corresponding image from the `images` array is set for the `selectedImage` variable.

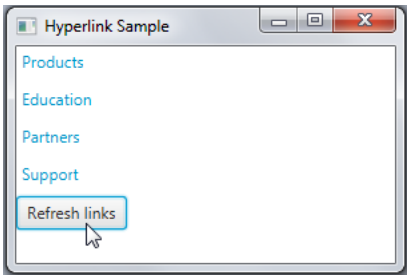
When a user clicks a hyperlink, it becomes visited. You can use the `setVisited` method of the `Hyperlink` class to refresh the links. The code fragment in Example 20–3 accomplishes this task.

Example 20–3 Refreshing the HyperlInks

```
final Button button = new Button("Refresh links");
button.setOnAction((ActionEvent e) -> {
    for (int i = 0; i < captions.length; i++) {
        hpls[i].setVisited(false);
        selectedImage.setImage(null);
    }
});
```

When clicked, the Refresh Links button brings all the hyperlinks to the unvisited state, as show in Figure 20–3.

Figure 20–3 Unvisited Hyperlinks



Because the `Hyperlink` class is an extension of the `Labeled` class, you can specify not only a text caption but also an image. The application provided in the next section uses both a text caption and an image to create hyperlinks and to load remote HTML pages.

Linking the Remote Content

You can render HTML content in your JavaFX application by embedding the `WebView` browser in the application scene. The `WebView` component provides basic web page browsing functionality. It renders web pages and supports user interaction such as navigating links and executing JavaScript commands.

Study the source code of the application in Example 20–4. It creates four hyperlinks with text captions and images. When a hyperlink is clicked, the corresponding value is passed as a URL to the embedded browser.

Example 20–4 Loading Remote Web Pages

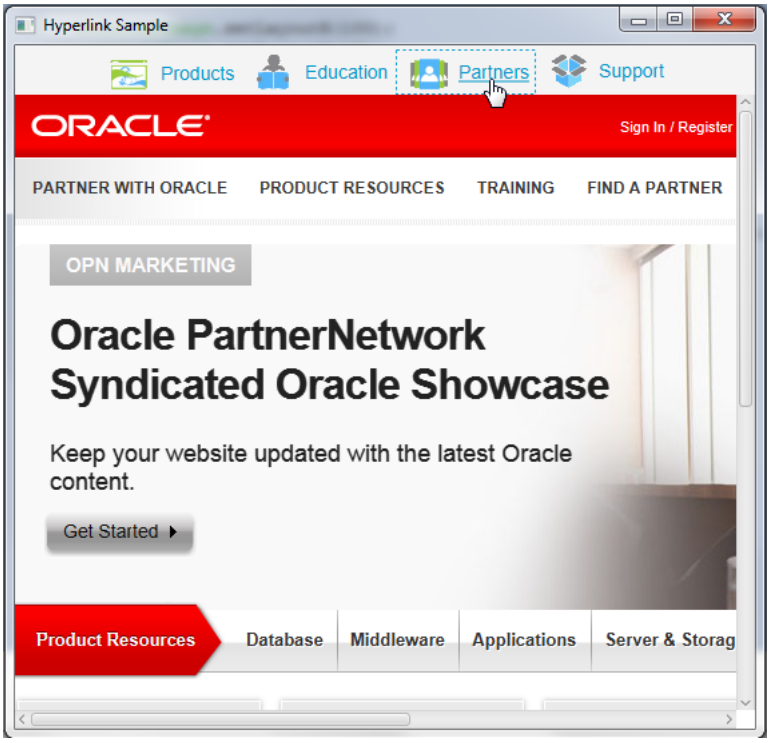
```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Pos;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
import javafx.stage.Stage;

public class HyperlinkWebViewSample extends Application {

    final static String[] imageFiles = new String[]{
        "product.png",
        "education.png",
        "partners.png",
        "support.png"
    };
    final static String[] captions = new String[]{
        "Products",
        "Education",
        "Partners",
        "Support"
    };
```

```
    final static String[] urls = new String[]{
        "http://www.oracle.com/us/products/index.html",
        "http://education.oracle.com/",
        "http://www.oracle.com/partners/index.html",
        "http://www.oracle.com/us/support/index.html"
    };

    final ImageView selectedImage = new ImageView();
    final Hyperlink[] hpls = new Hyperlink[captions.length];
    final Image[] images = new Image[imageFiles.length];

    public static void main(String[] args){
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        VBox vbox = new VBox();
        Scene scene = new Scene(vbox);
        stage.setTitle("Hyperlink Sample");
        stage.setWidth(570);
        stage.setHeight(550);

        selectedImage.setLayoutX(100);
        selectedImage.setLayoutY(10);

        final WebView browser = new WebView();
        final WebEngine webEngine = browser.getEngine();

        for (int i = 0; i < captions.length; i++) {
            final Hyperlink hpl = hpls[i] = new Hyperlink(captions[i]);
            final Image image = images[i] =
                new Image(getClass().getResourceAsStream(imageFiles[i]));
            hpl.setGraphic(new ImageView (image));
            hpl.setFont(Font.font("Arial", 14));
            final String url = urls[i];

            hpl.setOnAction((ActionEvent e) -> {
                webEngine.load(url);
            });
        }

        HBox hbox = new HBox();
        hbox.setAlignment(Pos.BASELINE_CENTER);
        hbox.getChildren().addAll(hpls);
        vbox.getChildren().addAll(hbox, browser);
        VBox.setVgrow(browser, Priority.ALWAYS);

        stage.setScene(scene);
        stage.show();
    }
}
```

The hyperlinks are created within a `for` loop similar to the one in Example 20–2. The action set for a hyperlink passes the corresponding URL from the `urls` array to the `WebEngine` object of the embedded browser.

When you compile and run this application, it produces the window shown in Figure 20–4.

Figure 20–4 Loading Pages from the Oracle Corporate Site



Related API Documentation

- `Hyperlink`
- `Labeled`
- `WebView`
- `WebEngine`

21

HTML Editor

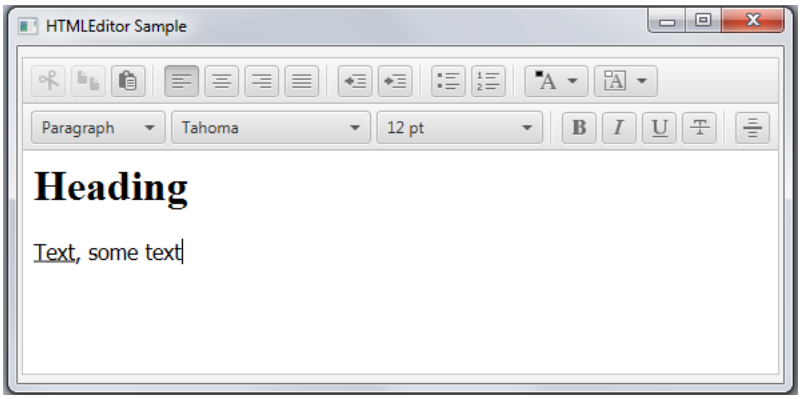
In this chapter, you learn how to edit text in your JavaFX applications by using the embedded HTML editor.

The `HTMLEditor` control is a full functional rich text editor. Its implementation is based on the document editing feature of HTML5 and includes the following editing functions:

- Text formatting including bold, italic, underline, and strike though styles
- Paragraph settings such as format, font family, and font size
- Foreground and background colors
- Text indent
- Bulleted and numbered lists
- Text alignment
- Adding a horizontal rule
- Copying and pasting text fragments

Figure 21–1 shows a rich text editor added to a JavaFX application.

Figure 21–1 HTML Editor



The `HTMLEditor` class presents the editing content in the form of an HTML string. For example, the content typed in the editor in Figure 21–1 is presented by the following string: "`<html><head></head><body contenteditable="true"><h1>Heading</h1><div><u>Text</u>, some text</div></body></html>`."

Because the `HTMLEditor` class is an extension of the `Node` class, you can apply visual effects or transformations to its instances.

Adding an HTML Editor

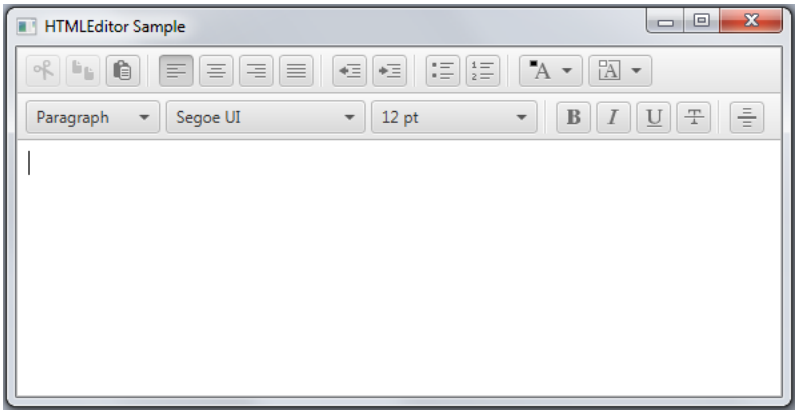
Like any other UI control, the `HTMLEditor` component must be added the scene so that it can appear in your application. You can add it directly to the scene as shown in Example 21–1 or through a layout container as done in other examples.

Example 21–1 Adding an HTML Editor to a JavaFX Application

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.web.HTMLEditor;
import javafx.stage.Stage;

public class HTMLEditorSample extends Application {

    @Override
    public void start(Stage stage) {
        stage.setTitle("HTMLEditor Sample");
        stage.setWidth(650650);
        stage.setHeight(300);
        final HTMLEditor htmlEditor = new HTMLEditor();
        htmlEditor.setPrefHeight(245);
        Scene scene = new Scene(htmlEditor);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Compiling and running this code fragment produces the window shown in Figure 21–2.

Figure 21–2 Initial View of the HTMLEditor Component



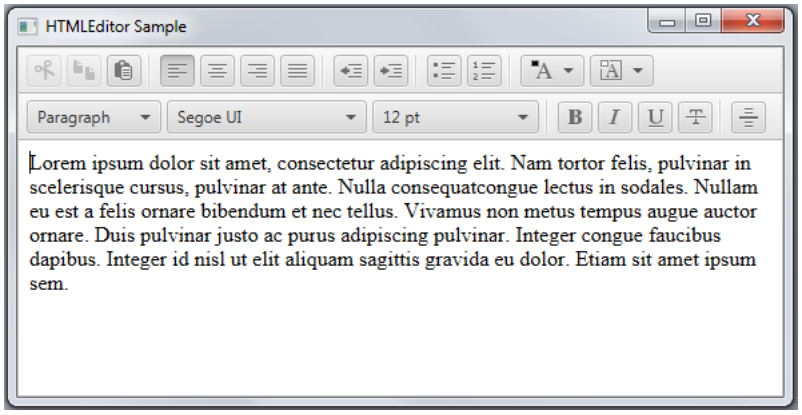
The `HTMLEditor` class provides a method to define the content that appears in the editing area when the application starts. Use the `setHtmlText` method, as shown in Example 21–2 to set the initial text for the editor.

Example 21–2 Setting the Text Content

```
private final String INITIAL_TEXT = "<html><body>Lorem ipsum dolor sit "
    + "amet, consectetur adipiscing elit. Nam tortor felis, pulvinar "
    + "in scelerisque cursus, pulvinar at ante. Nulla consequat"
    + "congue lectus in sodales. Nullam eu est a felis ornare "
    + "bibendum et nec tellus. Vivamus non metus tempus augue auctor "
    + "ornare. Duis pulvinar justo ac purus adipiscing pulvinar. "
    + "Integer congue faucibus dapibus. Integer id nisl ut elit "
    + "aliquam sagittis gravida eu dolor. Etiam sit amet ipsum "
    + "sem.</body></html>";

htmlEditor.setHtmlText(INITIAL_TEXT);
```

Figure 21–3 demonstrates the text editor with the text set by using the `setHTMLText` method.

Figure 21–3 HTMLEditor with the Predefined Text Content



You can use the HTML tags in this string to apply specific formatting for the initially rendered content.

Using an HTML Editor to Build the User Interface

You can use the `HTMLEditor` control to implement typical user interfaces (UIs) in your JavaFX applications. For example, you can implement instant messenger services, email clients, or even content management systems.

Example 21–3 presents the user interface of a message composing window that you can find in many email client applications.

Example 21–3 HTMLEditor Added to the Email Client UI

```
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.scene.web.HTMLEditor;
import javafx.stage.Stage;
```

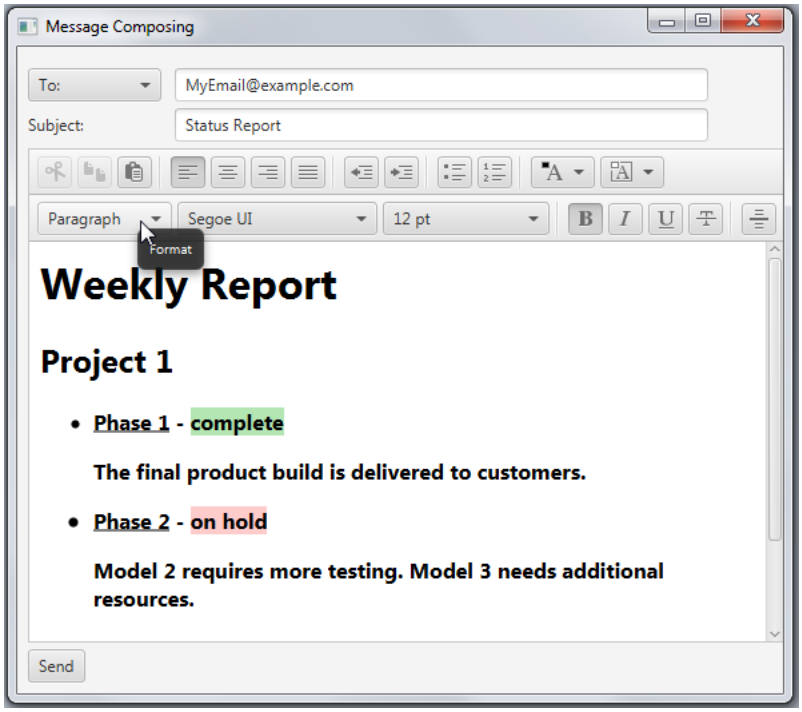
```
public class HTMLEditorSample extends Application {

    @Override
    public void start(Stage stage) {
        stage.setTitle("Message Composing");
        stage.setWidth(650);
        stage.setHeight(500);
        Scene scene = new Scene(new Group());

        final VBox root = new VBox();
        root.setPadding(new Insets(8, 8, 8, 8));
        root.setSpacing(5);
        root.setAlignment(Pos.BOTTOM_LEFT);

        final GridPane grid = new GridPane();
        grid.setVgap(5);
        grid.setHgap(10);

        final ChoiceBox sendTo =
            new ChoiceBox(FXCollections.observableArrayList(
                "To:", "Cc:", "Bcc:")
        );

        sendTo.setPrefWidth(100);
        GridPane.setConstraints(sendTo, 0, 0);
        grid.getChildren().add(sendTo);

        final TextField tbTo = new TextField();
        tbTo.setPrefWidth(400);
        GridPane.setConstraints(tbTo, 1, 0);
        grid.getChildren().add(tbTo);

        final Label subjectLabel = new Label("Subject:");
        GridPane.setConstraints(subjectLabel, 0, 1);
        grid.getChildren().add(subjectLabel);

        final TextField tbSubject = new TextField();
        tbTo.setPrefWidth(400);
        GridPane.setConstraints(tbSubject, 1, 1);
        grid.getChildren().add(tbSubject);

        root.getChildren().add(grid);

        final HTMLEditor htmlEditor = new HTMLEditor();
        htmlEditor.setPrefHeight(370);

        root.getChildren().addAll(htmlEditor, new Button("Send"));

        final Label htmlLabel = new Label();
        htmlLabel.setWrapText(true);

        scene.setRoot(root);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
```

```
}
```

The user interface includes a choice box to select a type of recipient, two text fields to enter the email address and the subject of the message, a label to indicate the subject field, the editor, and the Send button.

The UI controls are arranged on the application scene by using the `Grid` and `VBox` layout containers. When you compile and run this application, the window shown in Figure 21–4 shows the output of this application when a user is composing a weekly report.

Figure 21–4 Email Client User Interface



You can set the specific width and height values for the `HTMLEditor` object by calling the `setPrefWidth` or `setPrefHeight` methods, or you can leave its width and height unspecified. Example 21–3 specifies the height of the component. Its width is defined by the `VBox` layout container. When the text content exceeds the height of the editing area, the vertical scroll bar appears.

Obtaining HTML Content

With the JavaFX `HTMLEditor` control, you can edit text and set the initial content. In addition, you can obtain the entered and edited text in HTML format. The application shown in Example 21–4 implements this task.

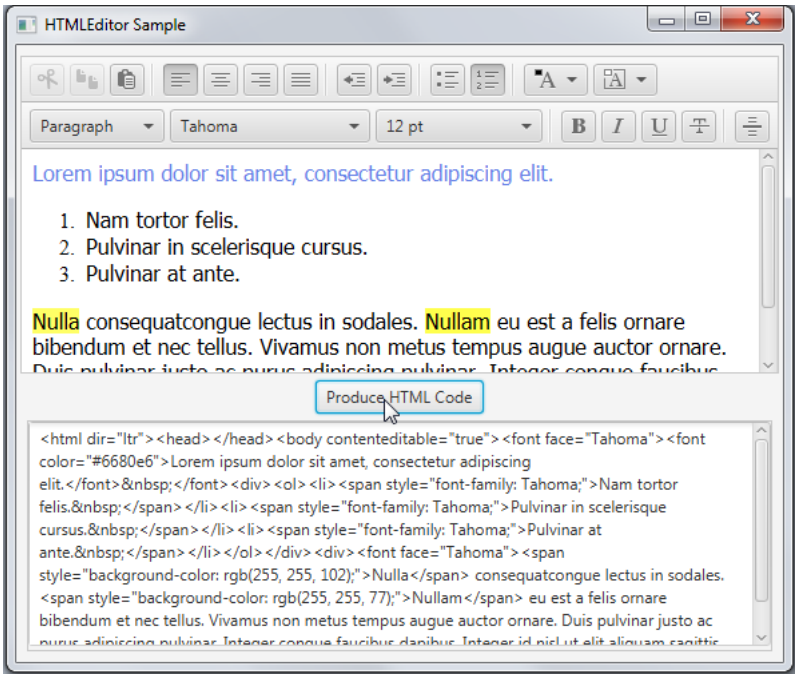
Example 21–4 Retrieving HTML Code

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Scene;
```

```
import javafx.scene.control.*;
import javafx.scene.layout.VBox;
import javafx.scene.web.HTMLEditor;
import javafx.stage.Stage;

public class HTMLEditorSample extends Application {
    private final String INITIAL_TEXT = "Lorem ipsum dolor sit "
            + "amet, consectetur adipiscing elit. Nam tortor felis, pulvinar "
            + "in scelerisque cursus, pulvinar at ante. Nulla consequat"
            + "congue lectus in sodales. Nullam eu est a felis ornare "
            + "bibendum et nec tellus. Vivamus non metus tempus augue auctor "
            + "ornare. Duis pulvinar justo ac purus adipiscing pulvinar. "
            + "Integer congue faucibus dapibus. Integer id nisl ut elit "
            + "aliquam sagittis gravida eu dolor. Etiam sit amet ipsum "
            + "sem.";

    @Override
    public void start(Stage stage) {
        stage.setTitle("HTMLEditor Sample");
        stage.setWidth(650);
        stage.setHeight(500);
        Scene scene = new Scene(new Group());

        VBox root = new VBox();
        root.setPadding(new Insets(8, 8, 8, 8));
        root.setSpacing(5);
        root.setAlignment(Pos.BOTTOM_LEFT);

        final HTMLEditor htmlEditor = new HTMLEditor();
        htmlEditor.setPrefHeight(245);
        htmlEditor.setHtmlText(INITIAL_TEXT);

        final TextArea htmlCode = new TextArea();
        htmlCode.setWrapText(true);

        ScrollPane scrollPane = new ScrollPane();
        scrollPane.getStyleClass().add("noborder-scroll-pane");
        scrollPane.setContent(htmlCode);
        scrollPane.setFitToWidth(true);
        scrollPane.setPrefHeight(180);

        Button showHTMLButton = new Button("Produce HTML Code");
        root.setAlignment(Pos.CENTER);
        showHTMLButton.setOnAction((ActionEvent arg0) -> {
            htmlCode.setText(htmlEditor.getHtmlText());
        });

        root.getChildren().addAll(htmlEditor, showHTMLButton, scrollPane);
        scene.setRoot(root);

        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

The `getHTMLText` method called on the `HTMLEditor` object derives the edited content and presents it as an HTML string. This information is passed to the text area, so that you can observe, copy, and paste the produced HTML code. Figure 21–5 shows an HTML code of the text is being edited in the HTMLEditor sample.

Figure 21–5 Obtaining the HTML Content



Similarly, you can obtain HTML code and save it in the file or send it to the `WebView` object to synchronize content in the editor and the embedded browser. See how this task is implemented in Example 21–5.

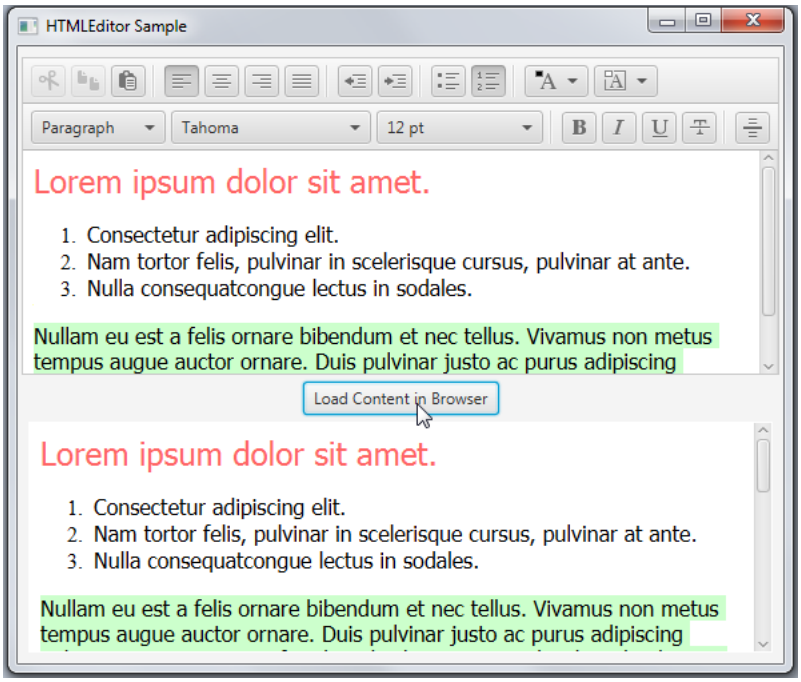
Example 21–5 Rendering Edited HTML Content in a Browser

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.VBox;
import javafx.scene.web.HTMLEditor;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
import javafx.stage.Stage;

public class HTMLEditorSample extends Application {
    private final String INITIAL_TEXT = "Lorem ipsum dolor sit "
            + "amet, consectetur adipiscing elit. Nam tortor felis, pulvinar "
            + "in scelerisque cursus, pulvinar at ante. Nulla consequat"
            + "congue lectus in sodales. Nullam eu est a felis ornare "
            + "bibendum et nec tellus. Vivamus non metus tempus augue auctor "
            + "ornare. Duis pulvinar justo ac purus adipiscing pulvinar. "
            + "Integer congue faucibus dapibus. Integer id nisl ut elit "
```

```
            + "aliquam sagittis gravida eu dolor. Etiam sit amet ipsum "
            + "sem.";

    @Override
    public void start(Stage stage) {
        stage.setTitle("HTMLEditor Sample");
        stage.setWidth(650);
        stage.setHeight(500);
        Scene scene = new Scene(new Group());

        VBox root = new VBox();
        root.setPadding(new Insets(8, 8, 8, 8));
        root.setSpacing(5);
        root.setAlignment(Pos.BOTTOM_LEFT);

        final HTMLEditor htmlEditor = new HTMLEditor();
        htmlEditor.setPrefHeight(245);
        htmlEditor.setHtmlText(INITIAL_TEXT);

        final WebView browser = new WebView();
        final WebEngine webEngine = browser.getEngine();

        ScrollPane scrollPane = new ScrollPane();
        scrollPane.getStyleClass().add("noborder-scroll-pane");
        scrollPane.setStyle("-fx-background-color: white");
        scrollPane.setContent(browser);
        scrollPane.setFitToWidth(true);
        scrollPane.setPrefHeight(180);

        Button showHTMLButton = new Button("Load Content in Browser");
        root.setAlignment(Pos.CENTER);
        showHTMLButton.setOnAction((ActionEvent arg0) -> {
            webEngine.loadContent(htmlEditor.getHtmlText());
        });

        root.getChildren().addAll(htmlEditor, showHTMLButton, scrollPane);
        scene.setRoot(root);

        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

HTML code received from the `htmlEditor` component is loaded in the `WebEngine` object that specifies the content for the embedded browser. Each time a user clicks the Load Content in Browser button, the edited content is updated in the browser. Figure 21–6 demonstrates Example 21–5 in action.

Figure 21–6 Loading Content in a Browser



You can use the `Text` component to add non-editing text content to your UI. See Using Text in JavaFX for more information about the `Text` component.

Related API Documentation

- `HTMLEditor`
- `WebView`
- `WebEngine`
- `Label`
- `Button`
- `TextField`
- `ChoiceBox`
- `ScrollPane`
- `TextFlow`

22

Tooltip

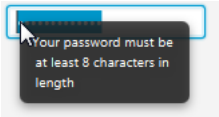
In this chapter, you learn about the tooltip, the control that can be set for any UI control to appear when that control is hovered over by the mouse cursor.

The `Tooltip` class represents a common UI component that is typically used to display additional information about the UI control. The tooltip can be set on any control by calling the `setTooltip` method.

The tooltip has two different states: activated and showing. When the tooltip is activated, the mouse moves over a control. When the tooltip is in the showing state, it actually appears. A shown tooltip is also activated. There is usually some delay between when the Tooltip becomes activated and when it is actually shown.

A password field with a tooltip is shown in Figure 22–1.

Figure 22–1 Tooltip Added to a Password Field



Creating a Tooltip

Study the code fragment in Example 22–1 that creates the password field with a tooltip in the JavaFX application shown in the preceding figure.

Example 22–1 Adding a Tooltip to the Password Field

```
final PasswordField pf = new PasswordField();
final Tooltip tooltip = new Tooltip();
tooltip.setText(
    "\nYour password must be\n" +
    "at least 8 characters in length\n"  +
);
pf.setTooltip(tooltip);
```

Each UI control from the `javafx.scene.control` package has the `setTooltip` method to add a tooltip. You can define a text caption within a `Tooltip` constructor or by using the `setText` method.

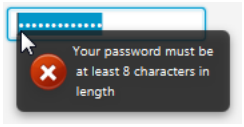
Because the `Tooltip` class is an extension of the `Labeled` class, you can add not only a text caption, but a graphical icon as well. The code fragment in Example 22–2 adds an icon to the tooltip for the password field.

Example 22–2 Adding an Icon to a Tooltip

```
Image image = new Image(
    getClass().getResourceAsStream("warn.png")
);
tooltip.setGraphic(new ImageView(image));
```

After you add this code fragment to the application, and the code is compiled, and run, the tooltip shown in Figure 22–2 appears.

Figure 22–2 Tooltip with an Icon

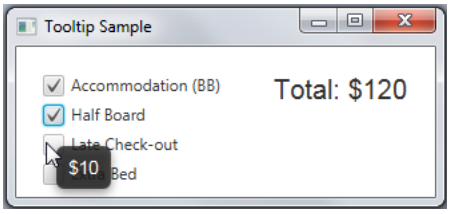


A tooltip can contain not only additional or auxiliary information, but it can also present data.

Presenting Application Data in Tooltips

The application in Figure 22–3 uses information presented in the tooltips to calculate the total cost of the hotel stay

Figure 22–3 Calculating Hotel Rates



Each checkbox is accompanied by a tooltip. Each tooltip displays the rate for a particular booking option. If a user selects a checkbox, the corresponding value is added to the total. If a selected checkbox is deselected, the corresponding value is deducted from the total.

Review the source code of the application shown in Example 22–3.

Example 22–3 Using Tooltips to Calculate Hotel Rates

```
import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.control.Label;
import javafx.scene.control.Tooltip;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;
```

```
public class TooltipSample extends Application {

    final static String[] rooms = new String[]{
        "Accommodation (BB)",
        "Half Board",
        "Late Check-out",
        "Extra Bed"
    };
    final static Integer[] rates = new Integer[]{
        100, 20, 10, 30
    };
    final CheckBox[] cbs = new CheckBox[rooms.length];
    final Label total = new Label("Total: $0");
    Integer sum = 0;

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Tooltip Sample");
        stage.setWidth(330);
        stage.setHeight(150);

        total.setFont(new Font("Arial", 20));

        for (int i = 0; i < rooms.length; i++) {
            final CheckBox cb = cbs[i] = new CheckBox(rooms[i]);
            final Integer rate = rates[i];
            final Tooltip tooltip = new Tooltip("$" + rates[i].toString());
            tooltip.setFont(new Font("Arial", 16));
            cb.setTooltip(tooltip);

        cb.selectedProperty().addListener(
                (ObservableValue<? extends Boolean> ov, Boolean old_val,
                Boolean new_val) -> {
                    if (cb.isSelected()) {
                        sum = sum + rate;
                    } else {
                        sum = sum - rate;
                    }
                    total.setText("Total: $" + sum.toString());
                }
            );
        }

        VBox vbox = new VBox();
        vbox.getChildren().addAll(cbs);
        vbox.setSpacing(5);
        HBox root = new HBox();
        root.getChildren().add(vbox);
        root.getChildren().add(total);
        root.setSpacing(40);
        root.setPadding(new Insets(20, 10, 10, 20));

        ((Group) scene.getRoot()).getChildren().add(root);
```

```
        stage.setScene(scene);
        stage.show();
    }
}
```

The code line in Example 22–4 was used in Example 22–3 to create a tooltip and assign a text caption to it. The `Integer` value of the option price was converted into a `String` value.

Example 22–4 Setting the Value for a Tooltip

```
final Tooltip tooltip = new Tooltip("$" + rates[i].toString())
```

You can alter visual appearance of a tooltip by applying CSS.

Related API Documentation

- `Tooltip`
- `Labeled`

23

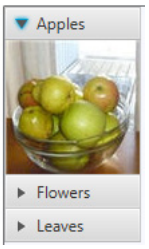
Titled Pane and Accordion

This chapter explains how to use a combination of the accordion and title panes in your JavaFX applications.

A titled pane is a panel with a title. It can be opened and closed, and it can encapsulate any `Node` such as UI controls or images, and groups of elements added to a layout container.

Titled panes can be grouped by using the accordion control, which enables you to create multiple panes and display them one at a time. Figure 23–1 shows an accordion control that combines three titled panes.

Figure 23–1 Accordion with Three Titled Panes



Use the `Accordion` and `TitledPane` classes in the JavaFX SDK API to implement these controls in your applications.

Creating Titled Panes

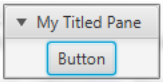
To create a `TitledPane` control define a title and some content. You can do this by using the two-parameter constructor of the `TitledPane` class, or by applying the `setText` and `setContent` methods. Both ways are shown in Example 23–1.

Example 23–1 Declaring a TitledPane Object

```
//using a two-parameter constructor
TitledPane tp = new TitledPane("My Titled Pane", new Button("Button"));
//applying methods
TitledPane tp = new TitledPane();
tp.setText("My Titled Pane");
tp.setContent(new Button("Button"));
```

Compiling and running an application with either of the code fragments produces the control shown in Figure 23–2.

Figure 23–2 Titled Pane with a Button



Do not explicitly set the minimal, maximum, or preferred height of a titled pane, as this may lead to unexpected behavior when the titled pane is opened or closed.

The code fragment shown in Example 23–2 adds several controls to the titled pane by putting them into the `GridPane` layout container.

Example 23–2 Titled Pane with the GridPane Layout Container

```
TitledPane gridTitlePane = new TitledPane();
GridPane grid = new GridPane();
grid.setVgap(4);
grid.setPadding(new Insets(5, 5, 5, 5));
grid.add(new Label("First Name: "), 0, 0);
grid.add(new TextField(), 1, 0);
grid.add(new Label("Last Name: "), 0, 1);
grid.add(new TextField(), 1, 1);
grid.add(new Label("Email: "), 0, 2);
grid.add(new TextField(), 1, 2);
gridTitlePane.setText("Grid");
gridTitlePane.setContent(grid);
```

When you run and compile an application with this code fragment, the output shown in Figure 23–3 appears.

Figure 23–3 Titled Pane that Contains Several Controls

You can define the way a titled pane opens and closes. By default, all titled panes are collapsible and their movements are animated. If your application prohibits closing a titled pane, use the `setCollapsible` method with the `false` value. You can also disable animated opening by applying the `setAnimated` method with the `false` value. The code fragment shown in Example 23–3 implements these tasks.

Example 23–3 Adjusting the Style of a Titled Pane

```
TitledPane tp = new TitledPane();
//prohibit closing
tp.setCollapsible(false);
//prohibit animating
tp.setAnimated(false);
```

Adding Titled Panes to an Accordion

In your applications, you can use titled panes as standalone elements, or you can combine them in a group by using the `Accordion` control. Do not explicitly set the minimal, maximum, or preferred height of an accordion, as this may lead to unexpected behavior when one of its titled pane is opened.

Adding several titled panes to an accordion is similar to adding toggle buttons to a toggle group: only one titled pane can be opened in an accordion at a time. Example 23–4 creates three titled panes and adds them to an accordion.

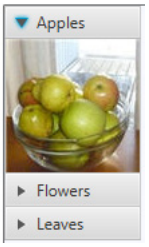
Example 23–4 Accordion and Three Titled Panes

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Accordion;
import javafx.scene.control.TitledPane;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;

public class TitledPaneSample extends Application {
    final String[] imageNames = new String[]{"Apples", "Flowers", "Leaves"};
    final Image[] images = new Image[imageNames.length];
    final ImageView[] pics = new ImageView[imageNames.length];
    final TitledPane[] tps = new TitledPane[imageNames.length];

    public static void main(String[] args) {
        launch(args);
    }

    @Override public void start(Stage stage) {
        stage.setTitle("TitledPane");
        Scene scene = new Scene(new Group(), 80, 180);

        final Accordion accordion = new Accordion ();

        for (int i = 0; i < imageNames.length; i++) {
            images[i] = new
                Image(getClass().getResourceAsStream(imageNames[i] + ".jpg"));
            pics[i] = new ImageView(images[i]);
            tps[i] = new TitledPane(imageNames[i],pics[i]);
        }
        accordion.getPanes().addAll(tps);
        accordion.setExpandedPane(tps[0]);

        Group root = (Group)scene.getRoot();
        root.getChildren().add(accordion);
        stage.setScene(scene);
        stage.show();
    }
}
```

Three titled panes are created within the loop. Content for each titled pane is defined as an `ImageView` object. The titled panes are added to the accordion by using the `getPanes` and `addAll` methods. You can use the `add` method instead of the `addAll` method to add a single titled pane.

By default, all the titled panes are closed when the application starts. The `setExpandedPane` method in Example 23–4 specifies that the titled pane with the Apples picture will be opened when you run the sample, as shown in Figure 23–4.

Figure 23–4 Accordion with Three Titled Panes



Processing Events for an Accordion with Titled Panes

You can use titled panes and accordions to present different data in your applications. Example 23–5 creates a standalone titled pane with the `GridPane` layout container and three titled panes combined by using the accordion. The standalone titled pane contains UI elements of an email client. The accordion enables the selection of an image to appear in the Attachment field of the Grid titled pane.

Example 23–5 Implementing ChangeListener for an Accordion

```
import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Accordion;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.control.TitledPane;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class TitledPaneSample extends Application {
    final String[] imageNames = new String[]{"Apples", "Flowers", "Leaves"};
    final Image[] images = new Image[imageNames.length];
    final ImageView[] pics = new ImageView[imageNames.length];
    final TitledPane[] tps = new TitledPane[imageNames.length];
    final Label label = new Label("N/A");

    public static void main(String[] args) {
        launch(args);
    }

    @Override public void start(Stage stage) {
        stage.setTitle("TitledPane");
        Scene scene = new Scene(new Group(), 800, 250);

        // --- GridPane container
        TitledPane gridTitlePane = new TitledPane();
```

```
        GridPane grid = new GridPane();
        grid.setVgap(4);
        grid.setPadding(new Insets(5, 5, 5, 5));
        grid.add(new Label("To: "), 0, 0);
        grid.add(new TextField(), 1, 0);
        grid.add(new Label("Cc: "), 0, 1);
        grid.add(new TextField(), 1, 1);
        grid.add(new Label("Subject: "), 0, 2);
        grid.add(new TextField(), 1, 2);
        grid.add(new Label("Attachment: "), 0, 3);
        grid.add(label,1, 3);
        gridTitlePane.setText("Grid");
        gridTitlePane.setContent(grid);

        // --- Accordion
        final Accordion accordion = new Accordion ();
        for (int i = 0; i < imageNames.length; i++) {
            images[i] = new
                Image(getClass().getResourceAsStream(imageNames[i] + ".jpg"));
            pics[i] = new ImageView(images[i]);
            tps[i] = new TitledPane(imageNames[i],pics[i]);
        }
        accordion.getPanes().addAll(tps);
        accordion.expandedPaneProperty().addListener(
            (ObservableValue<? extends TitledPane> ov, TitledPane old_val,
            TitledPane new_val) -> {
                if (new_val != null) {
                    label.setText(accordion.getExpandedPane().getText() +
                            ".jpg");
                }
        });

        HBox hbox = new HBox(10);
        hbox.setPadding(new Insets(20, 0, 0, 20));
        hbox.getChildren().setAll(gridTitlePane, accordion);

        Group root = (Group)scene.getRoot();
        root.getChildren().add(hbox);
        stage.setScene(scene);
        stage.show();
    }
}
```

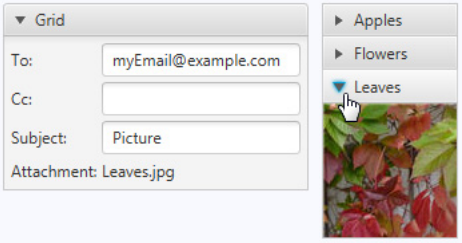
When a user opens a titled pane in the accordion, the `expandedPaneProperty` of the accordion changes. The expanded titled pane in the accordion is used to construct a file name of the attachment. This file name is set as text of the corresponding `Label` object.

Figure 23–5 shows how the application looks after its start. The Attachment label contains "N/A," because none of the titled panes are selected.

Figure 23–5 Initial View of the TitledPaneSample Application

If you expand the Leaves titled pane, the Attachment label will contain "Leaves.jpg," as shown in Figure 23–6.

Figure 23–6 TitledPaneSample Application with the Leaves Titled Pane Expanded



Because the `TitledPane` and `Accordion` classes are both extensions of the `Node` class, you can apply visual effects or transformations to them. You can also change the appearance of the controls by applying CSS styles.

Related API Documentation

- `TitledPane`
- `Accordion`
- `Label`
- `GridPane`
- `TextField`

24

Menu

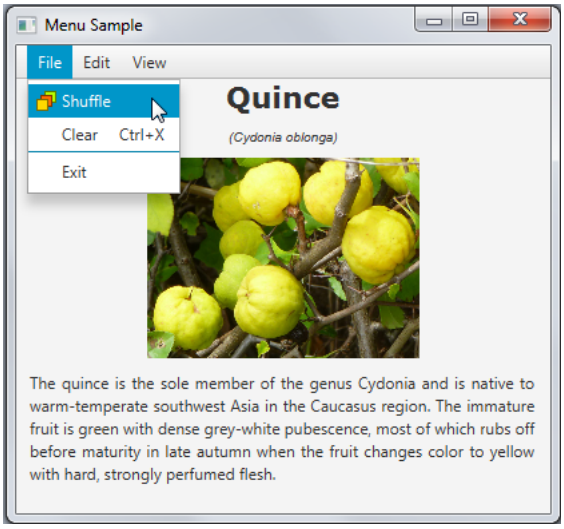
This chapter explains how to create menus and menu bars, add menu items, group the menus into categories, create submenus, and set context menus.

You can use the following classes of the JavaFX API to build menus in your JavaFX application.

- MenuBar
- MenuItem
  - Menu
  - CheckMenuItem
  - RadioMenuItem
  - Menu
  - CustomMenuItem
    - SeparatorMenuItem
- ContextMenu

Figure 24–1 shows a screen capture of an application with a typical menu bar.

Figure 24–1 Application with a Menu Bar and Three Menu Categories



Building Menus in JavaFX Applications

A menu is a list of actionable items that can be displayed upon a user's request. When a menu is visible, users can select one menu item at time. After a user clicks an item, the menu returns to the hidden mode. By using menus, you can save space in your application user interface (UI) by placing in menus the functionality that does not always need to be visible.

The menus in a menu bar are typically grouped into categories. The coding pattern is to declare a menu bar, define the category menus, and populate the category menus with menu items. Use the following menu item classes when building menus in your JavaFX applications:

- `MenuItem` – to create one actionable option
- `Menu` – to create a submenu
- `RadioButtonItem` – to create a mutually exclusive selection
- `CheckMenuItem` – to create an option that can be toggled between selected and unselected states

To separate menu items within one category, use the `SeparatorMenuItem` class.

The menus organized by categories in a menu bar are typically located at the top of the window, leaving the rest of the scene for crucial UI elements. If, for some reasons, you cannot allot any visual part of your UI for a menu bar, you can use context menus that the user opens with a mouse click.

Creating a Menu Bar

Although a menu bar can be placed elsewhere in the user interface, typically it is located at the top of the UI and it contains one or more menus. The menu bar automatically resizes to fit the width of the application window. By default, each menu added to the menu bar is represented by a button with the text value.

Consider an application that renders reference information about plants such as their name, binomial name, picture, and a brief description. You can create three menu categories: File, Edit, and View, and populate them with the menu items. Example 24–1 shows the source code of such an application with the menu bar added.

Example 24–1 Menu Sample Application

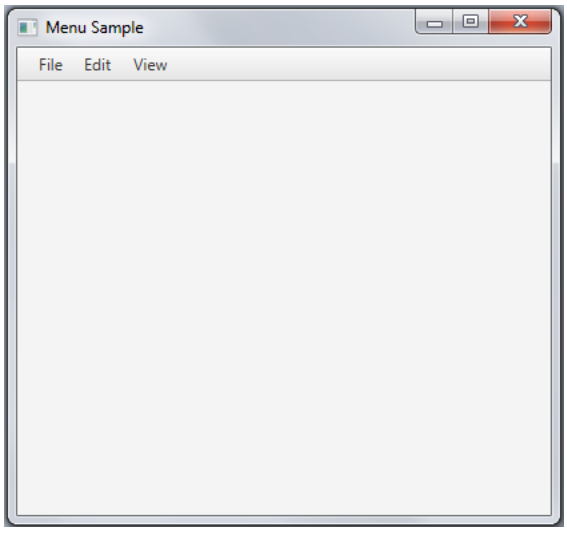
```
import java.util.AbstractMap.SimpleEntry;
import java.util.Map.Entry;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.effect.DropShadow;
import javafx.scene.effect.Effect;
import javafx.scene.effect.Glow;
import javafx.scene.effect.SepiaTone;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class MenuSample extends Application {
```

```
    final PageData[] pages = new PageData[] {
        new PageData("Apple",
            "The apple is the pomaceous fruit of the apple tree, species Malus "
            + "domestica in the rose family (Rosaceae). It is one of the most "
            + "widely cultivated tree fruits, and the most widely known of "
            + "the many members of genus Malus that are used by humans. "
            + "The tree originated in Western Asia, where its wild ancestor, "
            + "the Alma, is still found today.",
            "Malus domestica"),
        new PageData("Hawthorn",
            "The hawthorn is a large genus of shrubs and trees in the rose "
            + "family, Rosaceae, native to temperate regions of the Northern "
            + "Hemisphere in Europe, Asia and North America. "
            + "The name hawthorn was "
            + "originally applied to the species native to northern Europe, "
            + "especially the Common Hawthorn C. monogyna, and the unmodified "
            + "name is often so used in Britain and Ireland.",
            "Crataegus monogyna"),
        new PageData("Ivy",
            "The ivy is a flowering plant in the grape family (Vitaceae) native
to"
            + " eastern Asia in Japan, Korea, and northern and eastern China. "
            + "It is a deciduous woody vine growing to 30 m tall or more given "
            + "suitable support,  attaching itself by means of numerous small "
            + "branched tendrils tipped with sticky disks.",
            "Parthenocissus tricuspidata"),
        new PageData("Quince",
            "The quince is the sole member of the genus Cydonia and is native to "
            + "warm-temperate southwest Asia in the Caucasus region. The "
            + "immature fruit is green with dense grey-white pubescence, most "
            + "of which rubs off before maturity in late autumn when the fruit "
            + "changes color to yellow with hard, strongly perfumed flesh.",
            "Cydonia oblonga")
    };

    final String[] viewOptions = new String[] {
        "Title",
        "Binomial name",
        "Picture",
        "Description"
    };

    final Entry<String, Effect>[] effects = new Entry[] {
        new SimpleEntry<>("Sepia Tone", new SepiaTone()),
        new SimpleEntry<>("Glow", new Glow()),
        new SimpleEntry<>("Shadow", new DropShadow())
    };

    final ImageView pic = new ImageView();
    final Label name = new Label();
    final Label binName = new Label();
    final Label description = new Label();

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        stage.setTitle("Menu Sample");
```

```
        Scene scene = new Scene(new VBox(), 400, 350);

        MenuBar menuBar = new MenuBar();

        // --- Menu File
        Menu menuFile = new Menu("File");

        // --- Menu Edit
        Menu menuEdit = new Menu("Edit");

        // --- Menu View
        Menu menuView = new Menu("View");

        menuBar.getMenus().addAll(menuFile, menuEdit, menuView);


        ((VBox) scene.getRoot()).getChildren().addAll(menuBar);

        stage.setScene(scene);
        stage.show();
    }


    private class PageData {
        public String name;
        public String description;
        public String binNames;
        public Image image;
        public PageData(String name, String description, String binNames) {
            this.name = name;
            this.description = description;
            this.binNames = binNames;
            image = new Image(getClass().getResourceAsStream(name + ".jpg"));
        }
    }
}
```

Unlike other UI Controls, the `Menu` class and other extensions of the `MenuItem` class do not extend the `Node` class. They cannot be added directly to the application scene and remain invisible until added to the menu bar through the `getMenus` method.

Figure 24–2 Menu Bar is Added to the Application



You can navigate through the menus by using the arrow keys of the keyboard. However, when you select a menu, no action is performed, because the behavior for the menus is not defined yet.

Adding Menu Items

Set the functionality for the File menu by adding the following items:

- Shuffle – to load reference information about plants
- Clear – to remove the reference information and clear the scene
- Separator – to detach menu items
- Exit – to exit the application

Bold lines in Example 24–2 create a Shuffle menu by using the `MenuItem` class and add graphical components to the application scene. The `MenuItem` class enables creating an actionable item with text and graphics. The action performed on a user click is defined by the `setOnAction` method, similar to the `Button` class.

Example 24–2 Adding the Shuffle Menu Item with Graphics

```
import java.util.AbstractMap.SimpleEntry;
import java.util.Map.Entry;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.effect.DropShadow;
import javafx.scene.effect.Effect;
import javafx.scene.effect.Glow;
import javafx.scene.effect.SepiaTone;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
```

```
import javafx.scene.text.TextAlignment;
import javafx.stage.Stage;

public class MenuSample extends Application {

    final PageData[] pages = new PageData[] {
        new PageData("Apple",
            "The apple is the pomaceous fruit of the apple tree, species Malus "
            +"domestica in the rose family (Rosaceae). It is one of the most "
            +"widely cultivated tree fruits, and the most widely known of "
            +"the many members of genus Malus that are used by humans. "
            +"The tree originated in Western Asia, where its wild ancestor, "
            +"the Alma, is still found today.",
            "Malus domestica"),
        new PageData("Hawthorn",
            "The hawthorn is a large genus of shrubs and trees in the rose "
            + "family, Rosaceae, native to temperate regions of the Northern "
            + "Hemisphere in Europe, Asia and North America. "
            + "The name hawthorn was "
            + "originally applied to the species native to northern Europe, "
            + "especially the Common Hawthorn C. monogyna, and the unmodified "
            + "name is often so used in Britain and Ireland.",
            "Crataegus monogyna"),
        new PageData("Ivy",
            "The ivy is a flowering plant in the grape family (Vitaceae) native"
            +" to eastern Asia in Japan, Korea, and northern and eastern China."
            +" It is a deciduous woody vine growing to 30 m tall or more given "
            +"suitable support,  attaching itself by means of numerous small "
            +"branched tendrils tipped with sticky disks.",
            "Parthenocissus tricuspidata"),
        new PageData("Quince",
            "The quince is the sole member of the genus Cydonia and is native"
            +" to warm-temperate southwest Asia in the Caucasus region. The "
            +"immature fruit is green with dense grey-white pubescence, most "
            +"of which rubs off before maturity in late autumn when the fruit "
            +"changes color to yellow with hard, strongly perfumed flesh.",
            "Cydonia oblonga")
    };

    final String[] viewOptions = new String[] {
        "Title",
        "Binomial name",
        "Picture",
        "Description"
    };

    final Entry<String, Effect>[] effects = new Entry[] {
        new SimpleEntry<>("Sepia Tone", new SepiaTone()),
        new SimpleEntry<>("Glow", new Glow()),
        new SimpleEntry<>("Shadow", new DropShadow())
    };

    final ImageView pic = new ImageView();
    final Label name = new Label();
    final Label binName = new Label();
    final Label description = new Label();
    private int currentIndex = -1;

    public static void main(String[] args) {
        launch(args);
```

```
}

@Override
public void start(Stage stage) {
    stage.setTitle("Menu Sample");
    Scene scene = new Scene(new VBox(), 400, 350);
    scene.setFill(Color.OLDLACE);

    name.setFont(new Font("Verdana Bold", 22));
    binName.setFont(new Font("Arial Italic", 10));
    pic.setFitHeight(150);
    pic.setPreserveRatio(true);
    description.setWrapText(true);
    description.setTextAlignment(TextAlignment.JUSTIFY);

    shuffle();

    MenuBar menuBar = new MenuBar();

    final VBox vbox = new VBox();
    vbox.setAlignment(Pos.CENTER);
    vbox.setSpacing(10);
    vbox.setPadding(new Insets(0, 10, 0, 10));
    vbox.getChildren().addAll(name, binName, pic, description);

    // --- Menu File
    Menu menuFile = new Menu("File");
    MenuItem add = new MenuItem("Shuffle",
        new ImageView(new Image("menusample/new.png")));
    add.setOnAction((ActionEvent t) -> {
        shuffle();
        vbox.setVisible(true);
    });

    menuFile.getItems().addAll(add);

    // --- Menu Edit
    Menu menuEdit = new Menu("Edit");

    // --- Menu View
    Menu menuView = new Menu("View");

    menuBar.getMenus().addAll(menuFile, menuEdit, menuView);
    ((VBox) scene.getRoot()).getChildren().addAll(menuBar, vbox);
    stage.setScene(scene);
    stage.show();
}

private void shuffle() {
    int i = currentIndex;
    while (i == currentIndex) {
        i = (int) (Math.random() * pages.length);
    }
    pic.setImage(pages[i].image);
    name.setText(pages[i].name);
    binName.setText("(" + pages[i].binNames + ")");
    description.setText(pages[i].description);
    currentIndex = i;
}
```

```
    private class PageData {
        public String name;
        public String description;
        public String binNames;
        public Image image;
        public PageData(String name, String description, String binNames) {
            this.name = name;
            this.description = description;
            this.binNames = binNames;
            image = new Image(getClass().getResourceAsStream(name + ".jpg"));
        }
    }
}
```

When a user selects the Shuffle menu item, the `shuffle` method called within `setOnAction` specifies the title, the binomial name, a picture of the plant, and its description by calculating the index of the elements in the corresponding arrays.

The Clear menu item is used to erase the application scene. You can implement this by making the `VBox` container with the GUI elements invisible as shown in Example 24–3.

Example 24–3 Creating the Clear Menu Item with Accelerator

```
MenuItem clear = new MenuItem("Clear");
clear.setAccelerator(KeyCombination.keyCombination("Ctrl+X"));
clear.setOnAction((ActionEvent t) -> {
    vbox.setVisible(false);
});
```

Implementation of the `MenuItem` class enables developers to set a menu accelerator, a key combination that performs the same action as the menu item. With the Clear menu, users can either select the action from the File menu category or press Control Key and X key simultaneously.

The Exit menu closes the application window. Set `System.exit(0)` as an action for this menu item as shown in Example 24–4.

Example 24–4 Creating the Exit Menu Item

```
MenuItem exit = new MenuItem("Exit");
exit.setOnAction((ActionEvent t) -> {
    System.exit(0);
});
```

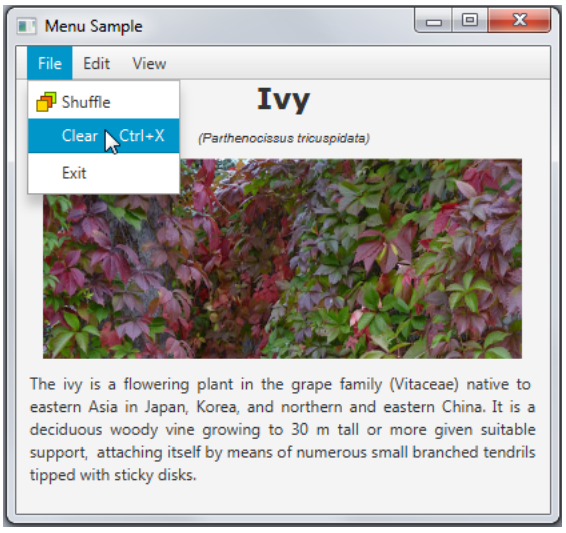
Use the `getItems` method shown in Example 24–5 to add the newly created menu items to the File menu. You can create a separator menu item and add it within the `getItems` method to visually detach the Exit menu item.

Example 24–5 Adding Menu Items

```
menuFile.getItems().addAll(add, clear, new SeparatorMenuItem(), exit);
```

Add Example 24–2, Example 24–3, Example 24–4, and Example 24–5 to the Menu Sample application, and then compile and run it. Select the Shuffle menu item to load reference information about different plants. Then clear the scene (Clear), and close the application (Exit). Figure 24–3 shows selection of the Clear menu item.

Figure 24–3 File Menu with Three Menu Items



With the View menu, you can hide and show elements of reference information. Implement the `createMenuItem` method and call it within the `start` method to create four `CheckMenuItem` objects. Then add newly created check menu items to the View menu to toggle visibility of the title, binomial name, picture of the plant, and its description. Example 24–6 shows two code fragments that implement these tasks.

Example 24–6 Applying the CheckMenuItem Class to Create Toggle Options

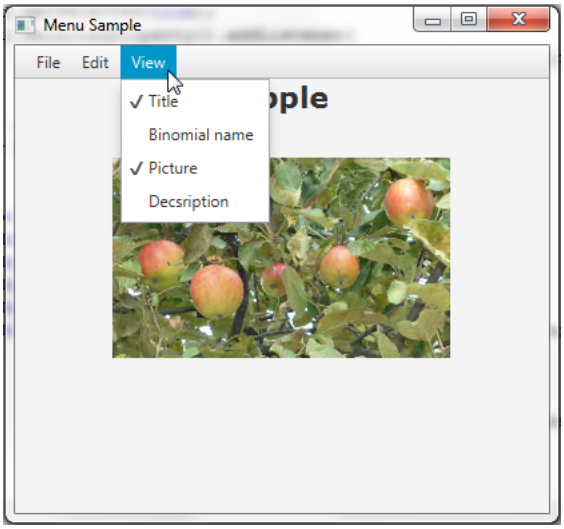
```
// --- Creating four check menu items within the start method
CheckMenuItem titleView = createMenuItem ("Title", name);
CheckMenuItem binNameView = createMenuItem ("Binomial name", binName);
CheckMenuItem picView = createMenuItem ("Picture", pic);
CheckMenuItem descriptionView = createMenuItem ("Description", description);
menuView.getItems().addAll(titleView, binNameView, picView, descriptionView);

...

// The createMenuItem method
private static CheckMenuItem createMenuItem (String title, final Node node){
    CheckMenuItem cmi = new CheckMenuItem(title);
    cmi.setSelected(true);
    cmi.selectedProperty().addListener(
        (ObservableValue<? extends Boolean> ov, Boolean old_val,
        Boolean new_val) -> {
            node.setVisible(new_val);
    });
    return cmi;
}
```

The `CheckMenuItem` class is an extension of the `MenuItem` class. It can be toggled between selected and deselected states. When selected, a check menu item shows a check mark.

Example 24–6 creates four `CheckMenuItem` objects and processes the changing of their `selectedProperty` property. When, for example, a user deselects the `picView` item, the `setVisible` method receives the `false` value, the picture of the plant becomes invisible. When you add this code fragment to the application, compile, and run the application, you can experiment with selecting and deselecting the menu items.

Figure 24–4 shows the application in the moment when the title and picture of the plant are shown, but its binomial name and description are hidden.

Figure 24–4 Using Check Menu Items



Creating Submenus

For the Edit menu, define two menu items: Picture Effect and No Effects. The Picture Effect menu item is designed as a submenu with three items to set one of the three available visual effects. The No Effects menu item removes the selected effect and restores the initial state of the image.

Use the `RadioMenuItem` class to create the items of the submenu. Add the radio menu buttons to a toggle group to make the selection mutually exclusive. Example 24–7 implements these tasks.

Example 24–7 Creating a Submenu with Radio Menu Items

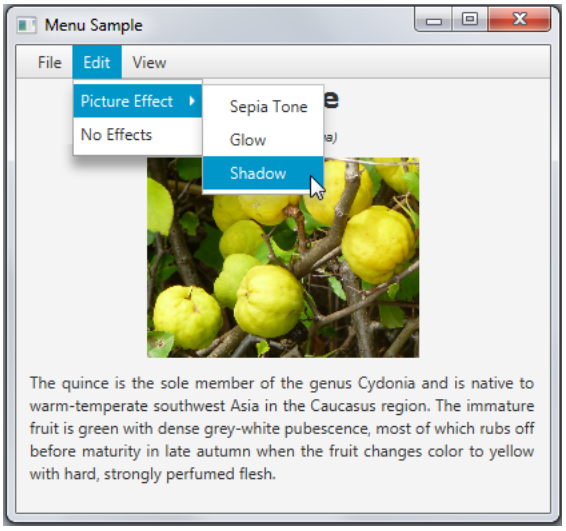
```
//Picture Effect menu
Menu menuEffect = new Menu("Picture Effect");
final ToggleGroup groupEffect = new ToggleGroup();
for (Entry<String, Effect> effect : effects) {
    RadioMenuItem itemEffect = new RadioMenuItem(effect.getKey());
    itemEffect.setUserData(effect.getValue());
    itemEffect.setToggleGroup(groupEffect);
    menuEffect.getItems().add(itemEffect);
}
//No Effects menu
final MenuItem noEffects = new MenuItem("No Effects");

noEffects.setOnAction((ActionEvent t) -> {
    pic.setEffect(null);
    groupEffect.getSelectedToggle().setSelected(false);
});

//Processing menu item selection
groupEffect.selectedToggleProperty().addListener(new ChangeListener<Toggle>() {
    public void changed(ObservableValue<? extends Toggle> ov,
        Toggle old_toggle, Toggle new_toggle) {
```

```
            if (groupEffect.getSelectedToggle() != null) {
                Effect effect =
                    (Effect) groupEffect.getSelectedToggle().getUserData();
                pic.setEffect(effect);
            }
        }
 });
groupEffect.selectedToggleProperty().addListener(
    (ObservableValue<? extends Toggle> ov, Toggle old_toggle,
    Toggle new_toggle) -> {
        if (groupEffect.getSelectedToggle() != null) {
             Effect effect =
                 (Effect) groupEffect.getSelectedToggle().getUserData();
             pic.setEffect(effect);
        }
});

//Adding items to the Edit menu
menuEdit.getItems().addAll(menuEffect, noEffects);
```

The `setUserData` method defines a visual effect for a particular radio menu item. When one of the items in the toggle group is selected, the corresponding effect is applied to the picture. When the No Effects menu item is selected, the `setEffect` method specifies the `null` value and no effects are applied to the picture.

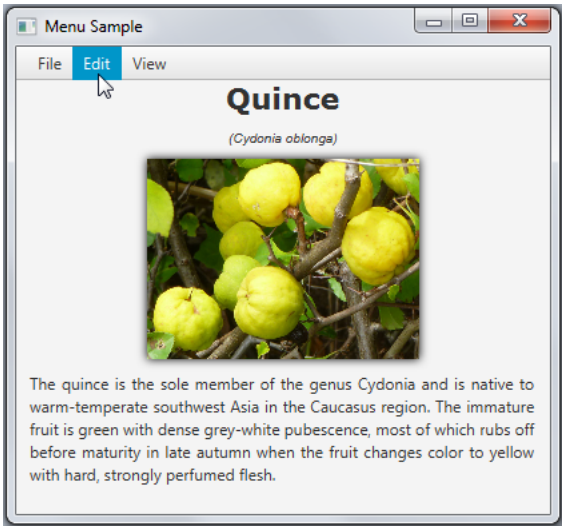
Figure 24–5 captures a moment when a user is selecting a Shadow menu item.

Figure 24–5 Submenu with Three Radio Menu Items



When the `DropShadow` effect is applied to the picture, it looks as shown in Figure 24–6.

Figure 24–6 Picture of Quince with a DropShadow Effect Applied



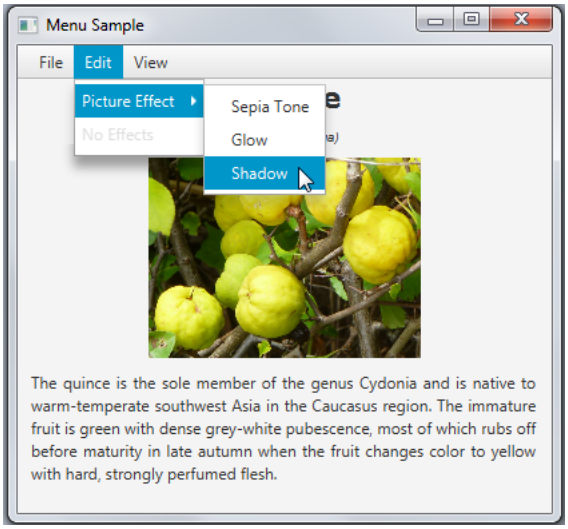
You can use the `setDisable` method of the `MenuItem` class to disable the No Effects menu when none of the effects are selected in the Picture Effect submenu. Modify Example 24–7 as shown in Example 24–8.

Example 24–8 Disabling a Menu Item

```
Menu menuEffect = new Menu("Picture Effect");
final ToggleGroup groupEffect = new ToggleGroup();
for (Entry<String, Effect> effect : effects) {
     RadioMenuItem itemEffect = new RadioMenuItem(effect.getKey());
     itemEffect.setUserData(effect.getValue());
     itemEffect.setToggleGroup(groupEffect);
     menuEffect.getItems().add(itemEffect);
}
final MenuItem noEffects = new MenuItem("No Effects");
noEffects.setDisable(true);
noEffects.setOnAction((ActionEvent t) -> {
    pic.setEffect(null);
    groupEffect.getSelectedToggle().setSelected(false);
    noEffects.setDisable(true);
});

groupEffect.selectedToggleProperty().addListener(
    (ObservableValue<? extends Toggle> ov, Toggle old_toggle,
    Toggle new_toggle) -> {
        if (groupEffect.getSelectedToggle() != null) {
            Effect effect =
                (Effect) groupEffect.getSelectedToggle().getUserData();
            pic.setEffect(effect);
            noEffects.setDisable(false);
        } else {
                noEffects.setDisable(true);
            }
});

menuEdit.getItems().addAll(menuEffect, noEffects);
```

When none of the `RadioMenuItem` options are selected, the No Effect menu item is disabled as shown in Figure 24–7. When a user selects one of the visual effects, the No Effects menu item is enabled.

Figure 24–7 Effect Menu Item Is Disabled



Adding Context Menus

When you cannot allocate any space of your user interface for a required functionality, you can use a context menu. A context menu is a pop-up window that appears in response to a mouse click. A context menu can contain one or more menu items.

In the Menu Sample application, set a context menu for the picture of the plant, so that users can copy the image.

Use the `ContextMenu` class to define the context menu as shown in Example 24–9.

Example 24–9 Defining a Context Menu

```
final ContextMenu cm = new ContextMenu();
MenuItem cmItem1 = new MenuItem("Copy Image");
cmItem1.setOnAction((ActionEvent e) -> {
    Clipboard clipboard = Clipboard.getSystemClipboard();
    ClipboardContent content = new ClipboardContent();
    content.putImage(pic.getImage());
    clipboard.setContent(content);
});

cm.getItems().add(cmItem1);
pic.addEventHandler(MouseEvent.MOUSE_CLICKED, (MouseEvent e) -> {
    if (e.getButton() == MouseButton.SECONDARY)
       cm.show(pic, e.getScreenX(), e.getScreenY());
});
```

When a user right clicks the `ImageView` object, the `show` method is called for the context menu to enable its showing.

The `setOnAction` method defined for the Copy Image item of the context menu creates a `Clipboard` object and adds the image as its content. Figure 24–8 captures a moment when a user is selecting the Copy Image context menu item.

Figure 24–8 Using the Context Menu

You can try to copy the image and paste it into in a graphical editor.

For further enhancements, you can add more menu items to the context menu and specify different actions. You can also create a custom menu by using the `CustomMenuItem` class. With this class you can embed an arbitrary node within a menu and specify, for example, a button or a slider as a menu item.

Related API Documentation

- `Menu`
- `MenuItem`
- `RadioMenuItem`
- `CheckMenuItem`
- `ContextMenu`
- `SeparatorMenuItem`
- `CustomMenuItem`

25

Color Picker

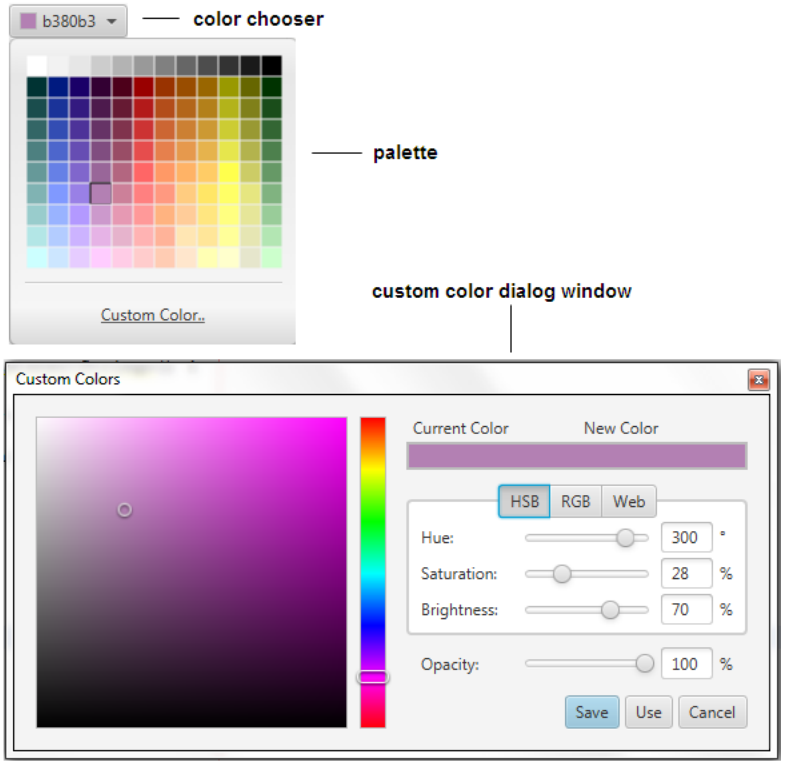
This chapter describes the `ColorPicker` control, provides its design overview, and explains how to use it in your JavaFX applications.

The color picker control in the JavaFX SDK is a typical user interface component that enables users to select a particular color from the available range, or set an additional color by specifying an RGB or HSB combination.

Design Overview

The `ColorPicker` control consists of the color chooser, color palette, and custom color dialog window. Figure 25–1 shows these elements.

Figure 25–1 Elements of a Color Picker Control

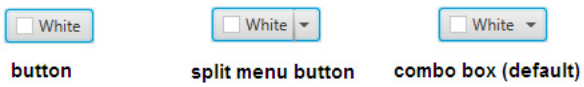


Color Chooser

The color chooser element of the color picker is the combo box with the enabled color indicator and the corresponding label shown at the top of Figure 25–1. The color indicator presents the currently selected color.

The implementation of the color picker control allows three looks of the color chooser element: button, split-menu-button, and combo box shown in Figure 25–2.

Figure 25–2 Views of a Color Chooser



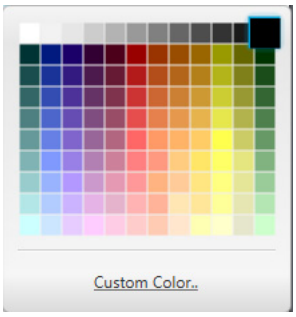
The button look provides a typical button control with the color indicator and the label. In the split-menu-button look, the button part of the control is separated from the drop-down menu. The combo-box look is the default appearance of the color chooser. It also has a drop-down menu but it is not separated from the button part.

To apply one of the looks, use the corresponding CSS classes. For more information about how to alter the appearance of a color picker, see Changing the Appearance of a Color Picker.

Color Palette

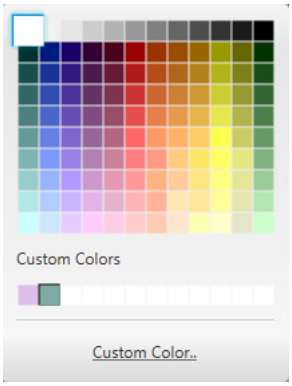
The color palette contains the predefined set of colors and the Custom Color link that points to the Custom Color dialog window. The initial appearance of the color palette is shown in Figure 25–3.

Figure 25–3 Initial Appearance of the Color Palette



If a custom color is already defined, this color is displayed in the Custom Color area in the color palette, as shown in Figure 25–4.

Figure 25–4 Color Palette with the Custom Color Area



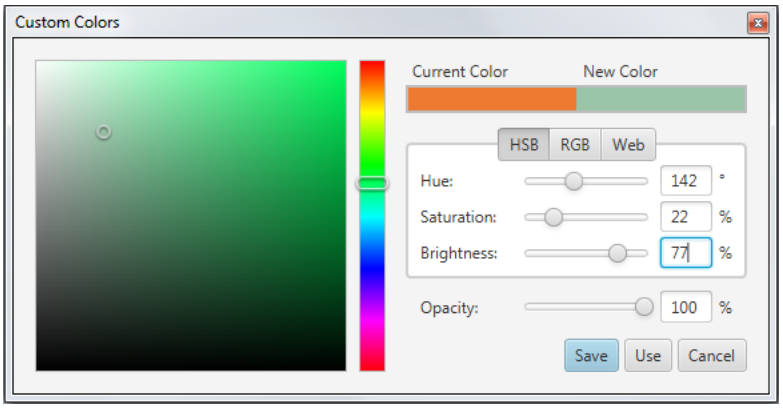
The color palette supports navigation by using the Up, Down, Left and Right keys.

The set of custom colors cannot be reloaded when the application starts again, unless saved within the application. Each color selected in the palette or in the custom color area is displayed in the color indicator of the color chooser. The color chooser label renders the corresponding hexadecimal web color value.

Custom Color Dialog Window

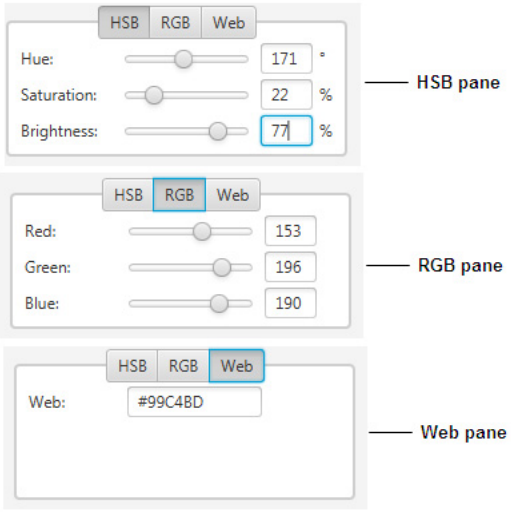
The Custom Color dialog window is a modal window that can be opened by clicking the corresponding link in the color palette. When the Custom Color window opens, it displays values of the color that is currently shown in the color indicator of the color chooser. Users can define a new color by moving the mouse cursor over the color area or over the vertical color bar, shown in Figure 25–5. Note that any time a user manipulates with the circle in the color area or with the rectangle in the color bar, the color value is automatically assigned to the corresponding property of the `ColorPicker` control.

Figure 25–5 Custom Colors Dialog Window



Another way to define a new color is to set the HSB (Hue Saturation Brightness) or RGB (Red Green Blue) values, or explicitly enter the web color value in the corresponding field. Figure 25–6 shows three panes for the custom color settings.

Figure 25–6 Color Setting Panes in the Custom Colors Dialog Window



Users can also set the transparency of the custom color by moving the Opacity slider or typing the value from 0 to 100.

When all the settings are done and the new color is specified, users can click Use to apply it, or Save to save the color to the custom color area.

Using a Color Picker

Use the `ColorPicker` class of the JavaFX SDK to build a color picker in your JavaFX application. You can add a color picker directly to the application scene, to a layout container, or to the application toolbar. Example 25–1 shows three ways to declare a `ColorPicker` object.

Example 25–1 Creating a Color Picker Control

```
//Empty contructor, the control appears with the default color, which is WHITE
ColorPicker colorPicker1 = new ColorPicker();
//Color constant set as the currently selected color
ColorPicker colorPicker2 = new ColorPicker(Color.BLUE);
//Web color value set as the currently selected color
ColorPicker colorPicker3 = new ColorPicker(Color.web("#ffcce6"));
```

Try the sample shown in Example 25–2 to evaluate the `ColorPicker` control in action.

Example 25–2 Using the ColorPicker Control to Alter the Color of the Text Component

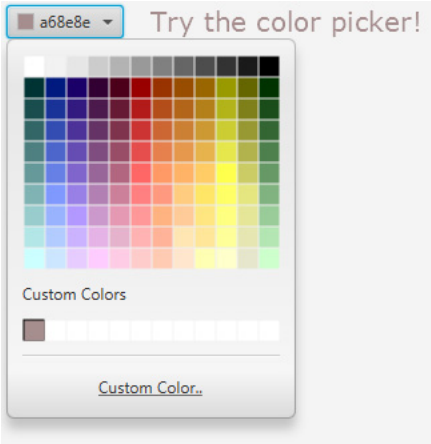
```
import javafx.application.Application;
import javafx.event.*;
import javafx.scene.Scene;
import javafx.scene.control.ColorPicker;
import javafx.geometry.Insets;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.text.*;
import javafx.stage.Stage;

public class ColorPickerSample extends Application {
    public static void main(String[] args) {
        launch(args);
```

```
    }

    @Override
    public void start(Stage stage) {
        stage.setTitle("ColorPicker");
        Scene scene = new Scene(new HBox(20), 400, 100);
        HBox box = (HBox) scene.getRoot();
        box.setPadding(new Insets(5, 5, 5, 5));
        final ColorPicker colorPicker = new ColorPicker();
        colorPicker.setValue(Color.CORAL);

        final Text text = new Text("Try the color picker!");
        text.setFont(Font.font ("Verdana", 20));
        text.setFill(colorPicker.getValue());


        colorPicker.setOnAction((ActionEvent t) -> {
            text.setFill(colorPicker.getValue());
        });

        box.getChildren().addAll(colorPicker, text);

        stage.setScene(scene);
        stage.show();
    }
}
```

This example creates a color picker and defines its behavior when the color is changed. The `Color` value obtained through the `getValue` method of the `ColorPicker` class is passed to the `setFill` method of the `Text` object. This is how the selected color is applied to the "Try the color picker!" text.

When you compile and run this sample, it produces the output shown in Figure 25–7. The figure captures the moment when a newly created custom color is applied to the `Text` component.

Figure 25–7 ColorPicker and a Text Component



Similarly, you can apply the selected color to the graphical `Node`. Example 25–3 shows how to use a color picker to model a T-shirt.

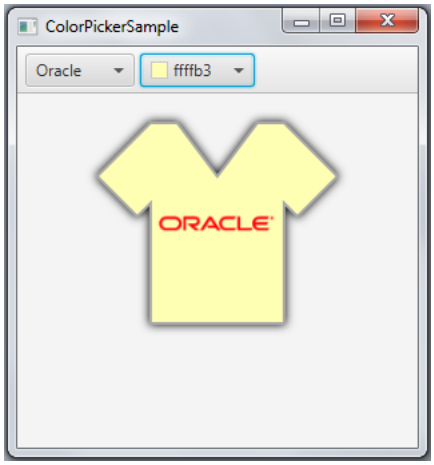
Example 25–3 Using ColorPicker to Alter the Color of a Graphical Object

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.ColorPicker;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.event.ActionEvent;
import javafx.scene.control.ComboBox;
import javafx.scene.control.ToolBar;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.SVGPath;
import javafx.stage.Stage;

public class ColorPickerSample extends Application {

    ImageView logo = new ImageView(
        new Image(getClass().getResourceAsStream("OracleLogo.png"))
    );

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        stage.setTitle("ColorPicker");

        Scene scene = new Scene(new VBox(20), 300, 300);
        VBox box = (VBox) scene.getRoot();

        ToolBar tb = new ToolBar();
        box.getChildren().add(tb);

        final ComboBox logoSamples = new ComboBox();
        logoSamples.getItems().addAll(
                "Oracle",
                "Java",
                "JavaFX",
                "Cup");
        logoSamples.setValue("Oracle");

        logoSamples.valueProperty().addListener(new ChangeListener<String>() {
            @Override
          public void changed(ObservableValue ov, String t, String t1) {
                logo.setImage(
                    new Image(getClass().getResourceAsStream(t1+"Logo.png"))
                );
            }
        });
        final ColorPicker colorPicker = new ColorPicker();
        tb.getItems().addAll(logoSamples, colorPicker);

        StackPane stack = new StackPane();
        box.getChildren().add(stack);
```

```
        final SVGPath svg = new SVGPath();
        svg.setContent("M70,50 L90,50 L120,90 L150,50 L170,50"
            + "L210,90 L180,120 L170,110 L170,200 L70,200 L70,110 L60,120 L30,90"
            + "L70,50");
        svg.setStroke(Color.DARKGREY);
        svg.setStrokeWidth(2);
        svg.setEffect(new DropShadow());
        svg.setFill(colorPicker.getValue());
        stack.getChildren().addAll(svg, logo);

        colorPicker.setOnAction((ActionEvent t) -> {
            svg.setFill(colorPicker.getValue());
        });

        stage.setScene(scene);
        stage.show();
    }
}
```

In this example, the color selected in the color picker and obtained through the `getValue` method is applied to the `SVGPath` object. This sample produces the output shown in Figure 25–8

Figure 25–8 ColorPickerSample Application



When you work with a color picker, you can obtain the created custom colors by calling the `getCustomColors()` method. It returns an `ObservableList` of the `Color` objects corresponding to the created colors. You cannot upload them to a color picker on the application start. However, you can set one of the custom colors as a `ColorPicker` value (shown in Example 25–4).

Example 25–4 Obtaining the Custom Colors

```
ObservableList<Color> customColors = colorPicker.getCustomColors();
colorPicker.setValue(customColors.get(index));
```

Changing the Appearance of a Color Picker

The default appearance of the color picker control is defined by the `com.sun.javafx.scene.control.skin.ColorPickerSkin` class. To apply an alternative

skin to the color pickers in your JavaFX application, redefine the `-fx-skin` property of the `color-picker` CSS class as shown in Example 25–5.

Example 25–5 Setting an Alternative Skin for a Color Picker

```
.color-picker {
    -fx-skin: "CustomSkin";
}
```

Use the `split-button` and `arrow-button` CSS classes to change the appearance of a color picker control in the JavaFX code, as shown in Example 25–6.

Example 25–6 Setting Appearance for a Color Picker

```
//Sets the split-menu-button
colorPicker.getStyleClass().add("split-button");
//Sets the button
colorPicker.getStyleClass().add("button");
```

You can also modify the default style of a color picker and customize its elements with various CSS classes defined in the `modena` style sheet. To view this file, go to the `\rt\lib\ext` directory under the directory in which the JavaFX SDK is installed. Use the following command to extract the style sheet from the JAR file: `jar -xf jfxrt.jar com/sun/javafx/scene/control/skin/modena/modena.css`. See Skinning JavaFX Applications with CSS for more information about CSS classes and properties. Example 25–7 shows how to alter the default background and the label of a color picker.

Example 25–7 Modifying the Default Appearance of a Color Picker

```
.color-picker {
    -fx-background-color: #669999;
    -fx-background-radius: 0 15 15 0;
}
.color-picker .color-picker-label .text {
    -fx-fill: #ccffcc;
}
```

Add these styles to the `ControlStyle.css` file and enable the style sheets in the JavaFX application by using the following code line: `scene.getStylesheets().add("colorpickersample/ControlStyle.css");`, then compile and the run the ColorPickerSample. The color picker should change its appearance as shown in Figure 25–9.

Figure 25–9 Modified Appearance of a Color Picker

Note that the `ColorPicker` class is an extension of the `ComboBoxBase` class and inherits its CSS properties. You can define new styles for the `combo-box-base` CSS style to unify the combo box and the color picker in the ColorPickerSample application. Replace the styles in the ControlStyle.css file with the styles shown in Example 25–8.

Example 25–8 Setting the Combo-Box-Base Styles

```
.tool-bar:horizontal {
    -fx-background-color: #b3e6b3;
}

.combo-box-base  {
    -fx-background-color: null;
}

.combo-box-base:hover  {
    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 8, 0.0 , 0 , 0 );
}
```

When you compile and run the ColorPickerSample application with the applied styles, the combo box and the color picker look as shown in Figure 25–10.

Figure 25–10 Unified Style of the Combo Box and Color Picker

Related API Documentation

- `ColorPicker`
- `ComboBoxBase`

26

Date Picker

This chapter provides an overview of the date data supported in JavaFX and describes the basic features of the `DatePicker` control.

JavaFX `DatePicker` is a control that enables selection of a day from the given calendar. It is used mainly on web sites or in applications that require users to enter a date. The `DatePicker` control consists of a combo box with a date field and a date chooser.

Working with Time Data and Date Formats

The new Date-Time API introduced in JDK 8 enables you to perform various operations with Date and Time data, including setting the calendar and local time across different time zones.

The basic package of the Date-Time API is `java.time`. It provides the following classes that define time in the calendar system based on the International Organization for Standardization (ISO) calendar.

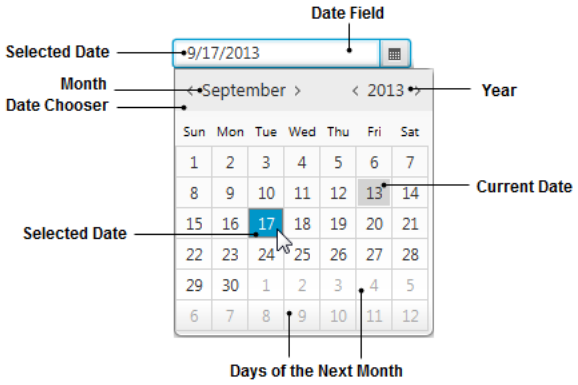
- `Clock` – a clock providing access to the current instant, date, and time using a time-zone
- `Duration` – a time-based amount of time
- `Instant` – an instantaneous point on the timeline
- `LocalDate` – a date without a time-zone in the ISO-8601 calendar system
- `LocalDateTime` – a date-time without a time zone in the ISO-8601 calendar system
- `LocalTime` – a time without time zone in the ISO-8601 calendar system
- `MonthDay` – a month day in the ISO-8601 calendar system
- `OffsetDateTime` – a date-time with an offset from Greenwich/UTC time in the ISO-8601 calendar system
- `OffsetTime` – a time with an offset from Greenwich/UTC time in the ISO-8601 calendar system
- `Period` – a date-based amount of time
- `Year` – a year in the ISO-8601 calendar system
- `YearMonth` – a year-month in the ISO-8601 calendar system
- `ZonedDateTime` – a date time with a time zone in the ISO-8601 calendar system
- `ZoneId` – a time zone ID
- `ZoneOffset` – a time zone offset from Greenwich/UTC time

See the Date-Time API trail of the Java Tutorials for more information about the available functionality and code samples.

Date Picker Design Overview

To enhance the user interfaces of JavaFX applications with the new capabilities of the JDK 8 Date-Time API, JavaFX SDK introduced the `DatePicker` control. The `DatePicker` control consists of an editable combo box (date field) and a calendar (date chooser) shown in Figure 26–1.

Figure 26–1 Example of a Date Picker Control



Adding a Date Picker to an Application UI

Use the `DatePicker` class available in the `javafx.scene.control` package to add a date picker component to the user interface (UI) of your JavaFX application as shown in Example 26–1.

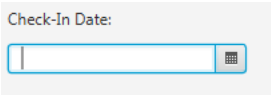
Example 26–1 Adding a Date Picker Component

```
import java.util.Locale;
import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.scene.Scene;
import javafx.scene.control.DatePicker;
import javafx.scene.control.Label;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class DatePickerSample extends Application {

    private Stage stage;
    private DatePicker checkInDatePicker;
    public static void main(String[] args) {
        Locale.setDefault(Locale.US);
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        this.stage = stage;
```

```
        stage.setTitle("DatePickerSample ");
        initUI();
        stage.show();
    }

    private void initUI() {
        VBox vbox = new VBox(20);
        vbox.setStyle("-fx-padding: 10;");
        Scene scene = new Scene(vbox, 400, 400);
        stage.setScene(scene);

        checkInDatePicker = new DatePicker();

        GridPane gridPane = new GridPane();
        gridPane.setHgap(10);
        gridPane.setVgap(10);

        Label checkInlabel = new Label("Check-In Date:");
        gridPane.add(checkInlabel, 0, 0);

        GridPane.setHalignment(checkInlabel, HPos.LEFT);
        gridPane.add(checkInDatePicker, 0, 1);
        vbox.getChildren().add(gridPane);
    }
}
```

Example 26–1 implements a typical UI for selecting a check-in date for hotel booking. When you compile and run the application, the component shown in Figure 26–2 appears.

Figure 26–2 Initial View of a Date Picker Component



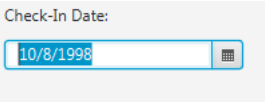
Note that in the initial state the date field is empty. However, you can change this behavior and specify the date value to be shown before any date is selected. Set the `value` property in the `DatePicker` constructor or call the `setValue` method inherited from the `ComboBox` class. Example 26–2 shows several options for setting the `LocalDate` values.

Example 26–2 Setting Date Values

```
//Setting a particular date value in the class constructor
checkInDatePicker = new DatePicker(LocalDate.of(1998, 10, 8));
//Setting a particular date value by using the setValue method
checkInDatePicker.setValue(LocalDate.of(1998, 10, 8));
//Setting the minimum date available in the calendar
checkInDatePicker.setValue(LocalDate.MIN);
//Setting the maximum date available in the calendar
checkInDatePicker.setValue(LocalDate.MAX);
//Setting the current date
checkInDatePicker.setValue(LocalDate.now());
```

Once the initial value is set, it appears in the date field after you compile and run the application. Figure 26–3 shows a specified initial date in a date field.

Figure 26–3 Date Field with the Specified Initial Date



The `DatePicker` API provides several properties and methods to alter the default appearance and behavior of a date picker. In particular, you can toggle showing of the week numbers, customize date formats, and define and apply date cell factories.

Customizing the Date Picker

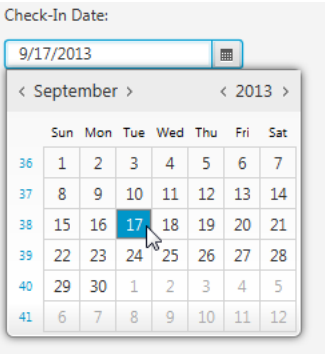
You can enable and disable showing the ISO week numbers in the calendar by using the `setShowWeekNumbers` method of the `DatePicker` class. Example 26–3 is a code line that implements this task for the checkInDatePicker.

Example 26–3 Enabling Week Numbers in the Date Picker

```
checkInDatePicker.setShowWeekNumbers(true);
```

This modification produces the date picker component with the week numbers added to the calendar element as shown in Figure 26–4.

Figure 26–4 Showing Week Numbers in the Calendar



By default, the dates in the date field are shown in the format defined by the system locale and the ISO calendar system. Thus, the selected dates in Example 26–1 are shown in the following format: mm/dd/yyyy. Normally, you do not need to modify that default format. Still, the `converter` property and the `setConverter` method of the `DatePicker` class enable you to set an alternative date format when required. Setting the `converter` value to `null` restores the default date format.

In Example 26–4, you create a converter to alter the format of the date according to the following pattern: yyyy-MM-dd. It converts the input text to an object of the `LocalDate` type when users enter a date in the date field. It also converts the `LocalDate` object that corresponds to the date selected in the calendar to the text shown in the date field.

Example 26–4 Converting a Date Format

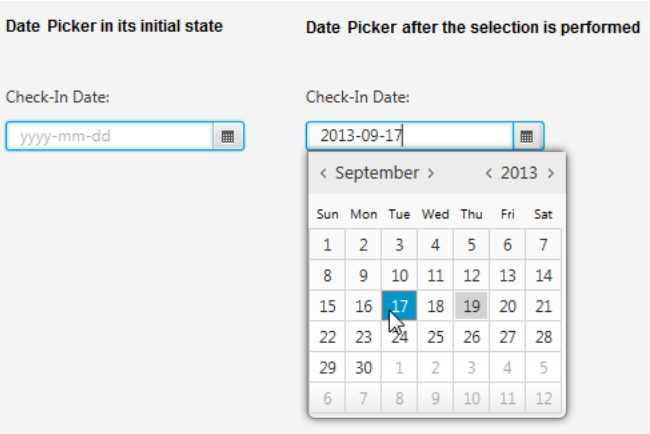
```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.Locale;
```

```
import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.scene.Scene;
import javafx.scene.control.DatePicker;
import javafx.scene.control.Label;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.util.StringConverter;

public class DatePickerSample extends Application {

    private Stage stage;
    private DatePicker checkInDatePicker;
    private final String pattern = "yyyy-MM-dd";

    public static void main(String[] args) {
        Locale.setDefault(Locale.US);
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        this.stage = stage;
        stage.setTitle("DatePickerSample ");
        initUI();
        stage.show();
    }

    private void initUI() {
        VBox vbox = new VBox(20);
        vbox.setStyle("-fx-padding: 10;");
        Scene scene = new Scene(vbox, 400, 400);
        stage.setScene(scene);
        checkInDatePicker = new DatePicker();
        StringConverter converter = new StringConverter<LocalDate>() {
            DateTimeFormatter dateFormatter =
                DateTimeFormatter.ofPattern(pattern);
            @Override
            public String toString(LocalDate date) {
                if (date != null) {
                    return dateFormatter.format(date);
                } else {
                    return "";
                }
            }
            @Override
            public LocalDate fromString(String string) {
                if (string != null && !string.isEmpty()) {
                    return LocalDate.parse(string, dateFormatter);
                } else {
                    return null;
                }
            }
        };
        checkInDatePicker.setConverter(converter);
        checkInDatePicker.setPromptText(pattern.toLowerCase());
        GridPane gridPane = new GridPane();
        gridPane.setHgap(10);
        gridPane.setVgap(10);
```

```
        Label checkInlabel = new Label("Check-In Date:");
        gridPane.add(checkInlabel, 0, 0);
        GridPane.setHalignment(checkInlabel, HPos.LEFT);
        gridPane.add(checkInDatePicker, 0, 1);
        vbox.getChildren().add(gridPane);
        checkInDatePicker.requestFocus();
    }
}
```

In addition to converting the date format, Example 26–4 sets the prompt text for the date field to inform users about the required date format. Figure 26–5 shows two states of the date picker with the date format converter applied. Both the prompt text and the selected date are shown in the new format.

Figure 26–5 Date Picker with the Format Converter and Prompt Text



You can also modify the default appearance and set a specific behavior for any cell or several cells of a calendar element. To implement this task, consider a real-life use case: selecting check-in and check-out dates during hotel booking. Example 26–5 creates a typical user interface with two date pickers.

Example 26–5 Adding Two Date Picker Components

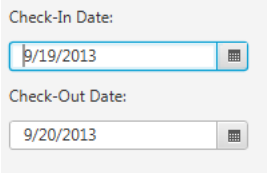
```
import java.time.LocalDate;
import java.util.Locale;
import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.scene.Scene;
import javafx.scene.control.DatePicker;
import javafx.scene.control.Label;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class DatePickerSample extends Application {

    private Stage stage;
    private DatePicker checkInDatePicker;
    private DatePicker checkOutDatePicker;

    public static void main(String[] args) {
        Locale.setDefault(Locale.US);
        launch(args);
```

```
    }

    @Override
    public void start(Stage stage) {
        this.stage = stage;
        stage.setTitle("DatePickerSample ");
        initUI();
        stage.show();
    }

    private void initUI() {
        VBox vbox = new VBox(20);
        vbox.setStyle("-fx-padding: 10;");
        Scene scene = new Scene(vbox, 400, 400);
        stage.setScene(scene);
        checkInDatePicker = new DatePicker();
        checkOutDatePicker = new DatePicker();
        checkInDatePicker.setValue(LocalDate.now());
        checkOutDatePicker.setValue(checkInDatePicker.getValue().plusDays(1));
        GridPane gridPane = new GridPane();
        gridPane.setHgap(10);
        gridPane.setVgap(10);
        Label checkInlabel = new Label("Check-In Date:");
        gridPane.add(checkInlabel, 0, 0);
        GridPane.setHalignment(checkInlabel, HPos.LEFT);
        gridPane.add(checkInDatePicker, 0, 1);
        Label checkOutlabel = new Label("Check-Out Date:");
        gridPane.add(checkOutlabel, 0, 2);
        GridPane.setHalignment(checkOutlabel, HPos.LEFT);
        gridPane.add(checkOutDatePicker, 0, 3);
        vbox.getChildren().add(gridPane);
    }
}
```

The predefined value of the checkInDatePicker in Example 26–5 is `LocalDate.now()`, which corresponds to the current date from the system clock. The predefined date of the checkOutDatePicker is the next date after the current date.

When you compile and run this example on September 19, 2013, it produces the output shown in Figure 26–6.

Figure 26–6 Two Date Picker Components



By default, all the cells in the calendar elements are available for selection. This could lead to a situation in which the check-out date precedes the check-in date, which is incorrect.

Example 26–6 creates a day cell factory for the checkOutDatePicker to disable the cell that corresponds to the date selected in the checkInDatePicker and all the cells corresponding to the preceding dates.

Example 26–6 Implementing a Day Cell Factory to Disable Some Days

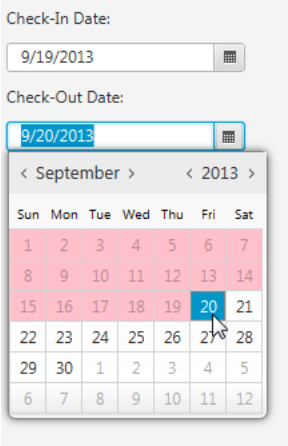
```
import java.time.LocalDate;
import java.util.Locale;
import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.scene.Scene;
import javafx.scene.control.DateCell;
import javafx.scene.control.DatePicker;
import javafx.scene.control.Label;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.util.Callback;

public class DatePickerSample extends Application {

    private Stage stage;
    private DatePicker checkInDatePicker;
    private DatePicker checkOutDatePicker;

    public static void main(String[] args) {
        Locale.setDefault(Locale.US);
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        this.stage = stage;
        stage.setTitle("DatePickerSample ");
        initUI();
        stage.show();
    }

    private void initUI() {
        VBox vbox = new VBox(20);
        vbox.setStyle("-fx-padding: 10;");
        Scene scene = new Scene(vbox, 400, 400);
        stage.setScene(scene);
        checkInDatePicker = new DatePicker();
        checkOutDatePicker = new DatePicker();
        checkInDatePicker.setValue(LocalDate.now());
        final Callback<DatePicker, DateCell> dayCellFactory =
            new Callback<DatePicker, DateCell>() {
                @Override
                public DateCell call(final DatePicker datePicker) {
                    return new DateCell() {
                        @Override
                        public void updateItem(LocalDate item, boolean empty) {
                            super.updateItem(item, empty);

                            if (item.isBefore(
                                    checkInDatePicker.getValue().plusDays(1))
                                ) {
                                    setDisable(true);
                                    setStyle("-fx-background-color: #ffc0cb;");
                            }
                        }
                    };
                }
            };
```

```
        checkOutDatePicker.setDayCellFactory(dayCellFactory);
        checkOutDatePicker.setValue(checkInDatePicker.getValue().plusDays(1));
        GridPane gridPane = new GridPane();
        gridPane.setHgap(10);
        gridPane.setVgap(10);
        Label checkInlabel = new Label("Check-In Date:");
        gridPane.add(checkInlabel, 0, 0);
        GridPane.setHalignment(checkInlabel, HPos.LEFT);
        gridPane.add(checkInDatePicker, 0, 1);
        Label checkOutlabel = new Label("Check-Out Date:");
        gridPane.add(checkOutlabel, 0, 2);
        GridPane.setHalignment(checkOutlabel, HPos.LEFT);
        gridPane.add(checkOutDatePicker, 0, 3);
        vbox.getChildren().add(gridPane);
    }
}
```

The `dayCellFactory` applied to the checkOutDatePicker calls the `setDisable` and `setStyle` methods on a `DateCell` item to protect the cells from selection and paint them with the pink color.

When you compile and run the code in Example 26–6, the DatePickerSample produces a UI with the expected behavior shown in Figure 26–7.

Figure 26–7 Disabling Cells in the Calendar Element



Now that you have learned how to create day cell factories, you can extend the default behavior of the checkOutDatePicker and provide additional functionality for the date cells that are enabled for selection.

Example 26–7 calculates the day interval between the date selected in the checkInDatePicker and the current date cell. The result is rendered in the cell tooltip.

Example 26–7 Calculating a Time Interval

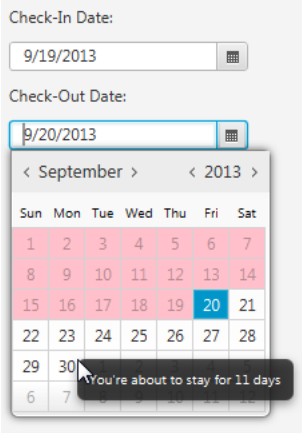
```
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.Locale;
import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.scene.Scene;
import javafx.scene.control.DateCell;
import javafx.scene.control.DatePicker;
```

```
import javafx.scene.control.Label;
import javafx.scene.control.Tooltip;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.util.Callback;

public class DatePickerSample extends Application {

    private Stage stage;
    private DatePicker checkInDatePicker;
    private DatePicker checkOutDatePicker;

    public static void main(String[] args) {
        Locale.setDefault(Locale.US);
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        this.stage = stage;
        stage.setTitle("DatePickerSample ");
        initUI();
        stage.show();
    }

    private void initUI() {
        VBox vbox = new VBox(20);
        vbox.setStyle("-fx-padding: 10;");
        Scene scene = new Scene(vbox, 400, 400);
        stage.setScene(scene);
        checkInDatePicker = new DatePicker();
        checkOutDatePicker = new DatePicker();
        checkInDatePicker.setValue(LocalDate.now());
        final Callback<DatePicker, DateCell> dayCellFactory =
            new Callback<DatePicker, DateCell>() {
                @Override
                public DateCell call(final DatePicker datePicker) {
                    return new DateCell() {
                        @Override
                        public void updateItem(LocalDate item, boolean empty) {
                            super.updateItem(item, empty);
                            if (item.isBefore(
                                    checkInDatePicker.getValue().plusDays(1))
                                ) {
                                    setDisable(true);
                                    setStyle("-fx-background-color: #ffc0cb;");
                            }
                            long p = ChronoUnit.DAYS.between(
                                    checkInDatePicker.getValue(), item
                            );
                            setTooltip(new Tooltip(
                                "You're about to stay for " + p + " days")
                            );
                        }
                    };
                }
        };
        checkOutDatePicker.setDayCellFactory(dayCellFactory);
        checkOutDatePicker.setValue(checkInDatePicker.getValue().plusDays(1));
```

```
        GridPane gridPane = new GridPane();
        gridPane.setHgap(10);
        gridPane.setVgap(10);
        Label checkInlabel = new Label("Check-In Date:");
        gridPane.add(checkInlabel, 0, 0);
        GridPane.setHalignment(checkInlabel, HPos.LEFT);
        gridPane.add(checkInDatePicker, 0, 1);
        Label checkOutlabel = new Label("Check-Out Date:");
        gridPane.add(checkOutlabel, 0, 2);
        GridPane.setHalignment(checkOutlabel, HPos.LEFT);
        gridPane.add(checkOutDatePicker, 0, 3);
        vbox.getChildren().add(gridPane);
    }
}
```

When you compile and run the modified DatePickerSample application, you can see the tooltip if you hover the mouse cursor over a date cell. Figure 26–8 captures the moment when you put a mouse cursor over the cell for September 30. The tooltip states that the time interval between September 19 and September 30 is 11 days.

Figure 26–8 Setting a Tooltip for a Date Cell



Altering the Calendar System

The Date-Time Java API introduced in JDK 8 enables developers to operate not only with ISO-based calendar systems but also with alternative chronologies, such as Japanese, Hijrah, Minguo, and Thai Buddhist.

The `DatePicker` control also supports non-ISO calendar systems by rendering the appropriate year. For the Hijrah chronology, in which month dates do not coincide with the ISO dates, it provides an additional visual theme to distinguish between ISO and Hijrah days of the month.

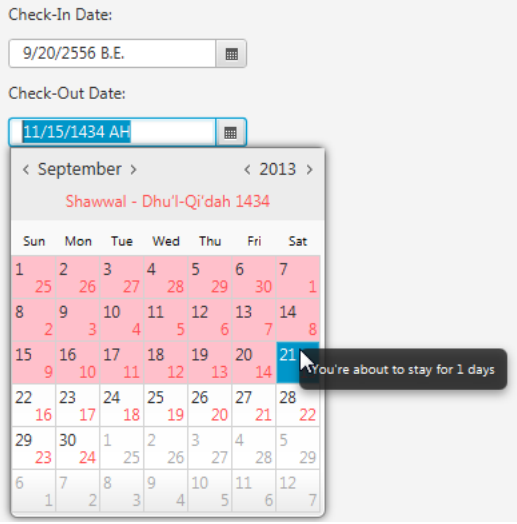
Example 26–8 applies the Thai Buddhist chronology to the checkInDatePicker and the Hijrah chronology to the checkOutDatePicker.

Example 26–8 Applying Thai Buddhist and Hijrah Chronologies

```
import java.time.chrono.*;

checkInDatePicker.setChronology(ThaiBuddhistChronology.INSTANCE);
checkOutDatePicker.setChronology(HijrahChronology.INSTANCE);
```

When these lines are added to the DatePickerSample application, the date picker controls change their appearance as shown in Figure 26–9.

Figure 26–9 Using Alternative Chronology



You can find out more details about Non-ISO Date support in the corresponding lesson of the Java Tutorials.

Related Documentation

- Date-Time Java Tutorial
- `DatePicker`
- `DateCell`

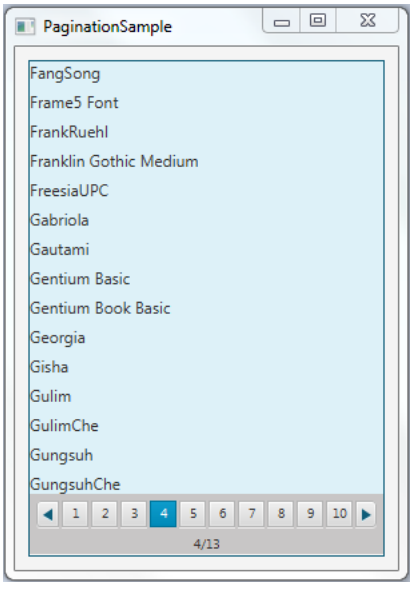
27

Pagination Control

This chapter explains how to add a pagination control to your JavaFX application. It teaches how to add a pagination control to your application, manage its page items, and style the elements of the control with CSS styles.

The pagination control that is used to navigate through multiple pages of content that are divided into smaller parts. Typical uses include navigating through email messages in a mailbox or choosing among search results. In touch devices, a pagination control can be used to browse through single pages of an article or to navigate between screens. Figure 27–1 shows a pagination control that displays the fonts available in an operating system.

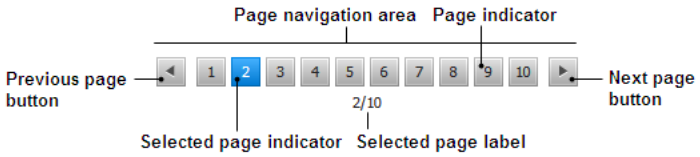
Figure 27–1 Pagination Control



Creating a Pagination Control

A pagination control consists of the page content and the page navigation areas. The Page content area renders and lays out the content according to the application logic. The Page navigation area contains a prefabricated control to preview a particular part of the content. Figure 27–2 shows the structure and basic elements of the navigation area.

Figure 27–2 Navigation Area of a Pagination Control



You can navigate through the pages by clicking a particular page indicator or by clicking the Next page and Previous page buttons. When the first page is selected, the Previous page button is disabled. Similarly, when the last navigation indicator is selected, the Next page button is disabled.

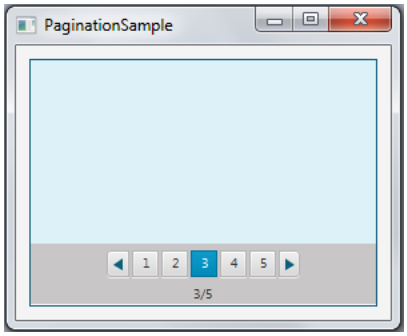
The JavaFX SDK API provides the `Pagination` class to add the pagination control to your JavaFX application. Example 27–1 shows three available constructors of the `Pagination` class.

Example 27–1 Three Constructors of the Pagination Class

```
//Creates a Pagination control with an INDETERMINATE page count
//and the current page index equal to zero
Pagination pagination1 = new Pagination();
//Creates a Pagination control with 5 pages
//and the current page index equal to zero
Pagination pagination2 = new Pagination(5);
//Creates a Pagination control with 5 pages
//and the current selected index equal to 2
Pagination pagination3 = new Pagination(5, 2);
```

The pagination3 control from Example 27–1 is displayed in Figure 27–3.

Figure 27–3 Pagination Control without the Content



Note that page indexes start with 0. Therefore, to start with the third page selected, you need to set the `currentPageIndexProperty` to 2.

The pages of the pagination3 control are empty, because no content is added to the control.

You cannot add any items directly to the pagination control, it requires setting a page factory. Use the `setPageFactory` method of the `Pagination` class to define the page content by implementing a page factory.

Implementing Page Factories

The `setPageFactory` is used to define a page factory for the pagination control. The application developer creates a callback method and sets the pagination page factory to use this callback. The callback method is called when a page has been selected. It loads and returns the content of the selected page. The `null` value must be returned if the selected page index does not exist. Example 27–2 creates a pagination control with 28 pages and populates the pages with the search results, allocating eight items per page.

Example 27–2 Adding Hyperlinks to a Pagination Control

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Pagination;
import javafx.scene.control.Hyperlink;
import javafx.scene.control.Label;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;


public class PaginationSample extends Application {
    private Pagination pagination;

    public static void main(String[] args) throws Exception {
        launch(args);
    }

    public int itemsPerPage() {
        return 8;
    }

    public VBox createPage(int pageIndex) {
        VBox box = new VBox(5);
        int page = pageIndex * itemsPerPage();
        for (int i = page; i < page + itemsPerPage(); i++) {
            VBox element = new VBox();
            Hyperlink link = new Hyperlink("Item " + (i+1));
            link.setVisited(true);
            Label text = new Label("Search results\nfor "+ link.getText());
            element.getChildren().addAll(link, text);
            box.getChildren().add(element);
        }
        return box;
    }

    @Override
    public void start(final Stage stage) throws Exception {
        pagination = new Pagination(28, 0);
        pagination.setStyle("-fx-border-color:red;");
        pagination.setPageFactory((Integer pageIndex) -> createPage(pageIndex));

        AnchorPane anchor = new AnchorPane();
        AnchorPane.setTopAnchor(pagination, 10.0);
```

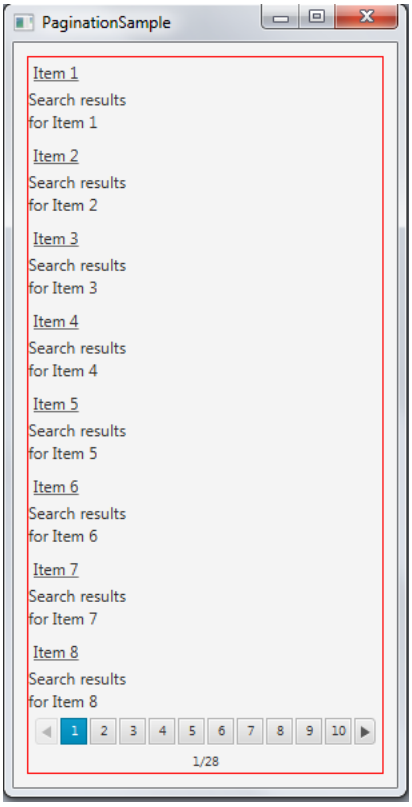
```
        AnchorPane.setRightAnchor(pagination, 10.0);
        AnchorPane.setBottomAnchor(pagination, 10.0);
        AnchorPane.setLeftAnchor(pagination, 10.0);
        anchor.getChildren().addAll(pagination);
        Scene scene = new Scene(anchor);
        stage.setScene(scene);
        stage.setTitle("PaginationSample");
        stage.show();
    }
}
```

The number of pages and the selected page are defined within the constructor of the `Pagination` class. Alternatively, you can create a `Pagination` control and set the number of pages and the index of the selected page afterward by using the `setPageCount` and `setCurrentPageIndex` methods.

The content of the `Pagination` control is declared within the `createPage` method that serves as a page factory and is called by the `setPageFactory` method. The `createPage` method creates the pairs of hyperlinks and the corresponding labels, and arranges them vertically, setting a five-pixel interval between the elements.

When you compile and run Example 27–2, you should see the output shown in Figure 27–4.

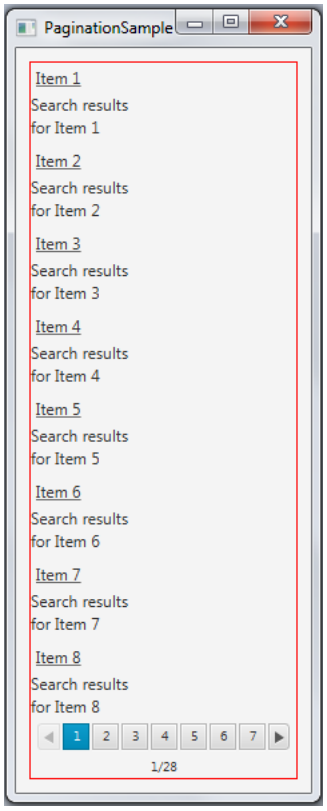
Figure 27–4 Using a Pagination Control to Preview Search Results



The current implementation of the `Pagination` control displays 10 page indicators if the number of pages exceeds 10. To set an alternative value for the displayed page indicators, use the `setMaxPageIndicatorCount` method of the `Pagination` class. For example, add the following line to Example 27–2 to show seven page indicators:

`pagination.setMaxPageIndicatorCount(7);`. Figure 27–5 shows the PaginationSample application after the change has been applied.

Figure 27–5 Changing the Number of Page Indicators



Example 27–3 shows another use for the pagination control. The application renders the text fragments, one per page. The number of the fragments is five, and the number of the declared pages of the pagination control is 28. To avoid an `ArrayIndexOutOfBoundsException` condition, add the page index check (highlighted in bold in Example 27–3) and make the callback method return `null` when the number of pages exceeds five.

Example 27–3 Adding Text Snippets to a Pagination Control

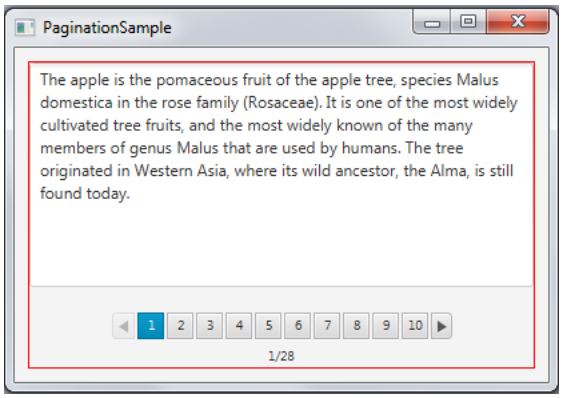
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Pagination;
import javafx.scene.control.TextArea;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class PaginationSample extends Application {

    private Pagination pagination;
    final String[] textPages = new String[]{
        "The apple is the pomaceous fruit of the apple tree, species Malus "
        + "domestica in the rose family (Rosaceae). It is one of the most "
        + "widely cultivated tree fruits, and the most widely known of "
        + "the many members of genus Malus that are used by humans. "
        + "The tree originated in Western Asia, where its wild ancestor, "
```

```
        + "the Alma, is still found today.",
        "The hawthorn is a large genus of shrubs and trees in the rose family,"
        + "Rosaceae, native to temperate regions of the Northern Hemisphere "
        + "in Europe, Asia and North America. The name hawthorn was "
        + "originally applied to the species native to northern Europe, "
        + "especially the Common Hawthorn C. monogyna, and the unmodified "
        + "name is often so used in Britain and Ireland.",
        "The ivy is a flowering plant in the grape family (Vitaceae) native to "
        + " eastern Asia in Japan, Korea, and northern and eastern China. "
        + "It is a deciduous woody vine growing to 30 m tall or more given "
        + "suitable support,  attaching itself by means of numerous small "
        + "branched tendrils tipped with sticky disks.",
        "The quince is the sole member of the genus Cydonia and is native to "
        + "warm-temperate southwest Asia in the Caucasus region. The "
        + "immature fruit is green with dense grey-white pubescence, most "
        + "of which rubs off before maturity in late autumn when the fruit "
        + "changes color to yellow with hard, strongly perfumed flesh.",
        "Aster (syn. Diplopappus Cass.) is a genus of flowering plants "
        + "in the family Asteraceae. The genus once contained nearly 600 "
        + "species in Eurasia and North America, but after morphologic "
        + "and molecular research on the genus during the 1990s, it was "
        + "decided that the North American species are better treated in a "
        + "series of other related genera. After this split there are "
        + "roughly 180 species within the genus, all but one being confined "
        + "to Eurasia."
    };

    public static void main(String[] args) throws Exception {
        launch(args);
    }

    public int itemsPerPage() {
        return 1;
    }

    public VBox createPage(int pageIndex) {
        VBox box = new VBox(5);
        int page = pageIndex * itemsPerPage();
        for (int i = page; i < page + itemsPerPage(); i++) {
            TextArea text = new TextArea(textPages[i]);
            text.setWrapText(true);
            box.getChildren().add(text);
        }
        return box;
    }

    @Override
    public void start(final Stage stage) throws Exception {
        pagination = new Pagination(28, 0);
        pagination.setStyle("-fx-border-color:red;");
        pagination.setPageFactory((Integer pageIndex) -> {
            if (pageIndex >= textPages.length) {
                return null;
            } else {
                return createPage(pageIndex);
            }
        });

        AnchorPane anchor = new AnchorPane();
        AnchorPane.setTopAnchor(pagination, 10.0);
```

```
        AnchorPane.setRightAnchor(pagination, 10.0);
        AnchorPane.setBottomAnchor(pagination, 10.0);
        AnchorPane.setLeftAnchor(pagination, 10.0);
        anchor.getChildren().addAll(pagination);
        Scene scene = new Scene(anchor, 400, 250);
        stage.setScene(scene);
        stage.setTitle("PaginationSample");
        stage.show();
    }
}
```

When you compile and run Example 27–3, you will see the output shown in Figure 27–6.

Figure 27–6 Rendering Text Fragments in a Pagination Control



In some cases you cannot set the exact number of items to render and, therefore, the number of pages in a pagination control. In such situations, you can include a line of code that calculates the number of pages within the constructor of the `Pagination` object. Example 27–4 outputs a list of system fonts and calculates the number of pages as the length of the fonts array divided by the number of items per page.

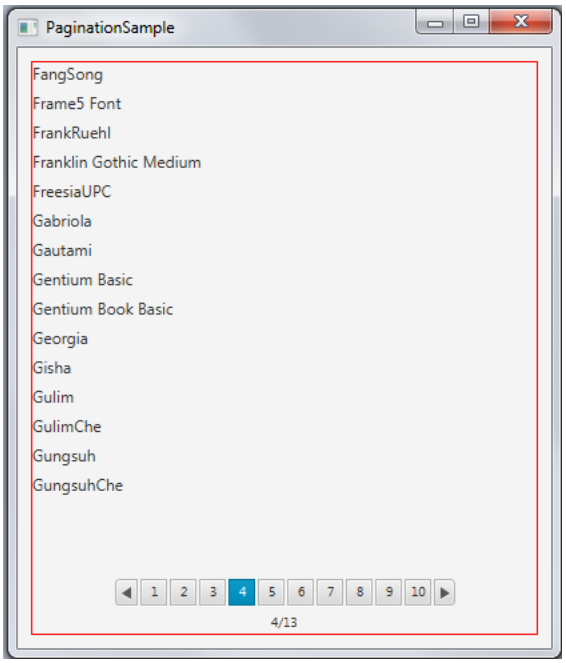
Example 27–4 Adding Content of an Undetermined Size

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Pagination;
import javafx.scene.control.Label;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class Test8 extends Application {

    private Pagination pagination;
    String[] fonts = new String[]{};

    public static void main(String[] args) throws Exception {
        launch(args);
    }

    public int itemsPerPage() {
        return 15;
```

```
    }

    public VBox createPage(int pageIndex) {
        VBox box = new VBox(5);
        int page = pageIndex * itemsPerPage();
        for (int i = page; i < page + itemsPerPage(); i++) {
            Label font = new Label(fonts[i]);
            box.getChildren().add(font);
        }
        return box;
    }

    @Override
    public void start(final Stage stage) throws Exception {
        fonts = Font.getFamilies().toArray(fonts);

        pagination = new Pagination(fonts.length/itemsPerPage(), 0);
        pagination.setStyle("-fx-border-color:red;");
        pagination.setPageFactory((Integer pageIndex) -> createPage(pageIndex));

        AnchorPane anchor = new AnchorPane();
        AnchorPane.setTopAnchor(pagination, 10.0);
        AnchorPane.setRightAnchor(pagination, 10.0);
        AnchorPane.setBottomAnchor(pagination, 10.0);
        AnchorPane.setLeftAnchor(pagination, 10.0);
        anchor.getChildren().addAll(pagination);
        Scene scene = new Scene(anchor, 400, 450);
        stage.setScene(scene);
        stage.setTitle("PaginationSample");
        stage.show();
    }
}
```

When you compile and run this example, it produces the application window shown in Figure 27–7.

Figure 27–7 Using a Pagination Control to Render the System Fonts



Styling a Pagination Control

You can customize the pagination control to display bullet page indicators instead of numeric page indicators by setting the style class `STYLE_CLASS_BULLET`. In addition, you can modify the default pagination styles in the `modena` style sheet.

Example 27–5 shows some alternative styles for the visual elements of the pagination control in Example 27–4.

Example 27–5 Modified Styles of the Pagination Control

```
.pagination {
    -fx-border-color:  #0E5D79;
}

.pagination .page {
    -fx-background-color: #DDF1F8;
}

.pagination .pagination-control {
    -fx-background-color: #C8C6C6;
}

.pagination .pagination-control .bullet-button {
    -fx-background-color: transparent, #DDF1F8, #0E5D79, white, white;
}

.pagination .pagination-control .bullet-button:selected {
    -fx-background-color: transparent, #DDF1F8, #0E5D79, white, #0E5D79;
}

.pagination .pagination-control .left-arrow, .right-arrow{
    -fx-background-color: #DDF1F8, #0E5D79;
}
```

Example 27–6 applies these styles to the pagination control and sets the bullet style for the page indicators.

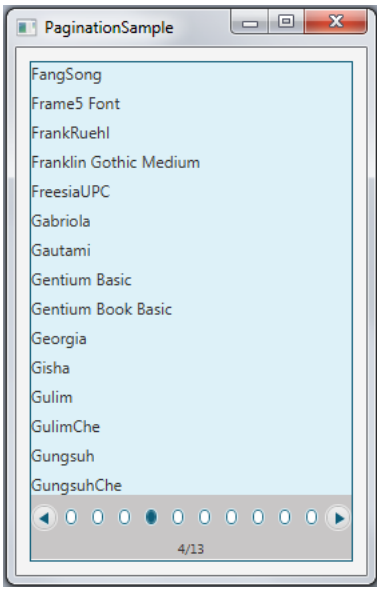
Example 27–6 Enabling the Modified Pagination Control Style in the PaginationSample Application

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Pagination;
import javafx.scene.Node;
import javafx.scene.control.Label;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;
import javafx.util.Callback;

public class PaginationSample extends Application {

    private Pagination pagination;
    String[] fonts = new String[]{};

    public static void main(String[] args) throws Exception {
        launch(args);
    }

    public int itemsPerPage() {
        return 15;
    }

    public VBox createPage(int pageIndex) {
        VBox box = new VBox(5);
        int page = pageIndex * itemsPerPage();
        for (int i = page; i < page + itemsPerPage(); i++) {
            Label font = new Label(fonts[i]);
            box.getChildren().add(font);
        }
        return box;
    }

    @Override
    public void start(final Stage stage) throws Exception {
        fonts = Font.getFamilies().toArray(fonts);

        pagination = new Pagination(fonts.length/itemsPerPage(), 0);
        pagination.getStyleClass().add(Pagination.STYLE_CLASS_BULLET);
        pagination.setPageFactory((Integer pageIndex) -> createPage(pageIndex));

        AnchorPane anchor = new AnchorPane();
        AnchorPane.setTopAnchor(pagination, 10.0);
        AnchorPane.setRightAnchor(pagination, 10.0);
        AnchorPane.setBottomAnchor(pagination, 10.0);
        AnchorPane.setLeftAnchor(pagination, 10.0);
        anchor.getChildren().addAll(pagination);
        Scene scene = new Scene(anchor, 400, 450);
        stage.setScene(scene);
        stage.setTitle("PaginationSample");
        scene.getStylesheets().add("paginationsample/ControlStyle.css");
        stage.show();
```

```
    }
}
```

When you apply the newly defined styles to the PaginationSample application, and compile and run it, you see the application window shown in Figure 27–8.

Figure 27–8 PaginationSample with Bullet Page Indicators and the New CSS Styles Applied



In addition to the applied styles you can consider the following styles to alter the appearance of the pagination control in your applications:

- `-fx-max-page-indicator-count` — Sets the maximum number of page indicators.
- `-fx-arrows-visible` — Toggles visibility of the Next and Previous button arrows, `true` by default.
- `-fx-tooltip-visible` — Toggles visibility of the page indicator tooltip, `true` by default.
- `-fx-page-information-visible` — Toggles visibility of the page information, `true` by default.
- `-fx-page-information-alignment` — Sets the alignment of the page information.

Related API Documentation

- `Pagination`
- `VBox`
- `AnchorPane`

28

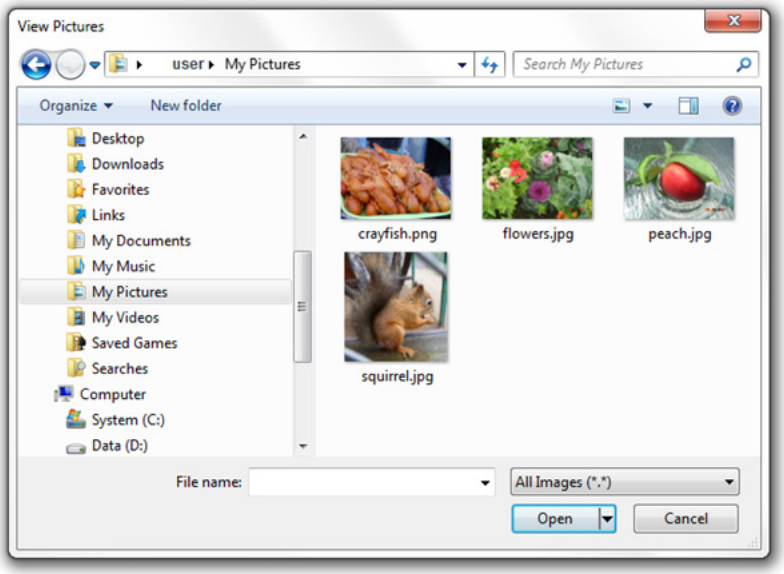
File Chooser

This chapter explains how to use the `FileChooser` class to enable users to navigate the file system. The samples provided in this chapter explain how to open one or several files, configure a file chooser dialog window, and save the application content.

Unlike other user interface component classes, the `FileChooser` class does not belong to the `javafx.scene.controls` package. However, this class deserves to be mentioned in the JavaFX UI Controls tutorial, because it supports one of the typical GUI application functions: file system navigation.

The `FileChooser` class is located in the `javafx.stage` package along with the other basic root graphical elements, such as `Stage`, `Window`, and `Popup`. The View Pictures window in Figure 28–1 is an example of the file chooser dialog in Windows.

Figure 28–1 Example of a File Chooser Window



Opening Files

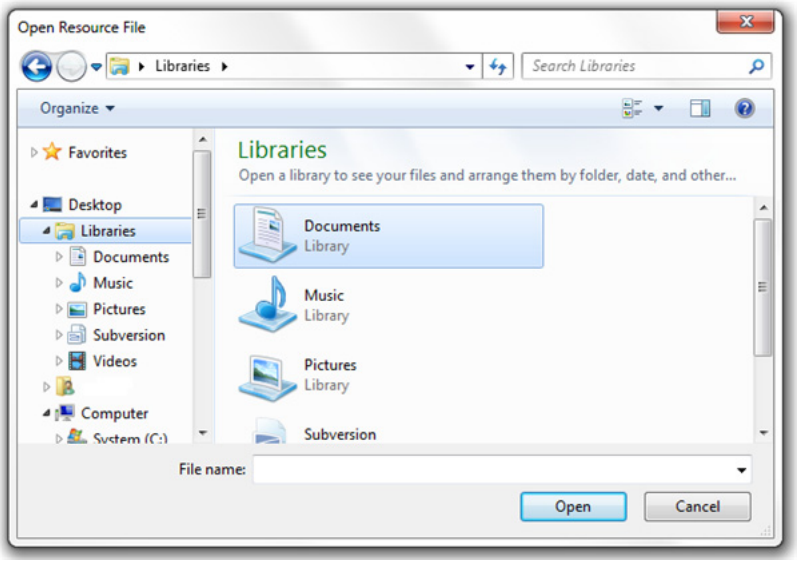
A file chooser can be used to invoke an open dialog window for selecting either a single file or multiple files, and to enable a file save dialog window. To display a file chooser, you typically use the `FileChooser` class. Example 28–1 provides the simplest way to enable a file chooser in your application.

Example 28–1 Showing a File Chooser

```
FileChooser fileChooser = new FileChooser();
fileChooser.setTitle("Open Resource File");
fileChooser.showOpenDialog(stage);
```

After the code from Example 28–1 is added to a JavaFX application, the file chooser dialog window appears immediately when the application starts, as shown in Figure 28–2.

Figure 28–2 Simple File Chooser



Note: Figure 28–2 shows the file chooser in Windows. When you open file choosers in other operating systems that support this functionality, you receive alternative windows. Figure 28–3 and Figure 28–4 show examples of file chooser windows in Linux and Mac OS.

Figure 28–3 File Chooser Window in Linux

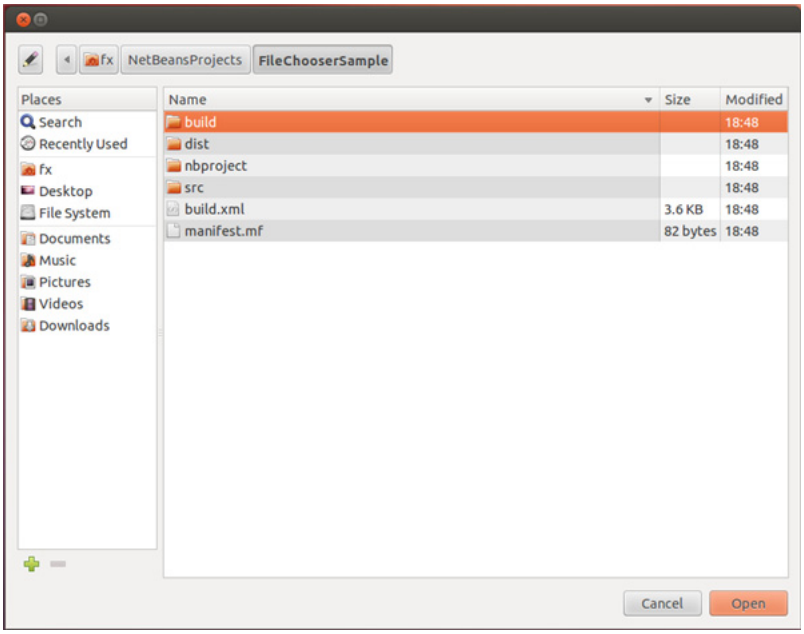
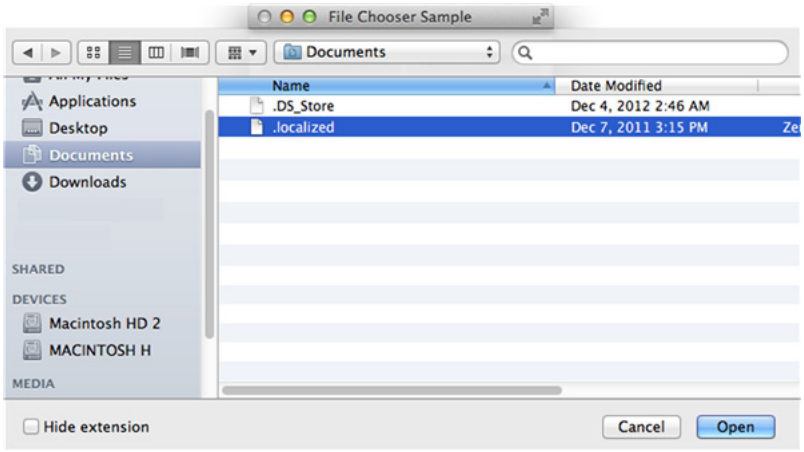


Figure 28–4 File Chooser Window in Mac OS



Although in the previous example the file chooser appears automatically when the application starts, a more typical approach is to invoke a file chooser by selecting the corresponding menu item or clicking a dedicated button. In this tutorial, you create an application that enables a user to click a button and open a one or more pictures located in the file system. Example 28–2 shows the code of the FileChooserSample application that implements this task.

Example 28–2 Opening File Chooser for Single and Multiple Selection

```
import java.awt.Desktop;
import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

public final class FileChooserSample extends Application {

    private final Desktop desktop = Desktop.getDesktop();

    @Override
    public void start(final Stage stage) {
        stage.setTitle("File Chooser Sample");

        final FileChooser fileChooser = new FileChooser();

        final Button openButton = new Button("Open a Picture...");
        final Button openMultipleButton = new Button("Open Pictures...");

        openButton.setOnAction(
            (final ActionEvent e) -> {
                File file = fileChooser.showOpenDialog(stage);
                if (file != null) {
                    openFile(file);
                }
        });
        openMultipleButton.setOnAction(
            (final ActionEvent e) -> {
                List<File> list =
                    fileChooser.showOpenMultipleDialog(stage);
                if (list != null) {
                    list.stream().forEach((file) -> {
                        openFile(file);
                    });
                }
        });

        final GridPane inputGridPane = new GridPane();

        GridPane.setConstraints(openButton, 0, 0);
        GridPane.setConstraints(openMultipleButton, 1, 0);
        inputGridPane.setHgap(6);
        inputGridPane.setVgap(6);
        inputGridPane.getChildren().addAll(openButton, openMultipleButton);

        final Pane rootGroup = new VBox(12);
        rootGroup.getChildren().addAll(inputGridPane);
        rootGroup.setPadding(new Insets(12, 12, 12, 12));

        stage.setScene(new Scene(rootGroup));
        stage.show();
    }

    public static void main(String[] args) {
        Application.launch(args);
```

```
    }

    private void openFile(File file) {
        EventQueue.invokeLater(() -> {
            try {
                desktop.open(file);
            } catch (IOException ex) {
                Logger.getLogger(FileChooserSample.
                        class.getName()).
                        log(Level.SEVERE, null, ex);
            }
        });
    }
}
```

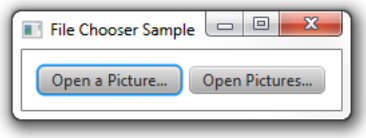
In Example 28–2, the Open a Picture button enables the user to open a file chooser for a single selection, and the Open Pictures button enables the user to open a file chooser for multiple selections. The `setOnAction` methods for these buttons are almost identical. The only difference is in the method that is used to invoke a `FileChooser`.

- The `showOpenDialog` method shows a new file open dialog in which one file can be selected. The method returns the value that specifies the file chosen by the user or `null` if no selection has been made.
- The `showOpenMultipleDialog` method shows a new file open dialog in which multiple files can be selected. The method returns the value that specifies the list of files chosen by the user or `null` if no selection has been made. The returned list cannot be modified and throws `UnsupportedOperationException` on each modification attempt.

Both methods do not return results until the displayed open dialog window is dismissed (in other words, until a user commits or cancels the selection).

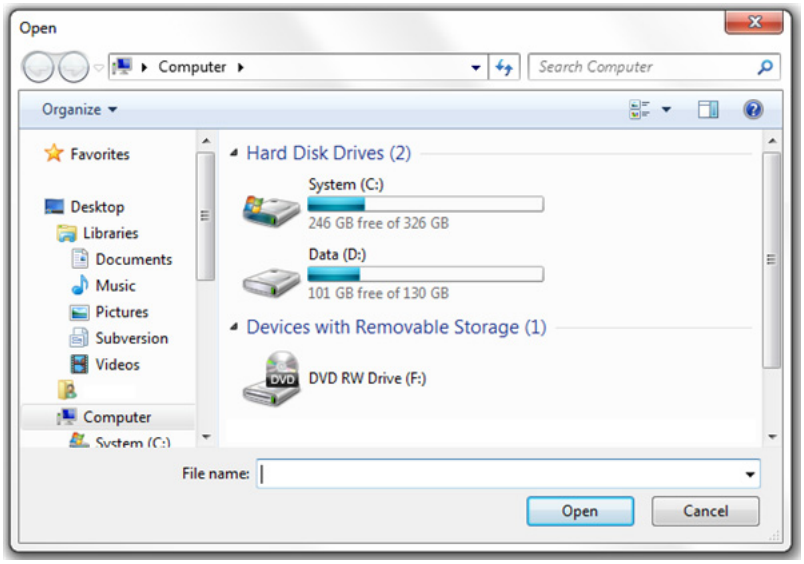
When you compile and run the FileChooserSample application, it produces the window shown in Figure 28–5.

Figure 28–5 FileChooserSample with Two Buttons



When you click either of the buttons, the dialog window shown in Figure 28–6 appears. The opened file chooser dialog window shows the location that is the default for your operating system.

Figure 28–6 Default File Chooser Window



Users of the FileChooserSample application may navigate to a directory containing pictures and select a picture. After a file is selected, it is opened with the associated application. The example code implements this by using the `open` method of the `java.awt.Desktop` class: `desktop.open(file);`.

Note: Availability of the `Desktop` class is platform dependent. Refer to API documentation for more information on the `Desktop` class. You can also use the `isDesktopSupported()` method to check if it is supported in your system.

You can improve the user experience for this application by setting the file chooser directory to the specific directory that contains the pictures.

Configuring a File Chooser

You can configure the file chooser dialog window by setting the `initialDirectory` and `title` properties of a `FileChooser` object. Example 28–3 shows how to specify the initial directory and a suitable title to preview and open pictures.

Example 28–3 Setting the Initial Directory and Window Title

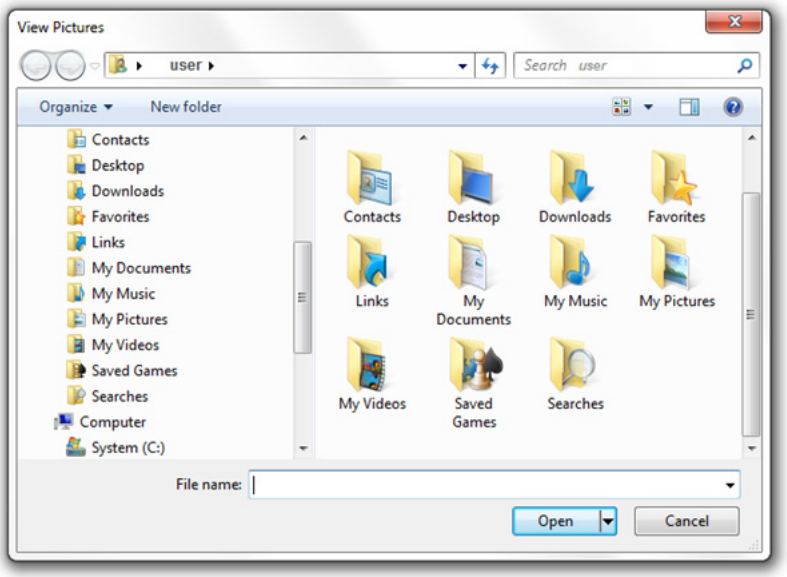
```
import java.awt.Desktop;
import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
```

```
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

public final class FileChooserSample extends Application {

    private final Desktop desktop = Desktop.getDesktop();

    @Override
    public void start(final Stage stage) {
        stage.setTitle("File Chooser Sample");

        final FileChooser fileChooser = new FileChooser();

        final Button openButton = new Button("Open a Picture...");
        final Button openMultipleButton = new Button("Open Pictures...");

        openButton.setOnAction(
            (final ActionEvent e) -> {
                configureFileChooser(fileChooser);
                File file = fileChooser.showOpenDialog(stage);
                if (file != null) {
                    openFile(file);
                }
        });

        openMultipleButton.setOnAction(
            (final ActionEvent e) -> {
                configureFileChooser(fileChooser);
                List<File> list =
                        fileChooser.showOpenMultipleDialog(stage);
                if (list != null) {
                    list.stream().forEach((file) -> {
                        openFile(file);
                    });
                }
        });

        final GridPane inputGridPane = new GridPane();

        GridPane.setConstraints(openButton, 0, 0);
        GridPane.setConstraints(openMultipleButton, 1, 0);
        inputGridPane.setHgap(6);
        inputGridPane.setVgap(6);
        inputGridPane.getChildren().addAll(openButton, openMultipleButton);

        final Pane rootGroup = new VBox(12);
        rootGroup.getChildren().addAll(inputGridPane);
        rootGroup.setPadding(new Insets(12, 12, 12, 12));

        stage.setScene(new Scene(rootGroup));
        stage.show();
    }

    public static void main(String[] args) {
        Application.launch(args);
    }

  private static void configureFileChooser(final FileChooser fileChooser){
```

```
            fileChooser.setTitle("View Pictures");
            fileChooser.setInitialDirectory(
                new File(System.getProperty("user.home"))
            );
        }

        private void openFile(File file) {
            EventQueue.invokeLater(() -> {
                try {
                    desktop.open(file);
                } catch (IOException ex) {
                    Logger.getLogger(FileChooserSample.
                            class.getName()).
                            log(Level.SEVERE, null, ex);
                }
            });
        }
    }
```

The `configureFileChooser` method sets the View Pictures title and the path to the user home directory with the My Pictures sub-directory. When you compile and run the FileChooserSample and click one of the buttons, the file chooser shown in Figure 28–7 appears.

Figure 28–7 Opening a Pictures Library



You can also let the users specify the target directory by using the `DirectoryChooser` class. In the code fragment shown in Example 28–4, the click of the `browseButton` invokes the `directoryChooser.showDialog` method.

Example 28–4 Use of the DirectoryChooser Class

```
final Button browseButton = new Button("...");
browseButton.setOnAction(
    (final ActionEvent e) -> {
        final DirectoryChooser directoryChooser
            new DirectoryChooser();
```

```
        final File selectedDirectory =
                directoryChooser.showDialog(stage);
        if (selectedDirectory != null) {
            selectedDirectory.getAbsolutePath();
        }
});
```

After the selection is performed, you can assign the corresponding value to the file chooser as follows: `fileChooser.setInitialDirectory(selectedDirectory); .`

Setting Extension Filters

As the next configuration option, you can set the extension filter to determine which files to open in a file chooser, as shown in Example 28–5.

Example 28–5 Setting an Image Type Filter

```
import java.awt.Desktop;
import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

public final class FileChooserSample extends Application {

    private final Desktop desktop = Desktop.getDesktop();

    @Override
    public void start(final Stage stage) {
        stage.setTitle("File Chooser Sample");

        final FileChooser fileChooser = new FileChooser();
        final Button openButton = new Button("Open a Picture...");
        final Button openMultipleButton = new Button("Open Pictures...");

        openButton.setOnAction(
            (final ActionEvent e) -> {
                configureFileChooser(fileChooser);
                File file = fileChooser.showOpenDialog(stage);
                if (file != null) {
                    openFile(file);
                }
        });

        openMultipleButton.setOnAction(
            (final ActionEvent e) -> {
                configureFileChooser(fileChooser);
                List<File> list =
```

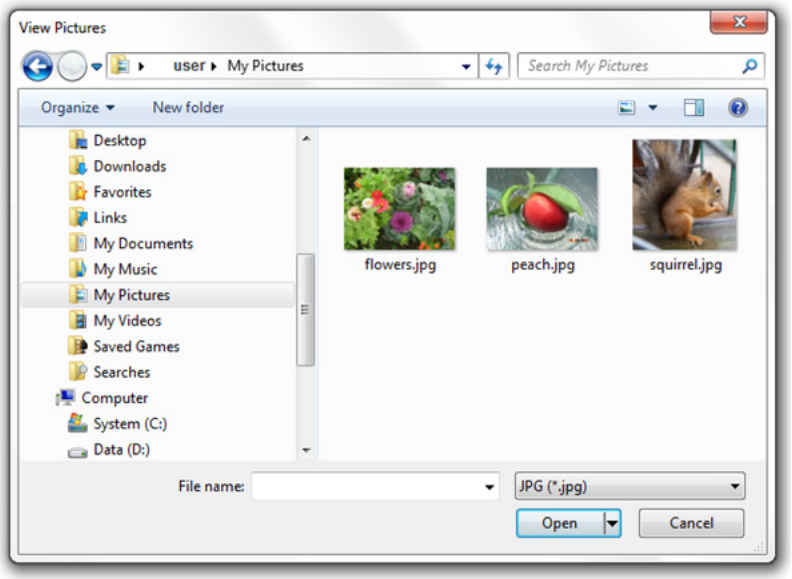
```
                        fileChooser.showOpenMultipleDialog(stage);
                if (list != null) {
                    list.stream().forEach((file) -> {
                        openFile(file);
                    });
                }
            });

            final GridPane inputGridPane = new GridPane();

            GridPane.setConstraints(openButton, 0, 1);
            GridPane.setConstraints(openMultipleButton, 1, 1);
            inputGridPane.setHgap(6);
            inputGridPane.setVgap(6);
            inputGridPane.getChildren().addAll(openButton, openMultipleButton);

            final Pane rootGroup = new VBox(12);
            rootGroup.getChildren().addAll(inputGridPane);
            rootGroup.setPadding(new Insets(12, 12, 12, 12));

            stage.setScene(new Scene(rootGroup));
            stage.show();
        }

        public static void main(String[] args) {
            Application.launch(args);
        }

        private static void configureFileChooser(
            final FileChooser fileChooser) {
                fileChooser.setTitle("View Pictures");
                fileChooser.setInitialDirectory(
                    new File(System.getProperty("user.home"))
                );
                fileChooser.getExtensionFilters().addAll(
                    new FileChooser.ExtensionFilter("All Images", "*.*"),
                    new FileChooser.ExtensionFilter("JPG", "*.jpg"),
                    new FileChooser.ExtensionFilter("PNG", "*.png")
                );
        }

        private void openFile(File file) {
            EventQueue.invokeLater(() -> {
                try {
                    desktop.open(file);
                } catch (IOException ex) {
                    Logger.getLogger(FileChooserSample.
                            class.getName()).
                            log(Level.SEVERE, null, ex);
                }
            });
        }
    }
```

In Example 28–5, you set an extension filter by using `FileChooser.ExtensionFilter` to define the following options for file selection: All images, JPG, and PNG.

When you compile, run the FileChooserSample code from Example 28–5, and click one of the buttons, the extension filters appear in the file chooser window. If a user selects JPG, then the file chooser displays only pictures of the JPG type. Figure 28–8 captures

the moment of selection JPG images in the My Pictures directory.

Figure 28–8 Filtering JPG Files in File Chooser



Saving Files

In addition to opening and filtering files, the `FileChooser` API provides a capability to let a user specify a file name (and its location in the file system) for a file to be saved by the application. The `showSaveDialog` method of the `FileChooser` class opens a save dialog window. Like other show dialog methods, the `showSaveDialog` method returns the file chosen by the user or `null` if no selection has been performed.

The code fragment shown in Example 28–6 is an addition to the Menu sample. It implements one more item of the context menu that saves the displayed image in the file system.

Example 28–6 Saving an Image with the FileChooser Class

```
MenuItem cmItem2 = new MenuItem("Save Image");
cmItem2.setOnAction((ActionEvent e) -> {
    FileChooser fileChooser1 = new FileChooser();
    fileChooser1.setTitle("Save Image");
    System.out.println(pic.getId());
    File file = fileChooser1.showSaveDialog(stage);
    if (file != null) {
        try {
            ImageIO.write(SwingFXUtils.fromFXImage(pic.getImage(),
                    null), "png", file);
        } catch (IOException ex) {
             System.out.println(ex.getMessage());
        }
    }
});
```

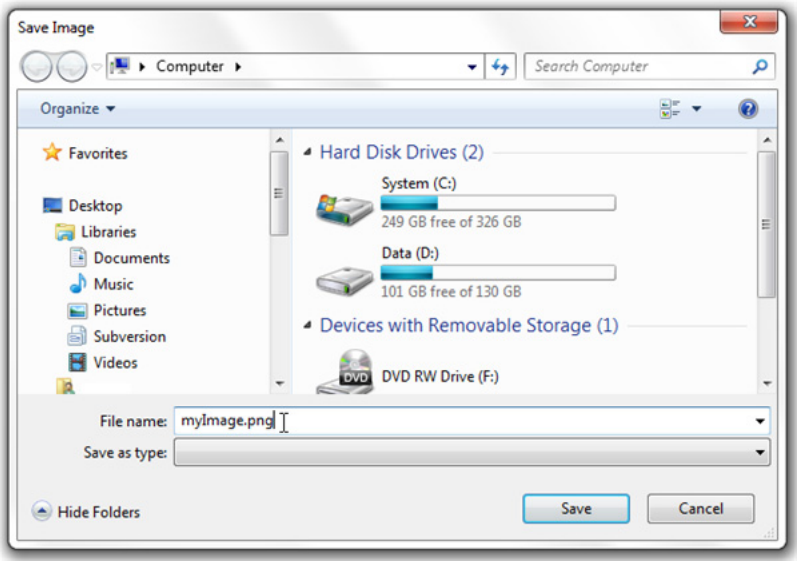
When you add Example 28–6 to the MenuSample application (find the source code in the Application Files), compile and run it, you enable the Save Image menu item, as

shown in Figure 28–9.

Figure 28–9 Saving Image

After a user selects the Save Image item, the Save Image window shown in Figure 28–10 appears.

Figure 28–10 The Save Image Window



The Save Image window corresponds to the typical user experience for the save dialog windows: the user needs to select the target directory, type in the name of the saving file, and click Save.

Related API Documentation

- `FileChooser`

- `DirectoryChooser`

29

Customization of UI Controls

This chapter describes the aspects of UI control customization and summarizes some tips and tricks provided by Oracle to help you modify the appearance and behavior of UI controls.

You can learn how to customize the controls from the sample applications in the UI Control Samples project by applying Cascading Style Sheets (CSS), redefining the default behavior, and using cell factories. For more specific cases, when the task of your application requires unique features that cannot be implemented with the classes of the `javafx.scene.control` package, extend the `Control` class to invent your own control.

Applying CSS

You can change the look of UI controls by redefining the style definitions of the JavaFX `modena` style sheets. Skinning JavaFX Applications with CSS explains the general concepts and approaches to modifying the styles and enabling them in a JavaFX application.

Consider some of the specific tasks that are frequently requested by developers at the JavaFX forum.

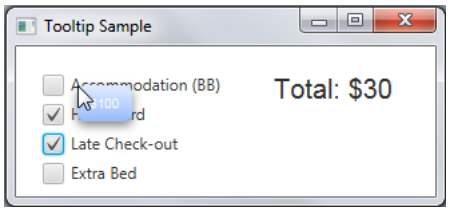
Although the `Tooltip` class does not have any properties or methods to change the default color of the tooltip, you can modify the `-fx-background-color` property of the `.tooltip` CSS class as shown in Example 29–1.

Example 29–1 Changing the Background Color of a Tooltip

```
.tooltip {
    -fx-background-color: linear-gradient(#e2ecfe, #99bcfd);
}
.page-corner {
    -fx-background-color: linear-gradient(from 0% 0% to 50% 50%,#3278fa,#99bcfd);
}
```

The `.page-corner` CSS class defines the color of the right-bottom corner of the tooltip. When you add the code in Example 29–1 to the style sheets of the TooltipSample and apply the style sheets to the scene, the tooltip changes its color to blue. See Figure 29–1 to evaluate the effect.

Figure 29–1 Tooltip with the Blue Background Color



Note that when you modify the default style of a tooltip, the new look is applied to all the tooltips in your application.

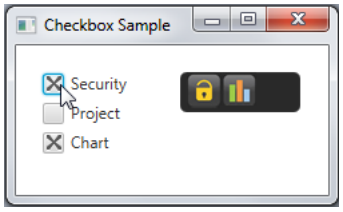
Another popular design task is changing the default marks for the controls. For example, the default style of the `CheckBox` class defines the traditional check mark for the selected state. You can redefine the shape of the mark as well as its color as shown in Example 29–2.

Example 29–2 Alternative Mark for a Checkbox

```
.check-box .mark {
    -fx-shape:
"M2,0L5,4L8,0L10,0L10,2L6,5L10,8L10,10L8,10L5,6L2,10L0,10L0,8L4,5L0,2L0,0Z";
}
.check-box:selected .mark {
    -fx-background-color: #0181e2;
}
```

The `-fx-shape` property sets the new SVG path for the mark, and the `-fx-background-color` property defines its color. When the modified style sheets are enabled in the CheckBoxSample application, the selected checkboxes contain X marks instead of check marks, as shown in Figure 29–2.

Figure 29–2 ComboBoxSample with the Modified Checkbox Style



Many developers asked how to overcome the limitation in visual style of the `TableView` and `ListView` controls. By default, all rows in these controls are shown, whether they are empty or not. With the proper CSS settings, you can set a specific color for all empty rows. Example 29–3 implements this task for a `TableView` control.

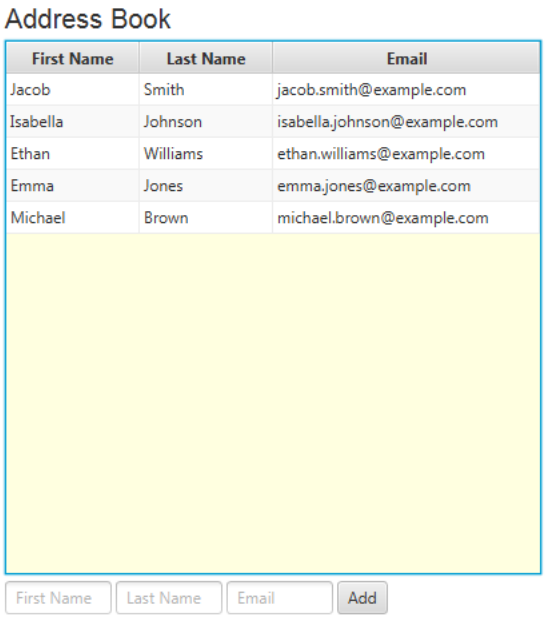
Example 29–3 Setting Color for Empty Rows in a Table View

```
.table-row-cell:empty {
    -fx-background-color: lightyellow;
}

.table-row-cell:empty .table-cell {
    -fx-border-width: 0px;
}
```

The first CSS style determines that all empty rows, regardless of whether they are even or odd, should have light yellow backgrounds. When the table-row-cell is empty, the second CSS statement removes the vertical border that is painted on the right-hand side of all table cells.

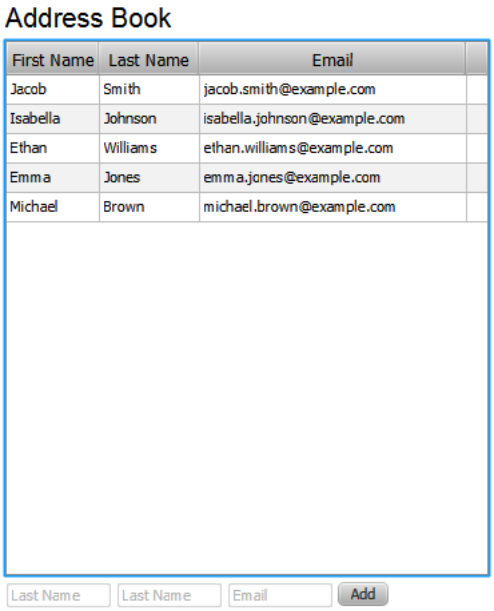
When the CSS styles from Example 29–3 are enabled in the TableViewSample application, the Address Book table looks as shown Figure 29–3.

Figure 29–3 TableViewSample with Color Added to the Empty Rows



You can even set the `null` value for the background color of the empty cells. The style sheets will use the default background color of the table view in this case. See Figure 29–4 to evaluate the effect.

Figure 29–4 TableViewSample with Null Background Color Added to the Empty Rows



You can set more CSS properties for UI Controls to alter their shapes, color schemes, and the applied effects. See the JavaFX CSS Reference Guide for more information about available CSS properties and classes.

Altering Default Behavior

Many developers requested a specific API to restrict input in the text field, for example, to allow only number values. Example 29–4 provides a simple application with a numeric text field.

Example 29–4 Prohibiting Letters in the Text Field

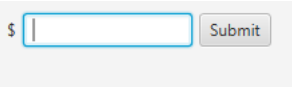
```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class CustomTextFieldSample extends Application {

    final static Label label = new Label();

    @Override
    public void start(Stage stage) {
        Group root = new Group();
        Scene scene = new Scene(root, 300, 150);
        stage.setScene(scene);
        stage.setTitle("Text Field Sample");

        GridPane grid = new GridPane();
        grid.setPadding(new Insets(10, 10, 10, 10));
```

```
        grid.setVgap(5);
        grid.setHgap(5);

        scene.setRoot(grid);
        final Label dollar = new Label("$");
        GridPane.setConstraints(dollar, 0, 0);
        grid.getChildren().add(dollar);

        final TextField sum = new TextField() {
            @Override
            public void replaceText(int start, int end, String text) {
                if (!text.matches("[a-z, A-Z]")) {
                    super.replaceText(start, end, text);
                }
                label.setText("Enter a numeric value");
            }

            @Override
            public void replaceSelection(String text) {
                if (!text.matches("[a-z, A-Z]")) {
                    super.replaceSelection(text);
                }
            }
        };

        sum.setPrefColumnCount(10);
        GridPane.setConstraints(sum, 1, 0);
        grid.getChildren().add(sum);

        Button submit = new Button("Submit");
        GridPane.setConstraints(submit, 2, 0);
        grid.getChildren().add(submit);

        submit.setOnAction((ActionEvent e) -> {
            label.setText(null);
        });

        GridPane.setConstraints(label, 0, 1);
        GridPane.setColumnSpan(label, 3);
        grid.getChildren().add(label);

        scene.setRoot(grid);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

To redefine the default implementation of the `TextField` class, you must override the `replaceText` and `replaceSelection` methods inherited from the `TextInputControl` class.

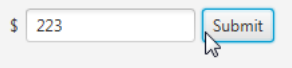
When the user tries to enter any letter in the Sum text field, no symbols appear, and the warning message is shown. Figure 29–5 illustrates this situation.

Figure 29–5 Attempt to Enter Letter Symbols



However, when the user attempts to enter the numeric values, they appear in the field as shown in Figure 29–6.

Figure 29–6 Entering Numeric Values



Implementing Cell Factories

Appearance and even behavior of four UI controls can be entirely customized by using the mechanism of cell factories. You can apply cell factories to `TableView`, `ListView`, `TreeView`, and `ComboBox`. A cell factory is used to generate cell instances, which are used to represent any single item of these controls.

The `Cell` class extends the `Labeled` class, which provides all the required properties and methods to implement the most typical use case — showing and editing text. However, when the task of your application requires showing graphical objects in the lists or tables, you can use the `graphic` property and place any `Node` in the cell (see the Cell class API specification for more information about custom cells).

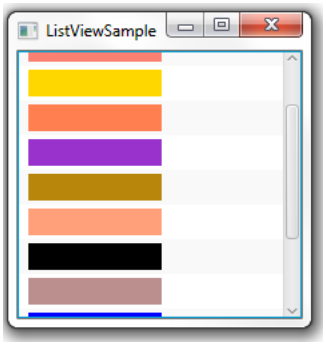
For instance, the code fragments in Example 29–5 create a cell factory for the list view and redefine the content of the cells within the `updateItem` method, so that the list shows rectangles of different colors.

Example 29–5 Implementing Cell Factories for the ListView Control

```
list.setCellFactory((ListView<String> l) -> new ColorRectCell());
...
static class ColorRectCell extends ListCell<String> {
    @Override
    public void updateItem(String item, boolean empty) {
        super.updateItem(item, empty);
        Rectangle rect = new Rectangle(100, 20);
        if (item != null) {
            rect.setFill(Color.web(item));
            setGraphic(rect);
        } else {
            setGraphic(null);
        }
    }
}
```

Figure 29–7 shows how this customized list looks in the ListViewSample of the UI Control Samples project.

Figure 29–7 List View with Color Rectangles



This tutorial uses the cell factory mechanism extensively to customize UI controls. You can customize these controls by using the cell factory mechanism or use the prefabricated cell editor implementations that provide specific data models underlying the visualization. Table 29–1 lists the corresponding classes available in the JavaFX API.

Table 29–1 Cell Editor Classes for the List View, Tree View, and Table View Controls

Control	Cell Editor Classes
List view	■ `CheckBoxListCell` ■ `ChoiceBoxListCell` ■ `ComboBoxListCell` ■ `TextFieldListCell`
Tree view	■ `CheckBoxTreeCell` ■ `ChoiceBoxTreeCell` ■ `ComboBoxTreeCell` ■ `TextFieldTreeCell`
Table view	■ `CheckBoxTableCell` ■ `ChoiceBoxTableCell` ■ `ComboBoxTableCell` ■ `ProgressBarTableCell` ■ `TextFieldTableCell`
Tree Table View	■ `CheckBoxTreeTableCell` ■ `ChoiceBoxTreeTableCell` ■ `ComboBoxTreeTableCell` ■ `ProgressBarTreeTableCell` ■ `TextFieldTreeTableCell`

Each cell editor class draws a specific node inside the cell. For example, the `CheckBoxListCell` class draws a `CheckBox` node inside the list cell.

To evaluate more cell factory and cell editor use cases, see the Table View, Tree View, and Combo Box chapters.

Related Documentation and Resources

- Skinning JavaFX Applications with CSS

- JavaFX CSS Reference Guide
- JavaFX News, Demos, and Insight

30

UI Controls on the Embedded Platforms

In addition to the full range of desktop features, the JavaFX Software Development Kit (SDK) introduces new operational capabilities of UI controls designed for touch-enabled devices.

This chapter describes the specific of using JavaFX UI controls in the embedded environments.

Embedded Runtime Features

Although JavaFX UI controls do not include any additional public APIs to work in the embedded environments, significant implementation changes were made to enable developers using the desktop controls in their JavaFX applications for touch-enabled devices.

Support for Touch-Enabled Devices

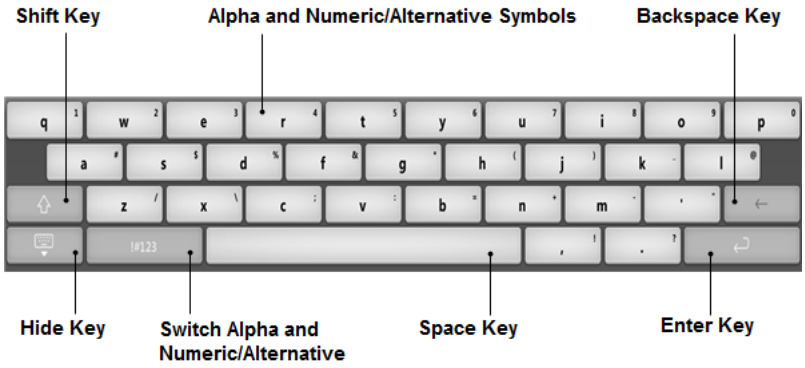
The JavaFX SDK introduces additional operational capabilities of UI controls for touch-enabled devices: gestures and touches. Gestures can be generated for both touchscreen and trackpad actions, but touches are delivered for touchscreen actions only. In the current release JavaFX SDK supports only a single-point touch and only swipe gestures.

See the chapter on working with events from touch-enabled devices in *Handling Events* for more information about handling gestures and touch events in JavaFX.

Virtual Keyboard

The virtual keyboard is a control that enables typing text on devices that lack a hardware keyboard. It operates the same way as any hardware keyboard except that numbers and symbols are located one tap away due to space constraints. Figure 30–1 shows an example of the virtual keyboard.

Figure 30–1 Virtual Keyboard



The virtual keyboard is shown automatically when a text input field is in focus. Note that the control that is associated with the keyboard remains visible when the keyboard is displayed. There is no need to push the parent stage up. The keyboard slides in from the bottom area pushing the parent stage up in order to keep the control that the keyboard is associated with visible on the screen.

When the virtual keyboard appears, users can type a character by tapping the appropriate key. After that, character that is associated with the key is sent to the control. User can type alpha, numeric, and symbol characters. To switch between the keyboards, they need to use the ABC/!#123 key and the Shift key. To access less frequently used characters, users need to press the appropriate key for longer time and choose the target key from the pop-up menu, as shown in Figure 30–2.

Figure 30–2 Typing Less Frequently Used Characters

Table 30–1 lists possible navigating actions within the virtual keyboard on touch-enabled devices.

Table 30–1 Navigation Within the Virtual Keyboard

Action	Sequence of Keys
Type a letter.	On the alpha keyboard: type a letter key. On the numeric/alternative keyboard: tap the "ABC" key, then type a letter key.
Type a capital letter.	On the alpha keyboard: tap the Shift key once, then type a letter key.
Type capital letters.	On the alpha keyboard: tap the Shift key twice, then tap letter keys.

Table 30–1 (Cont.) Navigation Within the Virtual Keyboard

Action	Sequence of Keys
Type a number.	On the numeric/alternative keyboard: type a number. On the alpha keyboard: tap the "!#123" key and type any number you want.
Type an alternative symbol.	On the numeric/alternative keyboard: type an alternative symbol. On the alpha keyboard: tap the "!#123" key and type an alternative symbol.
Delete a character.	Tap the Backspace key to delete character on the left side of the caret.
Confirmed entered data.	Tap the Enter key to confirm entered data.
Hide the virtual keyboard.	Tap a Hide button that is left-most in the lowest row of the keyboard.

When UI interface of your application requires typing email address, URL, or only numeric symbols, you can set one of the alternative keyboard layouts. This setting is defined for a particular text control as shown in Example 30–1.

Example 30–1 Setting Email Layout for the Virtual Keyboard

```
final TextField emailAddress = new TextField("myEmail@example.com");
text.getProperties().put("vkType", "email");
```

Currently, the `vkType` property supports the following values: `numeric`, `url`, `email`, and `text`. The last one corresponds to the default layout.

Figure 30–3 shows the email layout of the keyboard. In addition to letters, number, and alternative symbols, the layout has "@" and ".com" (".org") keys that are particularly helpful for quick typing email addresses.

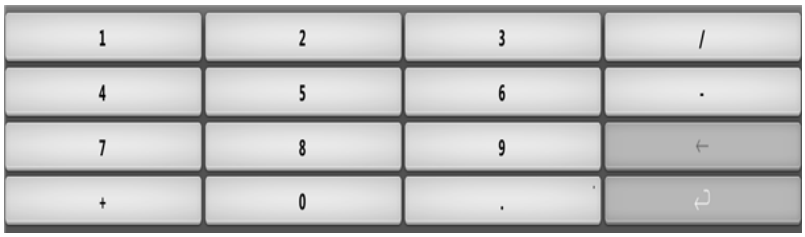
Figure 30–3 Email Layout of the Virtual Keyboard

The URL layout of the virtual keyboard, shown in Figure 30–4, does not have the Space key. Instead, it provides capabilities to type in "www." and "http//".

Figure 30–4 URL Layout of the Virtual Keyboard

In some UIs, users are supposed to enter only numbers. Figure 30–5 shows a simplified numeric layout of the virtual keyboard that implements this task. No Shift key, neither alternative symbols are available in this layout.

Figure 30–5 Numeric Layout of the Virtual Keyboard



Appearance of UI Controls on Embedded Platforms

The `embedded.css` style sheet is designed specifically for touch-enabled platforms. It supplements the `modena.css` style sheet and overrides some of its styles. The `UAStylesheetLoader` class manages the styles of UI controls switching the CSS files when a touch-enabled platform is detected.

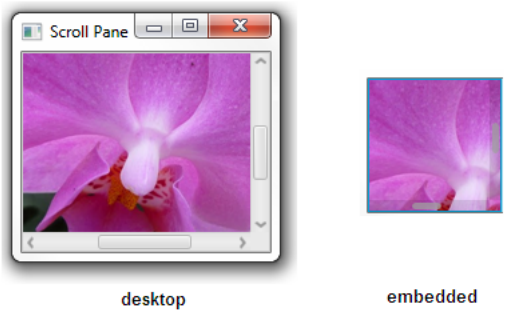
The `embedded.css` style sheet alters the appearance of the following controls:

- `ScrollBar`, `ScrollPane` — redefines the scrolling elements
- `TextArea`, `TextField`, `PasswordField`, `DatePicker` — visualizes carets
- `ContextMenu` — introduces the horizontal context menu

Scrolling Controls

A scroll pane on a mobile touch platform performs scrolling as a result of touch gestures targeted on scroll view content. Scroll bars are hidden when the user is not interacting with the scroll pane, even if the content is bigger than the scroll pane. Scroll bars appear automatically after the touch gestures are performed, but only as an indicator of content size and position, not as an active control that the user uses for scrolling. When scroll bars appear, they overlap visible content and do not shift it like on the desktop platform, as shown in Figure 30–6.

Figure 30–6 Appearance of the Scroll Pane in Desktop and Embedded Environments



Scroll bars disappear automatically when the content stops scrolling.

Text Input Controls

On touch-enabled platforms, `TextArea`, `TextField`, and `PasswordField` are implemented as rectangular spaces where users can enter a single line or multiple lines of text. Interaction with text controls is similar for all platforms. However, for embedded environments, the caret is used to facilitate navigation and text selection. The virtual keyboard is used to enter characters into text controls.

Figure 30–7 shows the TextFieldSample application running on a touch-enabled platform. The caret indicates the place where a user enters the symbols with the help of the virtual keyboard. See the code at TextFieldSample.java.

Figure 30–7 Appearance of the TextField Control

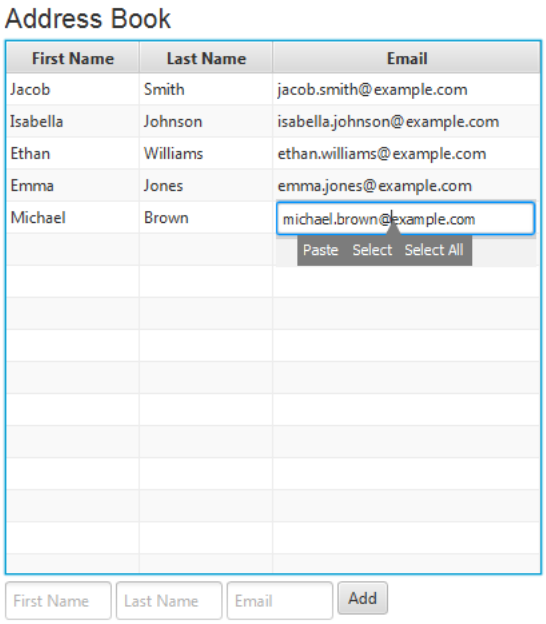
The virtual keyboard is shown automatically when a text input field is in focus and it reacts when devices are rotated. The virtual keyboard appears when users need to enter text into `TextField`, `TextArea`, `PasswordField`, and other UI controls if their edit state is enabled. To hide the keyboard, users can tap a Hide button or tap outside the control.

Context Menus

The default appearance of the context menu is changed for the embedded environments to provide the horizontal layout of the menu items.

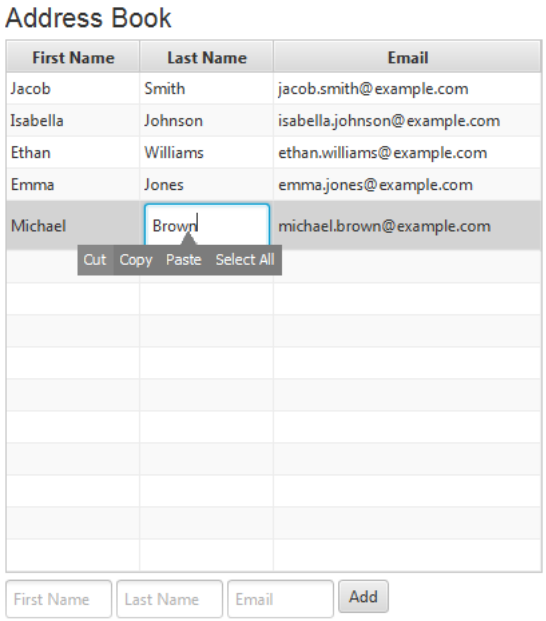
The context menu is invoked when users tap and hold. If nothing is in the clipboard, then the context menu shows Select and Select All menu items. If something is in the clipboard, then the context menu looks as shown in Figure 30–8.

Figure 30–8 Appearance of a Context Menu



If the text in the text control is selected, then the Cut, Copy, Paste, and Select All items are displayed, as shown in Figure 30–9.

Figure 30–9 Appearance of a Context Menu After Selection Is Performed



You can find another style sheet for the touch-enabled platforms in the JavaFX SDK. The `embedded-qvga.css` file defines the CSS styles for embedded devices that have QVGA screens. In particular, it specifies alternative padding for the `TextField` and `TextArea` controls.

UI Controls Features Available on the Embedded Platforms

In addition to the visual changes mentioned in the previous section, the behavior of UI controls is modified to accommodate single point touches and swipe gestures. Table 30–2 summarizes these changes.

Table 30–2 Features Specific to Embedded Runtime

UI Control	User Action	Touch Support
`Button`, `Hyperlink`, `MenuButton`, `ToggleButton`	Tap.	Activates the button.
`CheckBox`	Tap.	Switches the selected and deselected states of a check box.
`ComboBox`	Tap a drop-down icon.	Displays or hides the active list. The item that is displayed in the text field is selected when the active list is open.
	Tap on the text field.	For a noneditable combo box: displays the active list. For an editable combo box: places the caret into the text field.
	Tap on an item in the list.	Closes the active list and commits the value.
	Tap outside the list.	Closes the list.
	Drag.	Scrolls the content continuously following the drag gesture.
	Drop.	Stops scrolling.
	Swipe.	Invokes accelerated scrolling.
`ListView`	Drag.	Scrolls the content continuously following the drag gesture.
	Drop.	Stops scrolling.
	Swipe.	Invokes accelerated scrolling.
	Tap.	Selects an item and activates the action associated with it. If the content is scrolling, then it stops scrolling.
	Double-tap.	If enabled, enters inline editing mode.

Table 30–2 (Cont.) Features Specific to Embedded Runtime

UI Control	User Action	Touch Support
`TextField`, `TextArea`, `PasswordField`	Tap.	Sets the caret.
	Double-tap.	Selects the content.
	Tap and hold.	Opens a context menu.
	Swipe.	Invokes accelerated scrolling.
	Drag.	Scrolls the content continuously following the drag gesture.
	Drop.	Stops scrolling. When the user "over-scrolls" the content beyond the `TextArea` border, the drop gesture releases the content and it smoothly returns to the border of the `TextArea`.
`RadioButton`	Tap.	If the radio button is selected, then no action is performed. If it is deselected, then the tap makes it selected and other radio buttons in the group are deselected.
`ScrollBar`, `ScrollView`	Drag.	Scrolls the content continuously following the drag gesture.
	Drop.	Stops scrolling.
	Swipe.	Invokes accelerated scrolling.
	Tap.	Selects an item and activates the action associated with it. If the content is scrolling, then it stops scrolling.
`TableView`, `TreeView`	Tap.	Selects the cell or invokes an action if the cell is actionable. Expands or collapses the node for the tree view.
	Double-tap.	Switches to editing if the selected cell supports editing.
	Drag.	Scrolls the content continuously following the drag gesture.
	Drop.	Stops scrolling.
	Swipe.	Invokes accelerated scrolling of content.
`ColorPicker`	Short tap a color in the Color field or Custom Colors area in the Color Palette.	Updates the color in the color chooser. Closes the color palette. Applies the color.
	Tap outside the palette.	Closes the palette.
	Tap any place in the Color field of the Custom Color dialog window.	Updates values in RGB, HSB, and Web panes. Updates New Color in the Color Preview.
	Tap any place in the Hue bar of the Custom Color dialog window.	Updates values in RGB, HSB, and Web panes. Updates New Color in the Custom Color dialog window.
	Tap any place on the slider, or drag the sliders to the left or right.	Dismisses the dialog window. Closes the color palette. Updates the color chooser. Applies the color.

Table 30–2 (Cont.) Features Specific to Embedded Runtime

UI Control	User Action	Touch Support
`Pagination`	Tap a page button.	Opens the selected page.
	Tap the Next button, Swipe left.	Opens the next page.
	Tap the Previous button, Swipe right.	Opens the previous page.
`DatePicker`	Tap the date field.	Displays caret in the field; virtual keyboard appears.
	Tap the calendar icon.	Displays and hides the calendar.
	Tap the LEFT or RIGHT calendar arrows.	Display previous or next month and year.
	Tap any date in the calendar.	Updates the field with selected date and closes the calendar.
	Tap outside the calendar.	Closes the calendar without updating the field.

All the actions mentioned in Table 30–2 are enabled when the application is run on the touch-enabled platform, and they do not require any additional code.

UI Controls Features That Are Not Available on Embedded Touch Platforms

The following list shows some features of UI controls that are currently disabled on embedded platforms:

- Column resizing, column rearranging, and data sorting in table views
- Multi selection in list views and tree views
- Text copy support in password fields

Other Features That Are Not Available on Embedded Touch Platforms

Fhe following JavaFX features are not supported on the embedded platforms:

- Functionality of the `Stage` class related to desktop interaction is not available on some embedded platforms. For example, on these platforms a `Stage` window will have not have a title or decorations and will not be resizable by the user.
- Web component – a user interface component that provides a web viewer and full browsing functionality through its API. See Adding HTML Content to JavaFX Applications for more information.
- Media – media functionality available through the Java APIs for JavaFX. See Incorporating Media Assets Into JavaFX Applications for more information.

Part III

Working with JavaFX Charts

This tutorial describes the graphical charts available in the `javafx.scene.chart` package of the JavaFX SDK and contains the following chapters:

- Pie Chart

  Describes a chart that represents data in a form of circle divided into triangular wedges called slices.

- Line Chart

  Describes the line chart, a type of two-axis chart that presents data as a series of points connected by straight lines

- Area Chart

  Describes the area chart that presents data as an area between a series of points connected by straight lines and the axis.

- Bubble Chart

  Describes the bubble chart, a two-axis chart that plots bubbles for the data points in a series.

- Scatter Chart

  Describes the scatter chart, a two-axis chart that presents its data as a set of points.

- Bar Chart

  Describes the bar chart, a two-axis chart that presents discrete data with rectangular bars.

Each chapter provides code samples and applications to illustrate how to use a particular chart. You can find the source files of the applications and the corresponding NetBeans projects in the Chart Samples appendix.

31

Pie Chart

This chapter describes a chart that represents data in a form of circle divided into triangular wedges called slices. Each slice represents a percentage that corresponds to a particular value.

Figure 31–1 shows a pie chart created by using the `PieChart` class. The colors of the slices are defined by the order of the corresponding data items added to the `PieChart.Data` array.

Figure 31–1 Typical Pie Chart

Creating a Pie Chart

To create a pie chart in your JavaFX application, at a minimum, you must instantiate the `PieChart` class, define the data, assign the data items to the `PieChart` object, and add the chart to the application. When creating the chart data, define as many `PieChart.Data` objects for as many slices you want to appear. Each `PieChart.Data` object has two fields: the name of the pie slice and its corresponding value. Example 31–1 creates the basic pie chart.

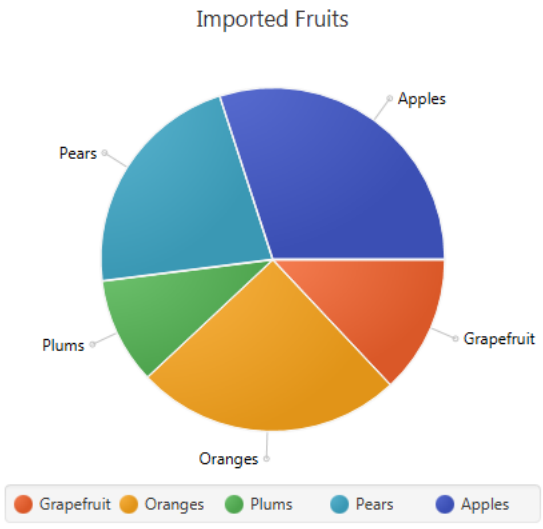
Example 31–1 Creating a Pie Chart

```
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.chart.*;
```

```
import javafx.scene.Group;

public class PieChartSample extends Application {

    @Override public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Imported Fruits");
        stage.setWidth(500);
        stage.setHeight(500);

        ObservableList<PieChart.Data> pieChartData =
                FXCollections.observableArrayList(
                new PieChart.Data("Grapefruit", 13),
                new PieChart.Data("Oranges", 25),
                new PieChart.Data("Plums", 10),
                new PieChart.Data("Pears", 22),
                new PieChart.Data("Apples", 30));
        final PieChart chart = new PieChart(pieChartData);
        chart.setTitle("Imported Fruits");

        ((Group) scene.getRoot()).getChildren().add(chart);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

The result of compiling and running this application is shown in Figure 31–2.

Figure 31–2 Imported Fruits Charts



In addition to the basic settings, Example 31–1 specifies the title of the chart by calling the `setTitle` method.

Setting a Pie Chart and a Legend

The default view of a pie chart includes the pie with the labels and the chart legend. The values of the labels are retrieved from the name field of a `PieChart.Data` object. You can manage the appearance of the labels by using the `setLabelsVisile` method. Similarly, you can manage the appearance of the chart legend by calling the `setLegendVisible` method.

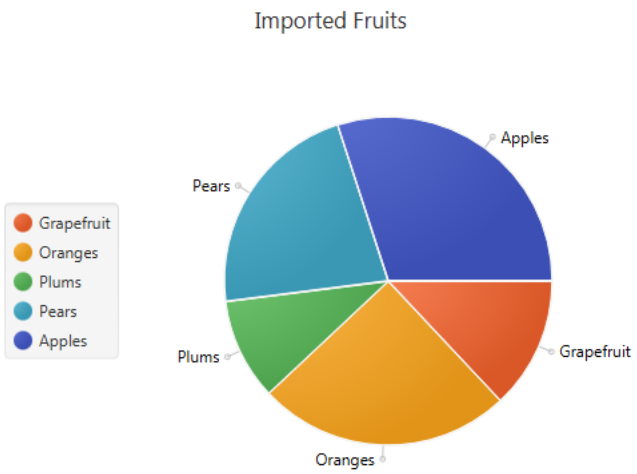
You can alter the position of the labels and the legend. With the `setLabelLineLength` method, you can specify the length of the line between the circumference of the pie and the labels. Use the `setLegendSide` method to alter the default position of the legend relative to the pie. Example 31–2 demonstrates how to apply these methods to the chart created in Example 31–1.

Example 31–2 Changing Position of the Labels and the Legend

```
chart.setLabelLineLength(10);
chart.setLegendSide(Side.LEFT);
```

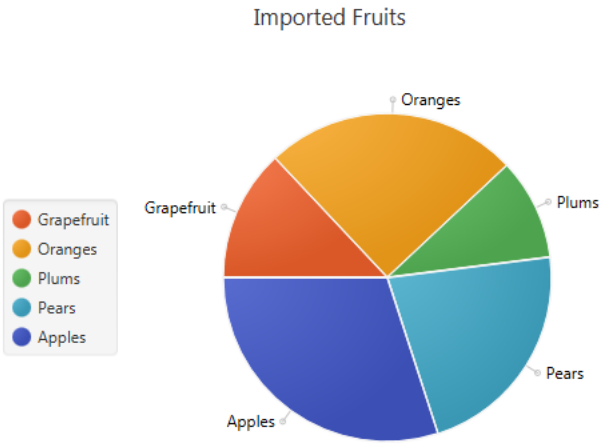
The result of adding these lines to the application code, compiling, and then running the application is shown in Figure 31–3.

Figure 31–3 Alternative Position of the Chart Legend and Labels



Your application might require that you alter the direction in which the slices are placed in the pie. By default, the slices are placed clockwise. However, you can change this by specifying the `false` value for the `setClockwise` method `chart.setClockwise(false)`. Use this method in combination with the `setStartAngle` method to attain the desired position of the slices. Figure 31–4 shows how the appearance of the pie chart changes when the `setStartAngle(180)` method is called for the `chart` object.

Figure 31–4 Changing the Start Angle of the Pie Chart Slices



Processing Events for a Pie Chart

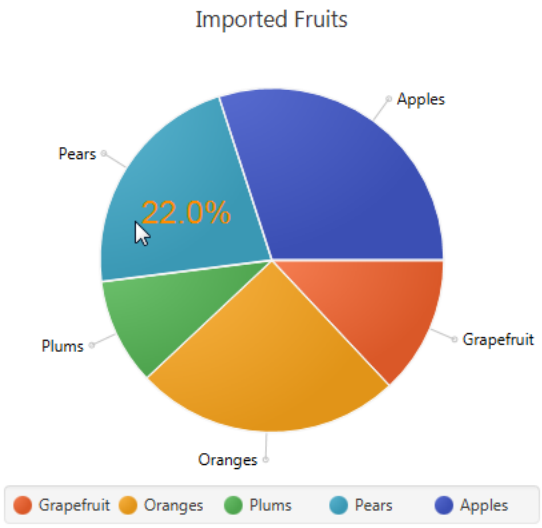
Although a pie chart slice is not a `Node` object, each `PieChart.Data` element has a node associated with it, which can be used to analyze events and process them accordingly. The code fragment shown in Example 31–3 creates an `EventHandler` object to process a `MOUSE_PRESSED` event that falls into a particular chart slice.

Example 31–3 Processing Mouse Events for a Pie Chart

```
final Label caption = new Label("");
caption.setTextFill(Color.DARKORANGE);
caption.setStyle("-fx-font: 24 arial;");

for (final PieChart.Data data : chart.getData()) {
    data.getNode().addEventHandler(MouseEvent.MOUSE_PRESSED,
        new EventHandler<MouseEvent>() {
            @Override public void handle(MouseEvent e) {
                caption.setTranslateX(e.getSceneX());
                caption.setTranslateY(e.getSceneY());
                caption.setText(String.valueOf(data.getPieValue()) + "%");
            }
        });
}
```

When you add this code fragment to the application code and compile and run it, the application starts reacting to the mouse clicks. Figure 31–5 shows the value displayed for Pears when a user clicks the corresponding slice.

Figure 31–5 Processing the Mouse-Pressed Events for a Pie Chart



By using this coding pattern, you can process various events or apply visual effects to the whole chart as well as to its slices.

Related API Documentation

- `PieChart`
- `PieChart.Data`
- `Chart`

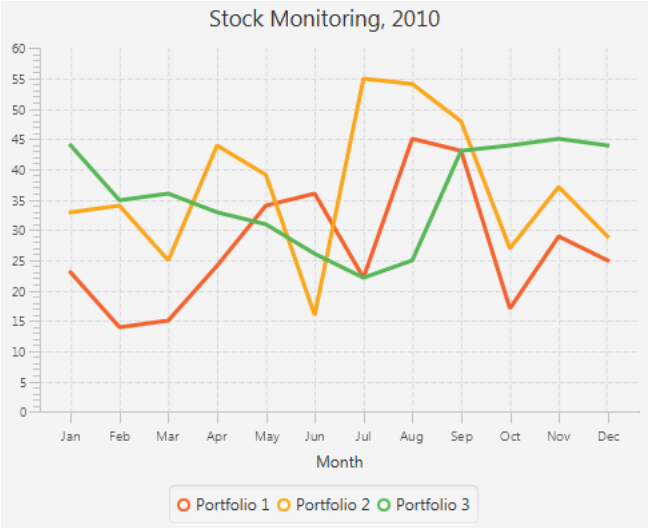
32

Line Chart

This chapter describes the line chart, a type of two-axis chart that presents data as a series of points connected by straight lines.

The line chart is often used to illustrate the dynamics of data over a particular interval of time. Figure 32–1 demonstrates a typical line chart with three series of data.

Figure 32–1 Example of a Line Chart



Each two-axis chart has two axes, the plot of data points, and the legend. You can also specify a title for the chart.

Chart Settings

For each chart, you can specify the title and its position relative to the graph. The title can be located on the top, right, left, or bottom of the graph. Similarly, you can specify the position of the chart legend.

For a two-axis graph, you can manage the appearance of the chart plot, the chart area where the graphical symbols corresponding to data values are rendered. You can set alternative columns and rows as well as horizontal and vertical grid lines and zero lines.

You can alter the default appearance of each chart by defining the following settings:

- The axis label

- The axis position relative to the chart plot
- The upper and lower boundaries of the axis range
- The minimum and maximum tick marks, tick units, the gap between two tick marks, and tick labels

You can also specify that any changes to the axis and its range will be animated, or you can enable the axis to automatically determine its range from the data.

Chart Data

The `XYChart` class, a super class for all two-axis charts, provides basic capabilities for building area, line, bar, scatter, and bubble charts. Use the `XYChart.Data` class to specify the data model for these types of charts. The `xValue` property defines the value of a chart element to be plotted on the X axis, and the `yValue` property defines the value for the Y axis. You can also set the extra value for each chart element. This value can be plotted in any way the chart needs, or it can be used to store additional information about the chart element. For example, it can be used to define a radius for bubble charts.

For two-axis charts, you can define several series of data by using the `XYChart.Series` class. You can also assign a particular name to each series to display in the chart legend.

Creating a Line Chart

To create a line chart, at a minimum, you must define two axes, create the `LineChart` object by instantiating the `LineChart` class, create one or more series of data by using the `XYChart.Series` class, and assign the data to the chart. Example 32–1 implements these tasks.

Example 32–1 Simple Line Chart

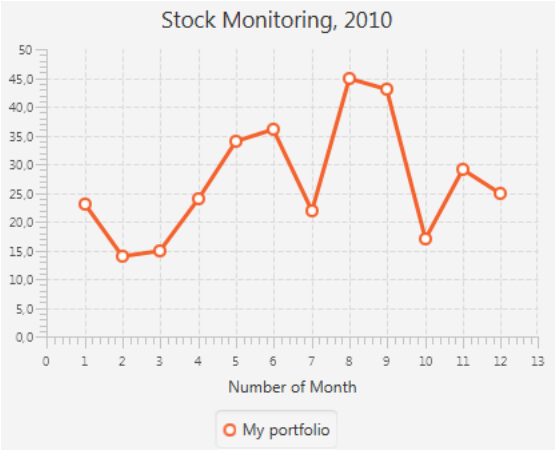
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.LineChart;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;


public class LineChartSample extends Application {

    @Override public void start(Stage stage) {
        stage.setTitle("Line Chart Sample");
        //defining the axes
        final NumberAxis xAxis = new NumberAxis();
        final NumberAxis yAxis = new NumberAxis();
        xAxis.setLabel("Number of Month");
        //creating the chart
        final LineChart<Number,Number> lineChart =
                new LineChart<Number,Number>(xAxis,yAxis);

        lineChart.setTitle("Stock Monitoring, 2010");
        //defining a series
        XYChart.Series series = new XYChart.Series();
        series.setName("My portfolio");
        //populating the series with data
```

```
        series.getData().add(new XYChart.Data(1, 23));
        series.getData().add(new XYChart.Data(2, 14));
        series.getData().add(new XYChart.Data(3, 15));
        series.getData().add(new XYChart.Data(4, 24));
        series.getData().add(new XYChart.Data(5, 34));
        series.getData().add(new XYChart.Data(6, 36));
        series.getData().add(new XYChart.Data(7, 22));
        series.getData().add(new XYChart.Data(8, 45));
        series.getData().add(new XYChart.Data(9, 43));
        series.getData().add(new XYChart.Data(10, 17));
        series.getData().add(new XYChart.Data(11, 29));
        series.getData().add(new XYChart.Data(12, 25));

        Scene scene  = new Scene(lineChart,800,600);
        lineChart.getData().add(series);

        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

In this example, both vertical and horizontal axes are created by using the `NumberAxis` class, a subclass of the `Axis` class, to represent numerical values. Having declared both `X` and `Y` axes numerical, you should specify `Number` parameters for `XYChart.Data` objects when creating a series of data. The first parameters of `XYChart.Data` objects define values for the horizontal axis, whereas, the second parameters of the `XYChart.Data` objects define values for the vertical axis.

The result of compiling and running this application is shown in Figure 32–2.

Figure 32–2 Line Chart with One Series of Data



The line chart shown in Figure 32–2 uses symbols to highlight each data item on the chart. If you want to show trends instead of specific data values on your line chart, you can disable the chart symbols as shown in Example 32–2.

Example 32–2 Disabling Symbols for a Line Chart

```
lineChart.setCreateSymbols(false);
```

The sample of a trend chart is shown in Figure 32–1.

In Figure 32–1, axes are shown in their default positions relative to the chart plot. However, you can display an axis on another side of the chart plot by applying the `setSide` method. Example 32–3 demonstrates how to move the horizontal axis to the top of the chart plot.

Example 32–3 Specifying the Axis Side

```
xAxis.setSide(Side.TOP);
```

Creating Categories for a Line Chart

Use the `CategoryAxis` class instead of the `NumberAxis` class to render non-numerical data in a line chart.

Examine the modified code of the application shown in Example 32–4. It creates the horizontal axis by instantiating the `CategoryAxis` class. The declaration of the `LineChart` object is modified to accommodate the change of the X axis type.

Example 32–4 Using Category Axis to Show Months

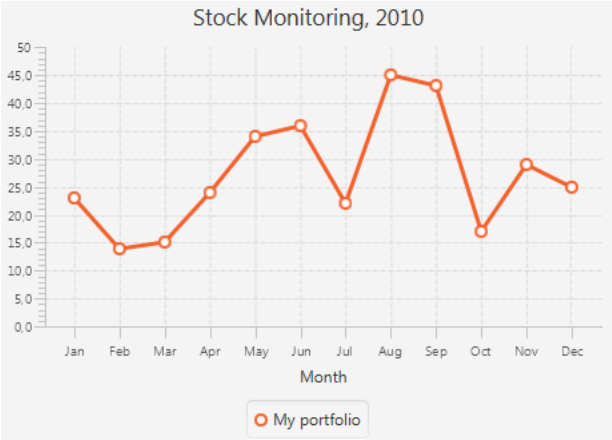
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.LineChart;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;


public class LineChartSample extends Application {

    @Override public void start(Stage stage) {
        stage.setTitle("Line Chart Sample");
        final CategoryAxis xAxis = new CategoryAxis();
        final NumberAxis yAxis = new NumberAxis();
        xAxis.setLabel("Month");

        final LineChart<String,Number> lineChart =
                new LineChart<String,Number>(xAxis,yAxis);

        lineChart.setTitle("Stock Monitoring, 2010");

        XYChart.Series series = new XYChart.Series();
        series.setName("My portfolio");

        series.getData().add(new XYChart.Data("Jan", 23));
        series.getData().add(new XYChart.Data("Feb", 14));
        series.getData().add(new XYChart.Data("Mar", 15));
        series.getData().add(new XYChart.Data("Apr", 24));
        series.getData().add(new XYChart.Data("May", 34));
        series.getData().add(new XYChart.Data("Jun", 36));
        series.getData().add(new XYChart.Data("Jul", 22));
        series.getData().add(new XYChart.Data("Aug", 45));
        series.getData().add(new XYChart.Data("Sep", 43));
        series.getData().add(new XYChart.Data("Oct", 17));
        series.getData().add(new XYChart.Data("Nov", 29));
```

```
        series.getData().add(new XYChart.Data("Dec", 25));


        Scene scene  = new Scene(lineChart,800,600);
        lineChart.getData().add(series);

        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

The `XYChartData` objects contain the month name and the corresponding numerical value. The label of the horizontal axis is modified accordingly.

The result of compiling and running the modified code of the application is shown in Figure 32–3.

Figure 32–3 Horizontal Category Axis



Often, line charts enable analyzing different set of data over the same period of time. Use several series of `XYChart.Data` objects to implement this task in your application.

Adding Series to the Line Chart

Example 32–5 provides source code for the stock monitoring application with three series of data. In addition to the series used in Example 32–4, the previous example, two new series are declared.

The series are assigned to the chart by using consecutive calls of the `getData` and `addAll` methods.

Example 32–5 Adding Two More Series to the Stock Monitoring Sample

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.LineChart;
import javafx.scene.chart.NumberAxis;
```

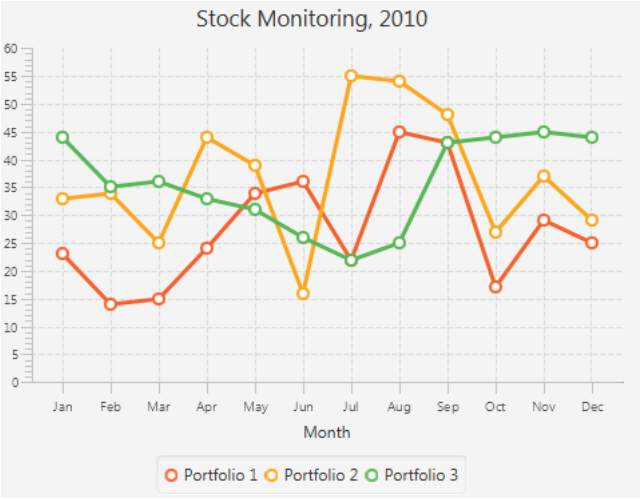
```
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;


public class LineChartSample extends Application {

    @Override public void start(Stage stage) {
        stage.setTitle("Line Chart Sample");
        final CategoryAxis xAxis = new CategoryAxis();
        final NumberAxis yAxis = new NumberAxis();
         xAxis.setLabel("Month");
        final LineChart<String,Number> lineChart =
                new LineChart<String,Number>(xAxis,yAxis);

        lineChart.setTitle("Stock Monitoring, 2010");

        XYChart.Series series1 = new XYChart.Series();
        series1.setName("Portfolio 1");

        series1.getData().add(new XYChart.Data("Jan", 23));
        series1.getData().add(new XYChart.Data("Feb", 14));
        series1.getData().add(new XYChart.Data("Mar", 15));
        series1.getData().add(new XYChart.Data("Apr", 24));
        series1.getData().add(new XYChart.Data("May", 34));
        series1.getData().add(new XYChart.Data("Jun", 36));
        series1.getData().add(new XYChart.Data("Jul", 22));
        series1.getData().add(new XYChart.Data("Aug", 45));
        series1.getData().add(new XYChart.Data("Sep", 43));
        series1.getData().add(new XYChart.Data("Oct", 17));
        series1.getData().add(new XYChart.Data("Nov", 29));
        series1.getData().add(new XYChart.Data("Dec", 25));

        XYChart.Series series2 = new XYChart.Series();
        series2.setName("Portfolio 2");
        series2.getData().add(new XYChart.Data("Jan", 33));
        series2.getData().add(new XYChart.Data("Feb", 34));
        series2.getData().add(new XYChart.Data("Mar", 25));
        series2.getData().add(new XYChart.Data("Apr", 44));
        series2.getData().add(new XYChart.Data("May", 39));
        series2.getData().add(new XYChart.Data("Jun", 16));
        series2.getData().add(new XYChart.Data("Jul", 55));
        series2.getData().add(new XYChart.Data("Aug", 54));
        series2.getData().add(new XYChart.Data("Sep", 48));
        series2.getData().add(new XYChart.Data("Oct", 27));
        series2.getData().add(new XYChart.Data("Nov", 37));
        series2.getData().add(new XYChart.Data("Dec", 29));

        XYChart.Series series3 = new XYChart.Series();
        series3.setName("Portfolio 3");
        series3.getData().add(new XYChart.Data("Jan", 44));
        series3.getData().add(new XYChart.Data("Feb", 35));
        series3.getData().add(new XYChart.Data("Mar", 36));
        series3.getData().add(new XYChart.Data("Apr", 33));
        series3.getData().add(new XYChart.Data("May", 31));
        series3.getData().add(new XYChart.Data("Jun", 26));
        series3.getData().add(new XYChart.Data("Jul", 22));
        series3.getData().add(new XYChart.Data("Aug", 25));
        series3.getData().add(new XYChart.Data("Sep", 43));
        series3.getData().add(new XYChart.Data("Oct", 44));
        series3.getData().add(new XYChart.Data("Nov", 45));
```

```
        series3.getData().add(new XYChart.Data("Dec", 44));

        Scene scene  = new Scene(lineChart,800,600);
        lineChart.getData().addAll(series1, series2, series3);

        stage.setScene(scene);
        stage.show();
    }


    public static void main(String[] args) {
        launch(args);
    }
}
```

Each series of data has its unique name defined by using the `setName` method.

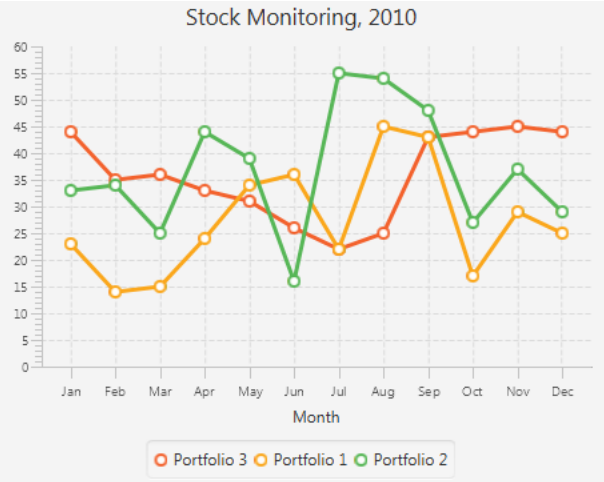
The result of compiling and running this application is shown in Figure 32–4.

Figure 32–4 Stock Monitoring Example with Tree Series of Data



Note that the different colors of the lines are defined by the declared order of the corresponding series in the `addAll` method. Change the order as follows: `lineChart.getData().addAll(series3, series1, series2)`, and then compile and run the application. The modified output is shown in Figure 32–5.

Figure 32–5 Alternative Order of Series in the Line Chart



Related API Documentation

- `LineChart`
- `Chart`
- `XYChart`
- `XYChart.Data`
- `XYChart.Series`
- `Axis`
- `NumberAxis`
- `CategoryAxis`

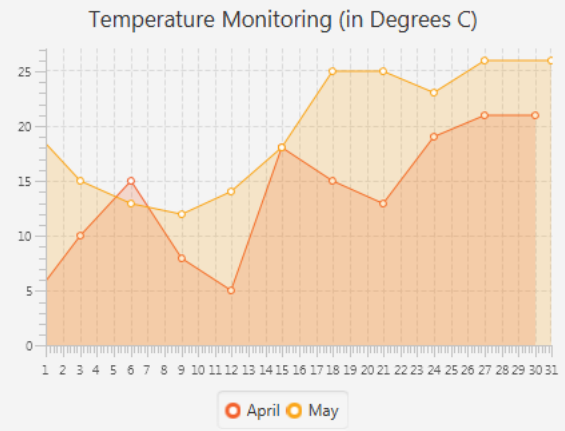
33

Area Chart

This chapter describes the area chart, yet another type of a two-axis chart.

Similar to line charts, it presents data as a series of points connected by straight lines. However, the area between the axis and the line is painted with color. Each series of data is painted with a different color. Figure 33–1 shows an area chart with two series of data.

Figure 33–1 Typical Area Chart



Creating an Area Chart

To create a simple area chart in your application, at minimum, you must define two axes, create the `AreaChart` object by instantiating the `AreaChart` class, create one or more series of data by using the `XYChart.Series` class, and assign the data to the chart.

When instantiating the `AreaChart` class, you can specify the observable list with a series of data within a constructor, or add the series later by calling the `getData` and `addAll` methods on the `AreaChart` object.

Example 33–1 creates an area chart to illustrate temperature monitoring data. The example uses two series of data collected for the periods of April and May.

Example 33–1 Creating an Area Chart

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.AreaChart;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;


public class AreaChartSample extends Application {

    @Override public void start(Stage stage) {
        stage.setTitle("Area Chart Sample");
        final NumberAxis xAxis = new NumberAxis(1, 31, 1);
        final NumberAxis yAxis = new NumberAxis();
        final AreaChart<Number,Number> ac =
            new AreaChart<>(xAxis,yAxis);
        ac.setTitle("Temperature Monitoring (in Degrees C)");

        XYChart.Series seriesApril= new XYChart.Series();
        seriesApril.setName("April");
        seriesApril.getData().add(new XYChart.Data(1, 4));
        seriesApril.getData().add(new XYChart.Data(3, 10));
        seriesApril.getData().add(new XYChart.Data(6, 15));
        seriesApril.getData().add(new XYChart.Data(9, 8));
        seriesApril.getData().add(new XYChart.Data(12, 5));
        seriesApril.getData().add(new XYChart.Data(15, 18));
        seriesApril.getData().add(new XYChart.Data(18, 15));
        seriesApril.getData().add(new XYChart.Data(21, 13));
        seriesApril.getData().add(new XYChart.Data(24, 19));
        seriesApril.getData().add(new XYChart.Data(27, 21));
        seriesApril.getData().add(new XYChart.Data(30, 21));

        XYChart.Series seriesMay = new XYChart.Series();
        seriesMay.setName("May");
        seriesMay.getData().add(new XYChart.Data(1, 20));
        seriesMay.getData().add(new XYChart.Data(3, 15));
        seriesMay.getData().add(new XYChart.Data(6, 13));
        seriesMay.getData().add(new XYChart.Data(9, 12));
        seriesMay.getData().add(new XYChart.Data(12, 14));
        seriesMay.getData().add(new XYChart.Data(15, 18));
        seriesMay.getData().add(new XYChart.Data(18, 25));
        seriesMay.getData().add(new XYChart.Data(21, 25));
        seriesMay.getData().add(new XYChart.Data(24, 23));
        seriesMay.getData().add(new XYChart.Data(27, 26));
        seriesMay.getData().add(new XYChart.Data(31, 26));

        Scene scene  = new Scene(ac,800,600);
        ac.getData().addAll(seriesApril, seriesMay);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

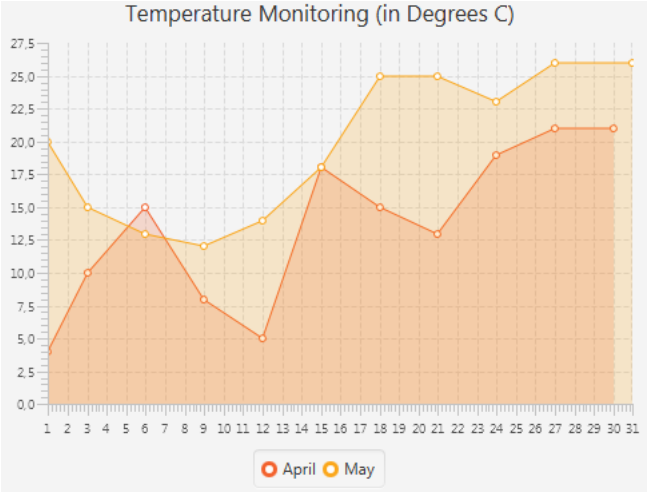
This example creates two `NumberAxis` objects to present numerical data on horizontal and vertical axes. Values rendered on the horizontal axis (X) are retrieved from the first

parameter of the `XYChart.Data` objects, whereas the second parameter provides data for the vertical axis (Y).

The series of data is assigned to the chart by using the `getData` and `addAll` methods. Because the `seriesMay` data is added last, the corresponding green area overlays the yellow area that shows April data.

The result of compiling and running the application, is shown in Figure 33–2.

Figure 33–2 Area Chart with Two Series of Data



Creating a Stacked Area Chart

You can represent data in the area chart by using the `StackedAreaChart` class. This class builds areas that are stacked so that each series adjoins but does not overlap the preceding series. Example 33–2 implements this task.

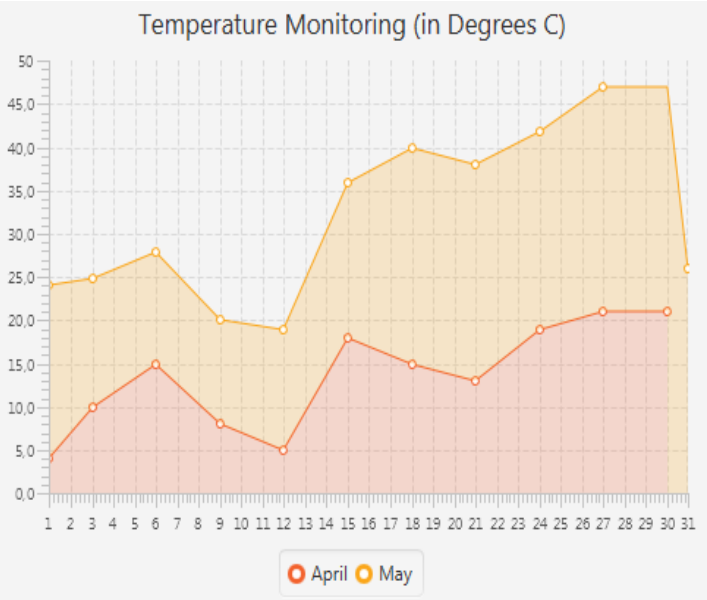
Example 33–2 Creating a Stacked Area Chart

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.StackedAreaChart;
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;

public class StackedAreaChartSample extends Application {
    final NumberAxis xAxis = new NumberAxis(1, 31, 1);
    final NumberAxis yAxis = new NumberAxis();
    final StackedAreaChart<Number, Number> sac =
        new StackedAreaChart<>(xAxis, yAxis);

    @Override
    public void start(Stage stage) {
        stage.setTitle("Area Chart Sample");
        sac.setTitle("Temperature Monitoring (in Degrees C)");
        XYChart.Series<Number, Number> seriesApril =
            new XYChart.Series<>();
```

```
        seriesApril.setName("April");
        seriesApril.getData().add(new XYChart.Data(1, 4));
        seriesApril.getData().add(new XYChart.Data(3, 10));
        seriesApril.getData().add(new XYChart.Data(6, 15));
        seriesApril.getData().add(new XYChart.Data(9, 8));
        seriesApril.getData().add(new XYChart.Data(12, 5));
        seriesApril.getData().add(new XYChart.Data(15, 18));
        seriesApril.getData().add(new XYChart.Data(18, 15));
        seriesApril.getData().add(new XYChart.Data(21, 13));
        seriesApril.getData().add(new XYChart.Data(24, 19));
        seriesApril.getData().add(new XYChart.Data(27, 21));
        seriesApril.getData().add(new XYChart.Data(30, 21));
        XYChart.Series<Number, Number> seriesMay =
            new XYChart.Series<>();
        seriesMay.setName("May");
        seriesMay.getData().add(new XYChart.Data(1, 20));
        seriesMay.getData().add(new XYChart.Data(3, 15));
        seriesMay.getData().add(new XYChart.Data(6, 13));
        seriesMay.getData().add(new XYChart.Data(9, 12));
        seriesMay.getData().add(new XYChart.Data(12, 14));
        seriesMay.getData().add(new XYChart.Data(15, 18));
        seriesMay.getData().add(new XYChart.Data(18, 25));
        seriesMay.getData().add(new XYChart.Data(21, 25));
        seriesMay.getData().add(new XYChart.Data(24, 23));
        seriesMay.getData().add(new XYChart.Data(27, 26));
        seriesMay.getData().add(new XYChart.Data(31, 26));
        Scene scene = new Scene(sac, 800, 600);
        sac.getData().addAll(seriesApril, seriesMay);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

When you compile and run this application, it creates the chart shown in Figure 33–3.

Figure 33–3 Stacked Area Chart with Two Areas



Compare the data shown in Figure 33–3 with the same data in Figure 33–2. The areas in the stacked area chart show cumulative values on the vertical axis at any given point along the horizontal axis. For example, the value on the vertical axis shown for May 15th in the stacked area chart is 36, which does not correspond to the actual temperature that day. This value represents the cumulative result for temperatures on April 15th and May 15th.

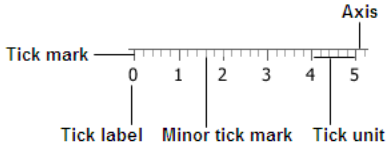
When you develop area charts in your JavaFX applications, remember that data on the vertical axes is interpreted according to the type of area charts (`AreaChart` or `StackedAreaChart`). Choose the data representation best suited for the task of the application.

Setting Axis and Tick Properties

The output of the Temperature Monitoring application in Figure 33–2 and Figure 33–3 presents the numerical values on the axes in the default double format., rather than in a user-friendly manner. For example, the month days should be integers and in the range of 1 to 31, instead of float numbers.

The JavaFX SDK API provides several methods to adjust the appearance of values rendered on chart axes. Figure 33–4 shows the main elements of the chart axis, including tick marks and tick labels that indicate numeric values of the range.

Figure 33–4 Elements of an Axis



You can specify the minimum and maximum values in the numerical range by using a constructor of the `NumberAxis` class or the corresponding methods, as shown in Example 33–3.

Example 33–3 Specifying a Data Range for the Horizontal Axis

```
//Using the NumberAxis constructor
final NumberAxis xAxis = new NumberAxis(1, 31, 1);
//Using the corresponding methods
xAxis.setLowerBound(1);
xAxis.setUpperBound(31);
xAxis.setTickUnit(1);
```

When using the three-parameter constructor of the `NumberAxis` class, remember that the first parameter defines the minimum value in the range, the second parameter is the maximum value in the range, and the third parameter defines the tick unit, a value between two tick marks on the axis.

Additionally, if you want to prohibit showing minor ticks on the horizontal axis, then specify 0 for the `minorTickCount` property, as shown in Example 33–4.

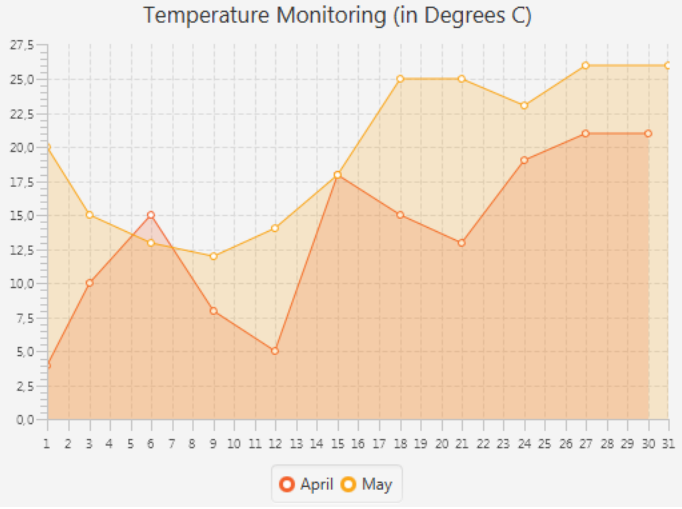
Example 33–4 Setting Zero Value for Minor Tick Count

```
xAxis.setMinorTickCount(0);
```

This property defines the number of minor ticks to be displayed between each major tick mark. By setting its value to 0, you disable the minor ticks for the horizontal axis.

When you add the code lines from Example 33–3 and Example 33–4 to the Temperature Monitoring application, the horizontal axis changes as shown in Figure 33–5.

Figure 33–5 Setting the Horizontal Axis



If your application requires no tick labels to be shown, use the `setTickLabelsVisible` method with the `false` value. Similarly, use `setTickMarkVisible` method with the `false` value if you do not want tick marks to be visible.

Use the code line shown in Example 33–5 to adjust the range of values for the vertical axis.

Example 33–5 Specifying a Data Range for the Vertical Axis

```
final NumberAxis yAxis = new NumberAxis(0, 27, 5);
```

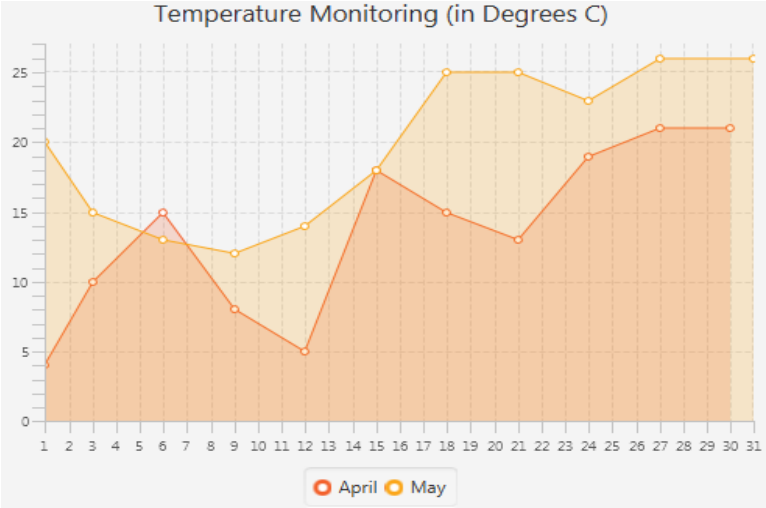
You can also adjust tick marks so that minor and major tick marks have equal length. Use the `tickLength` and `minorTickLength` properties as shown in Example 33–6.

Example 33–6 Adjusting the Length of Major and Minor Tick Marks

```
yAxis.setMinorTickLength(yAxis.getTickLength());
```

When you add code lines from Example 33–5 and Example 33–6 to the Temperature Monitoring application, the vertical axes changes as shown in Figure 33–6.

Figure 33–6 Setting the Vertical Axis



Adding Negative Values

Because the vertical axis in the Temperature Monitoring application is created by using the `NumberAxis` class, you can specify negative values for the area chart data.

Create one more series of data as shown in Example 33–7.

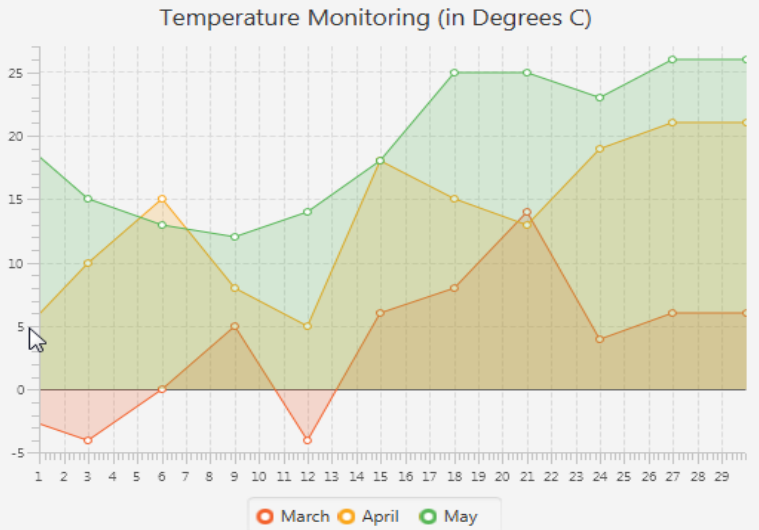
Example 33–7 Adding a Series of Data with Negative Values

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.AreaChart;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;


public class AreaChartSample extends Application {

    @Override public void start(Stage stage) {
        stage.setTitle("Area Chart Sample");
        final NumberAxis xAxis = new NumberAxis(1, 31, 1);
        xAxis.setMinorTickCount(0);
        final NumberAxis yAxis = new NumberAxis(-5, 27, 5);
        yAxis.setMinorTickLength(yAxis.getTickLength());
```

```
        yAxis.setForceZeroInRange(false);

        final AreaChart<Number,Number> ac =
            new AreaChart<Number,Number>(xAxis,yAxis);
        ac.setTitle("Temperature Monitoring (in Degrees C)");

        XYChart.Series seriesApril= new XYChart.Series();
        seriesApril.setName("April");
        seriesApril.getData().add(new XYChart.Data(0, 4));
        seriesApril.getData().add(new XYChart.Data(3, 10));
        seriesApril.getData().add(new XYChart.Data(6, 15));
        seriesApril.getData().add(new XYChart.Data(9, 8));
        seriesApril.getData().add(new XYChart.Data(12, 5));
        seriesApril.getData().add(new XYChart.Data(15, 18));
        seriesApril.getData().add(new XYChart.Data(18, 15));
        seriesApril.getData().add(new XYChart.Data(21, 13));
        seriesApril.getData().add(new XYChart.Data(24, 19));
        seriesApril.getData().add(new XYChart.Data(27, 21));
        seriesApril.getData().add(new XYChart.Data(30, 21));

        XYChart.Series seriesMay = new XYChart.Series();
        seriesMay.setName("May");
        seriesMay.getData().add(new XYChart.Data(0, 20));
        seriesMay.getData().add(new XYChart.Data(3, 15));
        seriesMay.getData().add(new XYChart.Data(6, 13));
        seriesMay.getData().add(new XYChart.Data(9, 12));
        seriesMay.getData().add(new XYChart.Data(12, 14));
        seriesMay.getData().add(new XYChart.Data(15, 18));
        seriesMay.getData().add(new XYChart.Data(18, 25));
        seriesMay.getData().add(new XYChart.Data(21, 25));
        seriesMay.getData().add(new XYChart.Data(24, 23));
        seriesMay.getData().add(new XYChart.Data(27, 26));
        seriesMay.getData().add(new XYChart.Data(31, 26));

        XYChart.Series seriesMarch = new XYChart.Series();
        seriesMarch.setName("March");
        seriesMarch.getData().add(new XYChart.Data(0, -2));
        seriesMarch.getData().add(new XYChart.Data(3, -4));
        seriesMarch.getData().add(new XYChart.Data(6, 0));
        seriesMarch.getData().add(new XYChart.Data(9, 5));
        seriesMarch.getData().add(new XYChart.Data(12, -4));
        seriesMarch.getData().add(new XYChart.Data(15, 6));
        seriesMarch.getData().add(new XYChart.Data(18, 8));
        seriesMarch.getData().add(new XYChart.Data(21, 14));
        seriesMarch.getData().add(new XYChart.Data(24, 4));
        seriesMarch.getData().add(new XYChart.Data(27, 6));
        seriesMarch.getData().add(new XYChart.Data(31, 6));


        Scene scene  = new Scene(ac,800,600);
        ac.getData().addAll(seriesMarch, seriesApril, seriesMay);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Figure 33–7 demonstrates the Temperature Monitoring application modified to display the weather data for three months: March, April, and May.

Figure 33–7 Adding Negative Data



Styling Area Charts

The color for each month in Example 33–7 is defined by the order of the corresponding data series as declared in the `addAll` method. That is why the March area in Figure 33–7 is painted yellow. You can set the color for `AreaChart` objects through CSS.

Create the `Chart.css` file and save it in the same directory as the main class of the `AreaChartSample` application. Add the lines shown in Example 33–8 to the `Chart.css` file.

Example 33–8 CSS Styles for an Area Chart

```
.default-color0.chart-area-symbol { -fx-background-color: #e9967a, #ffa07a; }
.default-color1.chart-area-symbol { -fx-background-color: #f0e68c, #fffacd; }
.default-color2.chart-area-symbol { -fx-background-color: #dda0dd, #d8bfd855; }

.default-color0.chart-series-area-line { -fx-stroke: #e9967a; }
.default-color1.chart-series-area-line { -fx-stroke: #f0e68c; }
.default-color2.chart-series-area-line { -fx-stroke: #dda0dd; }

.default-color0.chart-series-area-fill { -fx-fill: #ffa07a55; }
.default-color1.chart-series-area-fill { -fx-fill: #fffacd55; }
.default-color2.chart-series-area-fill { -fx-fill: #d8bfd855; }
```

The `chart-area-symbol` CSS class defines parameters of the symbol in the chart legend for a particular data series. Example 33–8 sets the inner and outer colors for the circles in the chart legend.

The `chart-series-area-line` CSS class sets parameters for the area chart lines. In this example, the color of the line stroke. The `chart-series-area-fill` CSS class defines the color and the opacity level of the areas.

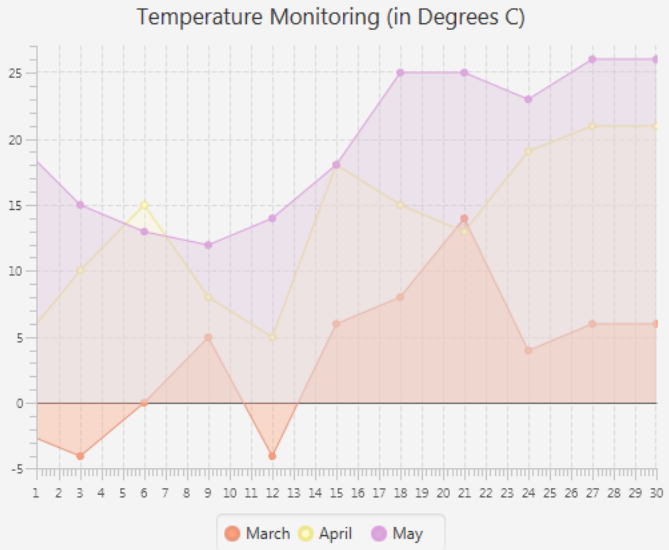
These styles are applied to the `AreaChartSample` application by using the `getStylesheets()` method of the `Scene` class, as shown Example 33–9.

Example 33–9 Applying CSS Styles to the Scene

```
scene.getStylesheets().add("areachartsample/Chart.css");
```

Compiling and running this application produces the modified appearance of the area chart shown in Figure 33–8.

Figure 33–8 Styled Area Chart



You can learn more about using CSS styles in JavaFX applications from Styling Charts with CSS.

Related API Documentation

- `AreaChart`
- `Chart`
- `XYChart`
- `XYChart.Data`
- `XYChart.Series`
- `NumberAxis`

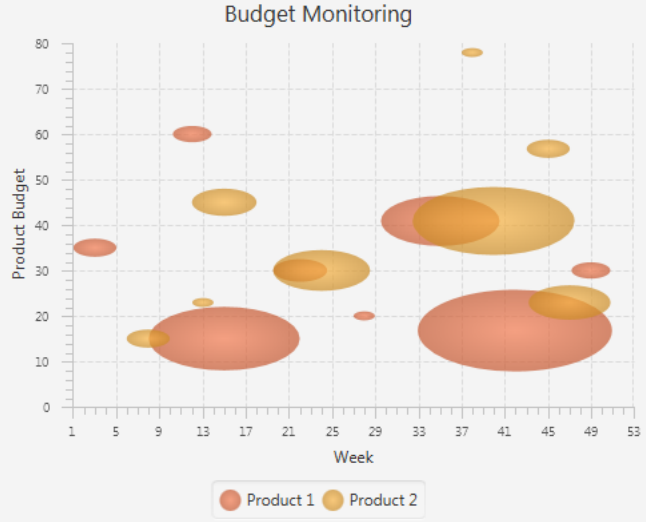
34

Bubble Chart

This chapter describes the bubble chart, a two-axis chart that plots bubbles for the data points in a series.

Each data item can be defined by two or three parameters. Figure 34–1 shows a typical bubble chart, where each data item is presented by the three following parameters: X value, Y value, and the radius of the bubble.

Figure 34–1 Typical Bubble Chart



Creating a Bubble Chart

To create a bubble chart in your JavaFX application, at minimum, you must instantiate the `BubbleChart` class, define horizontal and vertical axes, and specify one or more series of data by using constructors of the `XYChart.Data` class with two or three parameters. Example 34–1 creates a bubble chart with two series of data. Each data item is represented by X and Y values: a number of a week and an amount of product budget.

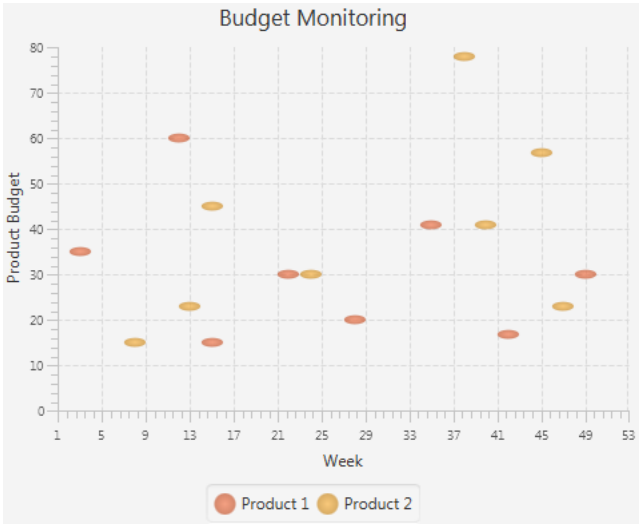
Example 34–1 Creating a Bubble Chart with Two Data Parameters

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.BubbleChart;
import javafx.scene.chart.NumberAxis;
```

```
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;


public class BubbleChartSample extends Application {

    @Override public void start(Stage stage) {
        stage.setTitle("Bubble Chart Sample");
        final NumberAxis xAxis = new NumberAxis(1, 53, 4);
        final NumberAxis yAxis = new NumberAxis(0, 80, 10);
        final BubbleChart<Number,Number> blc = new
            BubbleChart<>(xAxis,yAxis);
        xAxis.setLabel("Week");
        yAxis.setLabel("Product Budget");
        blc.setTitle("Budget Monitoring");

        XYChart.Series series1 = new XYChart.Series();
        series1.setName("Product 1");
        series1.getData().add(new XYChart.Data(3, 35));
        series1.getData().add(new XYChart.Data(12, 60));
        series1.getData().add(new XYChart.Data(15, 15));
        series1.getData().add(new XYChart.Data(22, 30));
        series1.getData().add(new XYChart.Data(28, 20));
        series1.getData().add(new XYChart.Data(35, 41));
        series1.getData().add(new XYChart.Data(42, 17));
        series1.getData().add(new XYChart.Data(49, 30));

        XYChart.Series series2 = new XYChart.Series();
        series2.setName("Product 2");
        series2.getData().add(new XYChart.Data(8, 15));
        series2.getData().add(new XYChart.Data(13, 23));
        series2.getData().add(new XYChart.Data(15, 45));
        series2.getData().add(new XYChart.Data(24, 30));
        series2.getData().add(new XYChart.Data(38, 78));
        series2.getData().add(new XYChart.Data(40, 41));
        series2.getData().add(new XYChart.Data(45, 57));
        series2.getData().add(new XYChart.Data(47, 23));

        Scene scene  = new Scene(blc);
        blc.getData().addAll(series1, series2);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

The result of compiling and running this application is shown in Figure 34–2.

Figure 34–2 Bubble Chart with Two Data Parameters



This application defines lower and upper boundaries of the data ranges and the tick units within the constructors of the `NumberAxis` class. Additionally, the minor tick count is set to 4, so that each minor tick corresponds to a particular week.

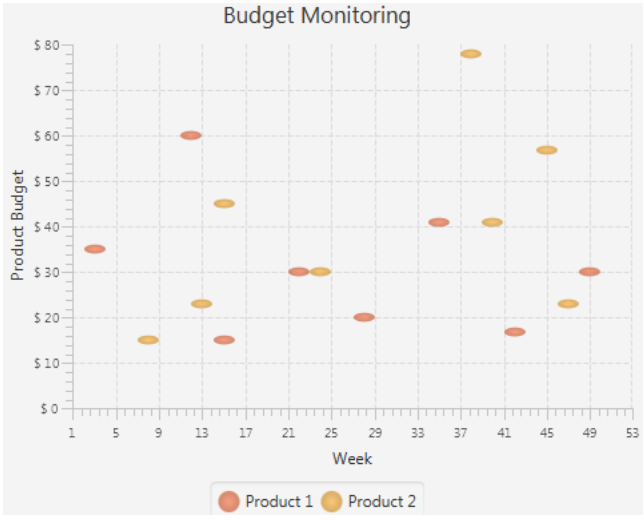
To indicate that the vertical axis renders the amount of money in US dollars, use a tick label formatter as shown in Example 34–2.

Example 34–2 Formatting Tick Labels

```
yAxis.setTickLabelFormatter(new NumberAxis.DefaultFormatter(yAxis,"$ ",null));
```

The `NumberAxis.DefaultFormatter` class adds prefixes and suffixes to the tick labels of the specified axis. In Example 34–2, the formatter defines a dollar sign ($) prefix for each tick label of the vertical axis. The `null` value for the suffix parameter indicates that no suffixes are added. Figure 34–3 shows the bubble chart after the formatting has been applied.

Figure 34–3 Specifying Prefixes for Tick Labels



Using the Extra Value Property

The bubble chart shown in Figure 34–1 provides information about budgets of two products for the period of a year. However, you can enhance this application and benefit from the additional capabilities of the `BubbleChart` class. Use the `extraValue` property and define three parameters in `XYChart.Data` objects when specifying the series of data for your bubble chart.

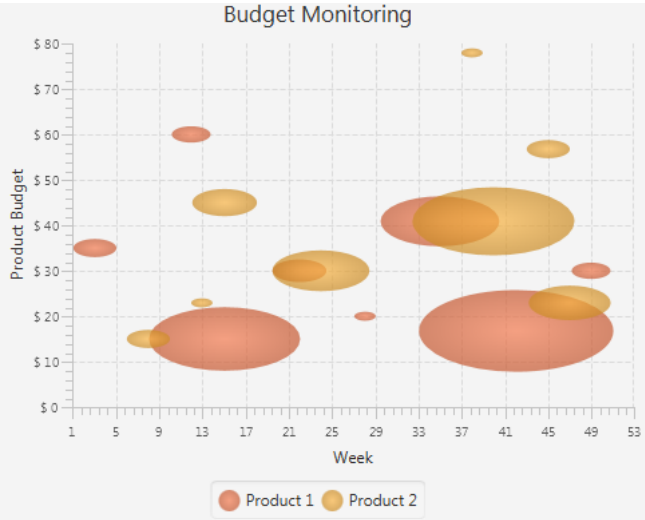
The code fragment shown in Example 34–3 demonstrates how to modify data series for the Budget Monitoring application, so that each bubble shows the percentage of consumed budget for a particular product. The third parameter in the `XYChart.Data` object defines the radius of each bubble: the bigger the radius, the higher the percentage of the budget consumed. Thus, a radius of 7.5 corresponds to consuming 75% of the budget, 5.5 to 55%, and so on.

Example 34–3 Adding Extra Value

```
XYChart.Series series1 = new XYChart.Series();
series1.setName("Product 1");
series1.getData().add(new XYChart.Data(3, 35, 2));
series1.getData().add(new XYChart.Data(12, 60, 1.8));
series1.getData().add(new XYChart.Data(15, 15, 7));
series1.getData().add(new XYChart.Data(22, 30, 2.5));
series1.getData().add(new XYChart.Data(28, 20, 1));
series1.getData().add(new XYChart.Data(35, 41, 5.5));
series1.getData().add(new XYChart.Data(42, 17, 9));
series1.getData().add(new XYChart.Data(49, 30, 1.8));

XYChart.Series series2 = new XYChart.Series();
series2.setName("Product 2");
series2.getData().add(new XYChart.Data(8, 15, 2));
series2.getData().add(new XYChart.Data(13, 23, 1));
series2.getData().add(new XYChart.Data(15, 45, 3));
series2.getData().add(new XYChart.Data(24, 30, 4.5));
series2.getData().add(new XYChart.Data(38, 78, 1));
series2.getData().add(new XYChart.Data(40, 41, 7.5));
series2.getData().add(new XYChart.Data(45, 57, 2));
series2.getData().add(new XYChart.Data(47, 23, 3.8));
```

The result of adding the modified code fragment to the Budget Monitoring application and then compiling and running it is shown in Figure 34–4.

Figure 34–4 Demonstrating Percentage of Consumed Budget



Changing the Appearance Visual Setting of the Plot and Tick Marks

You can alter the appearance of the chart plot and axes. Examine some methods and properties available in the JavaFX SDK API to modify the look of the Budget Monitoring application.

The `verticalGridLinesVisible` and `horizontalGridLinesVisible` properties enable and disable showing the grid lines.

In addition to the plot properties, you can modify the properties of the axis to attain the required appearance. Code lines in Example 34–4 specify the `CHOCOLATE` color for the tick labels.

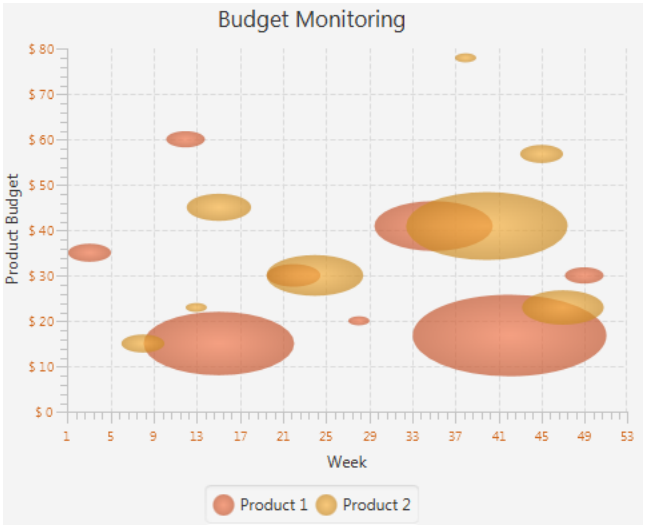
Example 34–4 Defining a Fill Color for the Tick Labels

```
xAxis.setTickLabelFill(Color.CHOCOLATE);
yAxis.setTickLabelFill(Color.CHOCOLATE);
```

Finally, you can adjust the position of the tick labels relative to the tick marks. Use the `setTickLabelGap` method to specify the gap between tick labels and the tick mark lines on the vertical axis: `yAxis.setTickLabelGap(10)`.

Figure 34–5 shows the output of the Budget Monitoring application after all the modifications are incorporated in the code.

Figure 34–5 Changes in the Appearance of the Bubble Chart



Related API Documentation

- `BubbleChart`
- `Chart`
- `XYChart`
- `XYChart.Data`
- `XYChart.Series`
- `NumberAxis`
- `NumberAxis.DefaultFormatter`

35

Scatter Chart

This chapter describes the scatter chart, a two-axis chart that presents its data as a set of points.

Each point is defined by an X and Y value. Similar to any other two-axis chart, you can create one or several series of data. Figure 35–1 illustrates a scatter chart with three series of data.

Figure 35–1 Sample of the Scatter Chart

Creating a Scatter Chart

To create a scatter chart, define at least one series of data, set horizontal and vertical axes, create the chart by instantiating the `ScatterChart` class, and assign data to the chart. Example 35–1 demonstrates how to create a simple scatter charts with two series of data.

Example 35–1 Scatter Chart with Two Series of Data

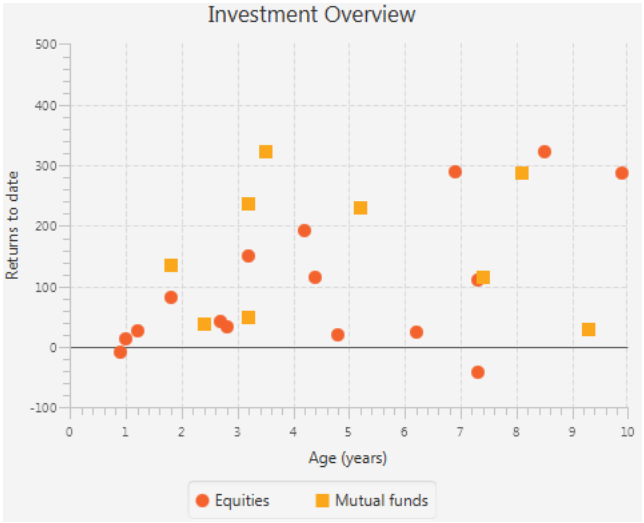
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.ScatterChart;
```

```
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;


public class ScatterChartSample extends Application {

    @Override public void start(Stage stage) {
        stage.setTitle("Scatter Chart Sample");
        final NumberAxis xAxis = new NumberAxis(0, 10, 1);
        final NumberAxis yAxis = new NumberAxis(-100, 500, 100);
        final ScatterChart<Number,Number> sc = new
            ScatterChart<>(xAxis,yAxis);
        xAxis.setLabel("Age (years)");
        yAxis.setLabel("Returns to date");
        sc.setTitle("Investment Overview");

        XYChart.Series series1 = new XYChart.Series();
        series1.setName("Equities");
        series1.getData().add(new XYChart.Data(4.2, 193.2));
        series1.getData().add(new XYChart.Data(2.8, 33.6));
        series1.getData().add(new XYChart.Data(6.2, 24.8));
        series1.getData().add(new XYChart.Data(1, 14));
        series1.getData().add(new XYChart.Data(1.2, 26.4));
        series1.getData().add(new XYChart.Data(4.4, 114.4));
        series1.getData().add(new XYChart.Data(8.5, 323));
        series1.getData().add(new XYChart.Data(6.9, 289.8));
        series1.getData().add(new XYChart.Data(9.9, 287.1));
        series1.getData().add(new XYChart.Data(0.9, -9));
        series1.getData().add(new XYChart.Data(3.2, 150.8));
        series1.getData().add(new XYChart.Data(4.8, 20.8));
        series1.getData().add(new XYChart.Data(7.3, -42.3));
        series1.getData().add(new XYChart.Data(1.8, 81.4));
        series1.getData().add(new XYChart.Data(7.3, 110.3));
        series1.getData().add(new XYChart.Data(2.7, 41.2));

        XYChart.Series series2 = new XYChart.Series();
        series2.setName("Mutual funds");
        series2.getData().add(new XYChart.Data(5.2, 229.2));
        series2.getData().add(new XYChart.Data(2.4, 37.6));
        series2.getData().add(new XYChart.Data(3.2, 49.8));
        series2.getData().add(new XYChart.Data(1.8, 134));
        series2.getData().add(new XYChart.Data(3.2, 236.2));
        series2.getData().add(new XYChart.Data(7.4, 114.1));
        series2.getData().add(new XYChart.Data(3.5, 323));
        series2.getData().add(new XYChart.Data(9.3, 29.9));
        series2.getData().add(new XYChart.Data(8.1, 287.4));

        sc.getData().addAll(series1, series2);
        Scene scene  = new Scene(sc, 500, 400);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

In this example, the `ScatterChart` object is created with two `Number` axes to present numerical data for years and amounts of returns. The range of the data and the tick unit are defined within constructors of the `NumberAxis` class.

The result of compiling and running this application is shown in Figure 35–2.

Figure 35–2 Scatter Chart with Two Series to Display Investment Overview



Managing Chart Data

Example 35–1 creates a scatter chart whose data is coded into the application and cannot be changed from its user interface. Use UI controls in your application to manage the set of data presented by the chart, for example, adding and removing a series of data.

Examine the code shown in Example 35–2. It creates two buttons, Add Series and Remove Series, to alter the set of data.

Example 35–2 Using Buttons to Manager Chart Data

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.ScatterChart;
import javafx.scene.chart.XYChart;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;


public class ScatterChartSample extends Application {

    @Override public void start(Stage stage) {
        stage.setTitle("Scatter Chart Sample");
        final NumberAxis xAxis = new NumberAxis(0, 10, 1);
        final NumberAxis yAxis = new NumberAxis(-100, 500, 100);
        final ScatterChart<Number,Number> sc =
```

```
            new ScatterChart<>(xAxis,yAxis);
        xAxis.setLabel("Age (years)");
        yAxis.setLabel("Returns to date");
        sc.setTitle("Investment Overview");

        XYChart.Series series1 = new XYChart.Series();

        series1.setName("Option 1");
        series1.getData().add(new XYChart.Data(4.2, 193.2));
        series1.getData().add(new XYChart.Data(2.8, 33.6));
        series1.getData().add(new XYChart.Data(6.2, 24.8));
        series1.getData().add(new XYChart.Data(1, 14));
        series1.getData().add(new XYChart.Data(1.2, 26.4));
        series1.getData().add(new XYChart.Data(4.4, 114.4));
        series1.getData().add(new XYChart.Data(8.5, 323));
        series1.getData().add(new XYChart.Data(6.9, 289.8));
        series1.getData().add(new XYChart.Data(9.9, 287.1));
        series1.getData().add(new XYChart.Data(0.9, -9));
        series1.getData().add(new XYChart.Data(3.2, 150.8));
        series1.getData().add(new XYChart.Data(4.8, 20.8));
        series1.getData().add(new XYChart.Data(7.3, -42.3));
        series1.getData().add(new XYChart.Data(1.8, 81.4));
        series1.getData().add(new XYChart.Data(7.3, 110.3));
        series1.getData().add(new XYChart.Data(2.7, 41.2));

        sc.setPrefSize(500, 400);
        sc.getData().addAll(series1);
        Scene scene  = new Scene(new Group());
        final VBox vbox = new VBox();
        final HBox hbox = new HBox();

        final Button add = new Button("Add Series");
        final Button remove = new Button("Remove Series");

        hbox.setSpacing(10);
        hbox.getChildren().addAll(add, remove);

        vbox.getChildren().addAll(sc, hbox);
        hbox.setPadding(new Insets(10, 10, 10, 50));

        ((Group)scene.getRoot()).getChildren().add(vbox);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Whereas Example 35–1 adds the scatter chart directly to the scene, Example 35–2 uses `VBox` and `HBox` layout containers to arrange components in the application scene.

Define the `setOnAction` methods for the Add Series button as shown in Example 35–3. It creates a new series of data by populating the `XYChart.Series` objects with randomly calculated values. Each new series is assigned to the chart by using the `add(series)` method.

Example 35–3 Adding Series of Data

```
        add.setOnAction((ActionEvent e) -> {
```

```
        if (sc.getData() == null) {
            sc.setData(FXCollections.<XYChart.Series<Number,
            Number>>observableArrayList());
        }
        ScatterChart.Series<Number, Number> series
                = new ScatterChart.Series<>();
        series.setName("Option " + (sc.getData().size() + 1));
        for (int i = 0; i < 100; i++) {
            series.getData().add(
                    new ScatterChart.Data<>(Math.random() * 100,
                            Math.random() * 500));
        }
        sc.getData().add(series);
    });
```

To remove a data series from the chart, define the `setOnAction` method for the Remove Series button as shown in Example 35–4. The `remove(int)` method called on the scatter chart removes a series of data by using a randomly generated index.

Example 35–4 Removing Series of Data

```
    remove.setOnAction((ActionEvent e) -> {
        if (!sc.getData().isEmpty())
            sc.getData().remove((int)(
                Math.random()*(sc.getData().size()-1)));
    });
```

When you add Example 35–3 and Example 35–4 into the application in Example 35–2, the output shown in Figure 35–3 appears. It captures the moment when five series are added to the Option 1 series.

Figure 35–3 Added Series of Data

The symbols used to indicate a series of data are coded into the implementation of the `ScatterChart` class. Example 35–5 shows the default styles for one of the scatter chart symbols.

Example 35–5 Styling a ScatterChart Symbol

```
.default-color5.chart-symbol { /* hollow circle */
    -fx-background-color: #860061, white;
    -fx-background-insets: 0, 2;
    -fx-background-radius: 5px;
    -fx-padding: 5px;
}
```

You can change the styles for this symbol by setting the alternative values for the `.default-color5.chart-symbol` property. See Styling Charts with CSS for more information.

Adding Effects to Charts

All the chart classes available in the `javafx.scene.chart` are extensions of the `Node` class. Therefore, you can apply visual effects or transformation to every type of charts. Examine the code fragment in Example 35–6. It creates and applies a drop shadow effect to the scatter chart.

Example 35–6 Creating and Applying a Drop Shadow

```
final DropShadow shadow = new DropShadow();
shadow.setOffsetX(2);
shadow.setColor(Color.GREY);
sc.setEffect(shadow);
```

When you add this code fragment to the Investment Overview application, then compile and run it, the scatter chart is highlighted by the shadow as shown in Figure 35–4.

Figure 35–4 Scatter Chart with a Drop Shadow

Note that the visual effect of the drop shadow is applied to all elements of the chart including axes, tick marks, and tick labels.

Changing the Chart Symbol

Each data series in a scatter chart is represented by the symbols defined in the `modena.css`, the default style sheet for JavaFX applications. However, you can change the chart symbol by implementing your own style sheet.

Create the `Chart.css` file and save it in the same directory as the main class of the `AreaChartSample` application. Add the lines shown in Example 35–7 to the `Chart.css` file.

Example 35–7 Creating a New Chart Symbol with CSS

```
.chart-symbol{
    -fx-stroke: #a9e200;
    -fx-shape: "M0,4 L2,4 L4,8 L7,0 L9,0 L4,11 Z";
}
```

This code fragment creates the symbol shape by defining its SVG path in the `-fx-shape` parameter and sets the stroke color for the symbol.

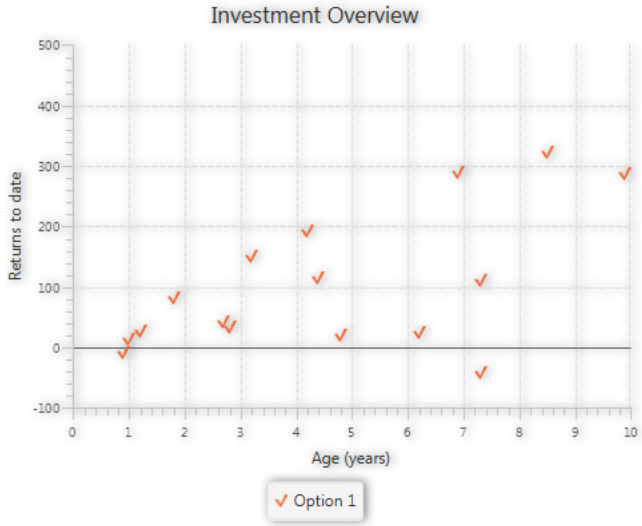
Use the `getStylesheets()` method of the `Scene` class to apply the style sheet to the application, as shown in Example 35–8.

Example 35–8 Applying a CSS Style to the Scene

```
scene.getStylesheets().add("scaterchartsample/Chart.css");
```

Compiling and running this application produces the modified appearance of the area chart shown in Figure 35–5.

Figure 35–5 Scatter Chart with the Modified Chart Symbol



You can learn more about using CSS styles in JavaFX applications from Styling Charts with CSS and Styling UI Controls with CSS.

Related API Documentation

- `ScatterChart`
- `Chart`

- `XYChart`
- `XYChart.Data`
- `XYChart.Series`
- `NumberAxis`
- `Button`

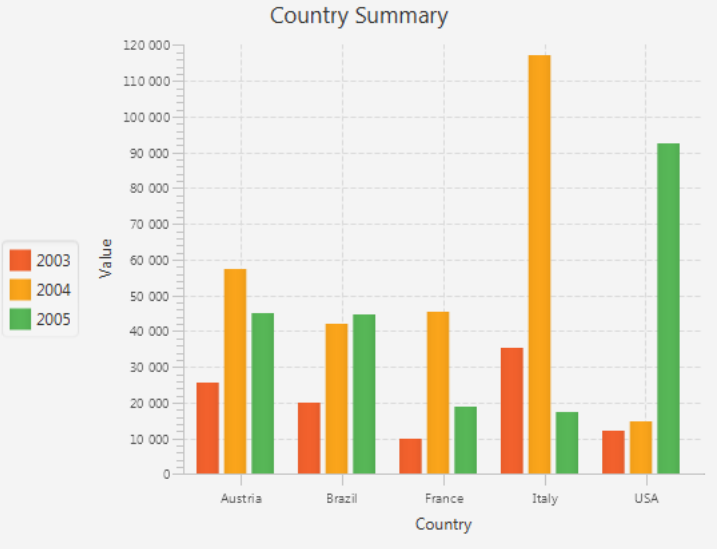
36

Bar Chart

This chapter describes the bar chart, a two-axis chart with rectangular bars that can be either vertical or horizontal.

The length of each bar is proportional to a particular value that the chart presents. Typically, bar charts are used to display discrete data. You can use groups of bars as categories to plot data, as shown in Figure 36–1.

Figure 36–1 Sample Bar Chart



Creating a Bar Chart

To build a bar chart in your JavaFX application, create two axes, instantiate the `BarChar` class, define the series of data, and assign the data to the chart. Example 36–1 creates a bar chart with three series of data to present financial information about five countries. Each country is presented as a category that is a group of bars on the horizontal axis.

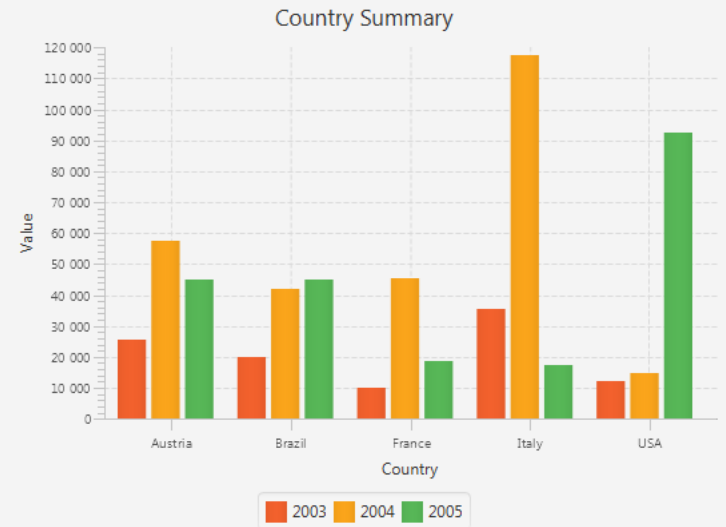
Example 36–1 Creating a Bar Chart with Three Series of Data

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.NumberAxis;
```

```
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;

public class BarChartSample extends Application {
    final static String austria = "Austria";
    final static String brazil = "Brazil";
    final static String france = "France";
    final static String italy = "Italy";
    final static String usa = "USA";

    @Override public void start(Stage stage) {
        stage.setTitle("Bar Chart Sample");
        final CategoryAxis xAxis = new CategoryAxis();
        final NumberAxis yAxis = new NumberAxis();
        final BarChart<String,Number> bc =
            new BarChart<>(xAxis,yAxis);
        bc.setTitle("Country Summary");
        xAxis.setLabel("Country");
        yAxis.setLabel("Value");

        XYChart.Series series1 = new XYChart.Series();
        series1.setName("2003");
        series1.getData().add(new XYChart.Data(austria, 25601.34));
        series1.getData().add(new XYChart.Data(brazil, 20148.82));
        series1.getData().add(new XYChart.Data(france, 10000));
        series1.getData().add(new XYChart.Data(italy, 35407.15));
        series1.getData().add(new XYChart.Data(usa, 12000));

        XYChart.Series series2 = new XYChart.Series();
        series2.setName("2004");
        series2.getData().add(new XYChart.Data(austria, 57401.85));
        series2.getData().add(new XYChart.Data(brazil, 41941.19));
        series2.getData().add(new XYChart.Data(france, 45263.37));
        series2.getData().add(new XYChart.Data(italy, 117320.16));
        series2.getData().add(new XYChart.Data(usa, 14845.27));

        XYChart.Series series3 = new XYChart.Series();
        series3.setName("2005");
        series3.getData().add(new XYChart.Data(austria, 45000.65));
        series3.getData().add(new XYChart.Data(brazil, 44835.76));
        series3.getData().add(new XYChart.Data(france, 18722.18));
        series3.getData().add(new XYChart.Data(italy, 17557.31));
        series3.getData().add(new XYChart.Data(usa, 92633.68));

        Scene scene  = new Scene(bc,800,600);
        bc.getData().addAll(series1, series2, series3);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Figure 36–2 shows the expected output of this application when you compile and run it.

Figure 36–2 Creating a Bar Chart with Three Series of Data



Two properties of the `BarChart` class enable managing space between categories of data and between bars within the same category. Use the `barGap` and `categoryGap` properties to better distribute bars in the chart plot. Example 36–2 uses the `setBarGap` and `setCategoryGap` methods to set specific values for these properties.

Example 36–2 Setting Gaps Between Bars and Categories

```
bc.setBarGap(3);
bc.setCategoryGap(20);
```

Horizontal Bar Chart

You can change the orientation of the bar chart from vertical to horizontal by defining the category for the vertical axis. Example 36–3 implements this for the Country Summary application. Declare the horizontal axis of the `NumberAxis` type and the vertical axis of the `CategoryAxis` type. Do not forget to modify the declaration of the `BarChart` object.

Example 36–3 Changing Orientation of the Bar Chart

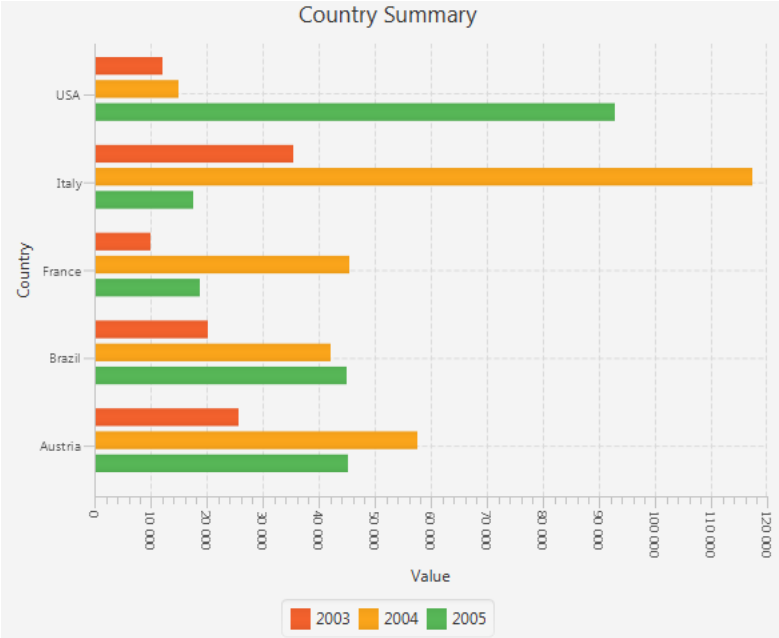
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;

public class BarChartSample extends Application {
    final static String austria = "Austria";
    final static String brazil = "Brazil";
    final static String france = "France";
    final static String italy = "Italy";
    final static String usa = "USA";

    @Override public void start(Stage stage) {
```

```
        stage.setTitle("Bar Chart Sample");
        final NumberAxis xAxis = new NumberAxis();
        final CategoryAxis yAxis = new CategoryAxis();
        final BarChart<Number,String> bc =
            new BarChart<>(xAxis,yAxis);
        bc.setTitle("Country Summary");
        xAxis.setLabel("Value");
        xAxis.setTickLabelRotation(90);
        yAxis.setLabel("Country");

        XYChart.Series series1 = new XYChart.Series();
        series1.setName("2003");
        series1.getData().add(new XYChart.Data(25601.34, austria));
        series1.getData().add(new XYChart.Data(20148.82, brazil));
        series1.getData().add(new XYChart.Data(10000, france));
        series1.getData().add(new XYChart.Data(35407.15, italy));
        series1.getData().add(new XYChart.Data(12000, usa));

        XYChart.Series series2 = new XYChart.Series();
        series2.setName("2004");
        series2.getData().add(new XYChart.Data(57401.85, austria));
        series2.getData().add(new XYChart.Data(41941.19, brazil));
        series2.getData().add(new XYChart.Data(45263.37, france));
        series2.getData().add(new XYChart.Data(117320.16, italy));
        series2.getData().add(new XYChart.Data(14845.27, usa));

        XYChart.Series series3 = new XYChart.Series();
        series3.setName("2005");
        series3.getData().add(new XYChart.Data(45000.65, austria));
        series3.getData().add(new XYChart.Data(44835.76, brazil));
        series3.getData().add(new XYChart.Data(18722.18, france));
        series3.getData().add(new XYChart.Data(17557.31, italy));
        series3.getData().add(new XYChart.Data(92633.68, usa));

        Scene scene  = new Scene(bc,800,600);
        bc.getData().addAll(series1, series2, series3);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Note that the `setTickLabelRotation` method is called on the horizontal axis to rotate labels and make the value captions easier to read.

The result of compiling and running the modified application is shown in Figure 36–3.

Figure 36–3 Horizontal Bar Chart



Horizontal bar charts can be particularly helpful when you want to represent data as ranked lists.

Creating a Stacked Bar Chart

You can represent data in a bar chart so that the bars in a category are stacked. Use the `StackedBarChart` class available in the JavaFX API, as shown in Example 36–4.

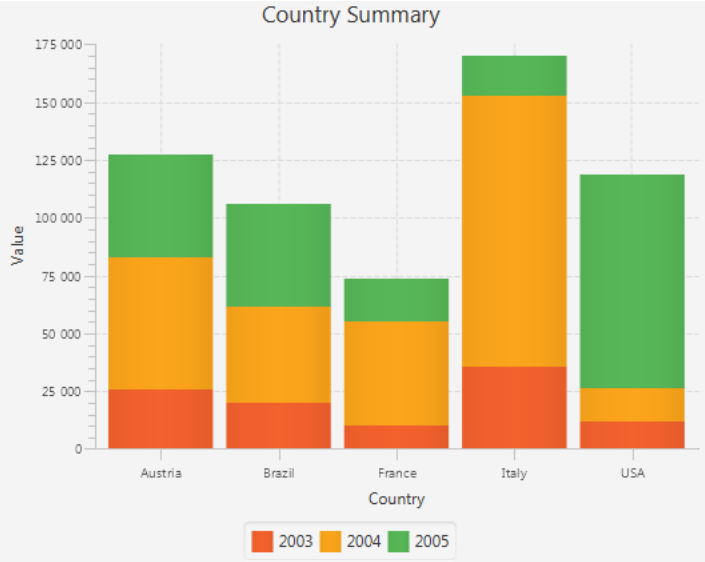
Example 36–4 Creating a Stacked Bar Chart

```
import java.util.Arrays;
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.scene.Scene;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.StackedBarChart;
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;

public class StackedBarChartSample extends Application {

    final static String austria = "Austria";
    final static String brazil = "Brazil";
    final static String france = "France";
    final static String italy = "Italy";
    final static String usa = "USA";
    final CategoryAxis xAxis = new CategoryAxis();
    final NumberAxis yAxis = new NumberAxis();
    final StackedBarChart<String, Number> sbc =
            new StackedBarChart<>(xAxis, yAxis);
    final XYChart.Series<String, Number> series1 =
            new XYChart.Series<>();
```

```
    final XYChart.Series<String, Number> series2 =
            new XYChart.Series<>();
    final XYChart.Series<String, Number> series3 =
            new XYChart.Series<>();

    @Override
    public void start(Stage stage) {
        stage.setTitle("Bar Chart Sample");
        sbc.setTitle("Country Summary");
        xAxis.setLabel("Country");
        xAxis.setCategories(FXCollections.<String>observableArrayList(
                Arrays.asList(austria, brazil, france, italy, usa)));
        yAxis.setLabel("Value");
        series1.setName("2003");
        series1.getData().add(new XYChart.Data<>(austria, 25601.34));
        series1.getData().add(new XYChart.Data<>(brazil, 20148.82));
        series1.getData().add(new XYChart.Data<>(france, 10000));
        series1.getData().add(new XYChart.Data<>(italy, 35407.15));
        series1.getData().add(new XYChart.Data<>(usa, 12000));
        series2.setName("2004");
        series2.getData().add(new XYChart.Data<>(austria, 57401.85));
        series2.getData().add(new XYChart.Data<>(brazil, 41941.19));
        series2.getData().add(new XYChart.Data<>(france, 45263.37));
        series2.getData().add(new XYChart.Data<>(italy, 117320.16));
        series2.getData().add(new XYChart.Data<>(usa, 14845.27));
        series3.setName("2005");
        series3.getData().add(new XYChart.Data<>(austria, 45000.65));
        series3.getData().add(new XYChart.Data<>(brazil, 44835.76));
        series3.getData().add(new XYChart.Data<>(france, 18722.18));
        series3.getData().add(new XYChart.Data<>(italy, 17557.31));
        series3.getData().add(new XYChart.Data<>(usa, 92633.68));
        Scene scene = new Scene(sbc, 800, 600);
        sbc.getData().addAll(series1, series2, series3);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

When you define axes for a stacked bar chart in your application, you must explicitly assign the categories of data to a particular axis. In Example 36–4, the categories are assigned to the horizontal axis by using the `setCategories` method. The bar chart produced by this application is shown in Figure 36–4.

Figure 36–4 Stacked Bar Chart with Five Categories of Data



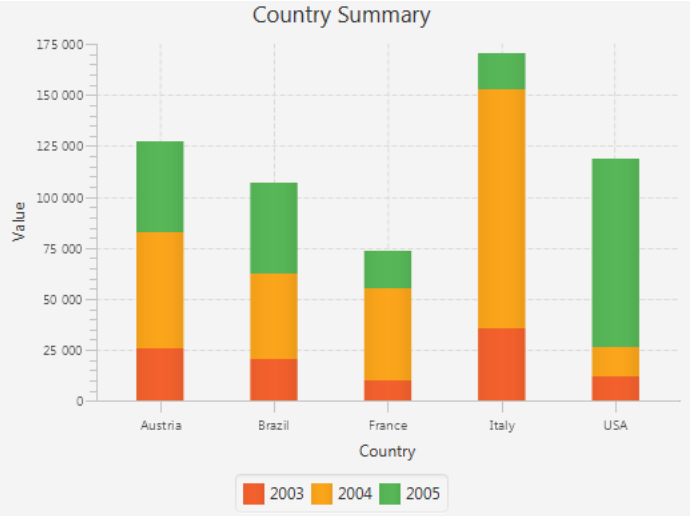
Compare the data shown in Figure 36–3 with the same data in Figure 36–4. The areas in the stacked bar chart show cumulative values on the vertical axis at any given point along the horizontal axis. For example, on the vertical axis of Figure 36–4, the 2004 values for Austria range from approximately 25,000 to approximately 85,000. However, the data in Example 36–4 indicates a value of 57,401.85 for Austria in 2004. The 2004 high value of approximately 85,000 in Figure 36–4 represents the cumulative results for Austria in 2003 and 2004.

When you develop bar charts in your JavaFX application, remember that data on the vertical axes are interpreted differently for a `BarChart` than for a `StackedBarChart`. Choose the type of chart that best illustrates the task of the application.

You can specify the distance between the stacked categories by setting the value in the `setCategoryGap` method. For example, you can set the distance of 50 pixels for the Country Summary bar chart: `sbc.setCategoryGap(50);`

When you apply this method to the stacked bar chart in Example 36–4, the bar categories look as shown in Figure 36–5.

Figure 36–5 Stacked Bar Chart with the Specified Gap Between the Categories



Animating Data in Charts

You can implement animated charts to illustrate dynamic behavior of financial activities. Example 36–5 defines an animation timeline and creates key frames to randomly set the X value for the data of the bar chart. The timeline starts when the application does and continues indefinitely in the auto-reverse mode.

Example 36–5 Animating Data in a Bar Chart

```
Timeline tl = new Timeline();
tl.getKeyFrames().add(new KeyFrame(Duration.millis(500),
    (ActionEvent actionEvent) -> {
        bc.getData().stream().forEach((series) -> {
            series.getData().stream().forEach((data) -> {
                data.setXValue(Math.random() * 1000);
            });
        });
    }
));
tl.setCycleCount(Animation.INDEFINITE);
tl.setAutoReverse(true);
tl.play();
```

When you add this code fragment to the Country Summary application in Example 36–3, and then compile and run the modified code, you will notice that both the axis and the chart plot change smoothly to accommodate new values in ranges and new lengths of the bars. This is because of the `animated` properties of the `Chart` and `Axis` classes. By default, they set to `true` to animate any data changes.

For the Country Summary application, you can prohibit animating data along the vertical axis when the data on this axis is presented in categories and does not change. To avoid undesirable flickering of the country labels, use the `setAnimated` method as shown in Example 36–6.

Example 36–6 Managing Animation of Data Changes

```
yAxis.setAnimated(false);
```

See the API documentation for more information about the features and capabilities of JavaFX charts.

Related API Documentation

- `BarChart`
- `Chart`
- `XYChart`
- `XYChart.Data`
- `XYChart.Series`
- `Axis`
- `NumberAxis`
- `CategoryAxis`
- `Timeline`
- `KeyFrame`

Part IV

Skinning JavaFX Applications with CSS

This part provides general information about creating cascading style sheets (CSS) styles and applying them to the user interface (UI) components of your JavaFX applications. It contains the following chapters:

- Styling UI Controls with CSS

  Describes how to use CSS with JavaFX applications and create a custom look for its UI controls.

- Styling Charts with CSS

  Explains how to change a chart color scheme, modify its legend or axes, and alter chart symbols.

37

Styling UI Controls with CSS

This topic describes how to use cascading style sheets (CSS) with JavaFX and create a custom look for your application.

Style sheets contain style definitions that control the look of user interface elements. Using CSS in JavaFX applications is similar to using CSS in HTML. JavaFX CSS are based on the W3C CSS version 2.1 specification (available at `http://www.w3.org/TR/CSS21/`) with some additions from current work on version 3 of the specification and some extensions that support specific JavaFX features.

Skinning your UI with JavaFX CSS enables you to change the UI shown in Figure 37–1 to the UI shown in Figure 37–2 just by changing the style sheet used.

Figure 37–1 Style 1

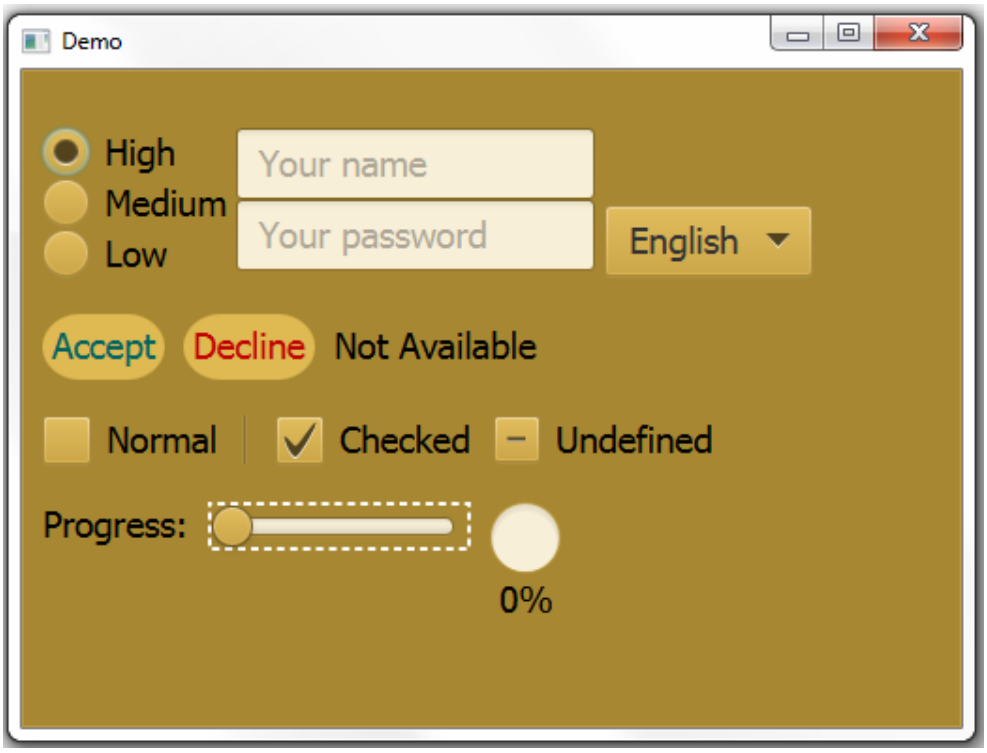
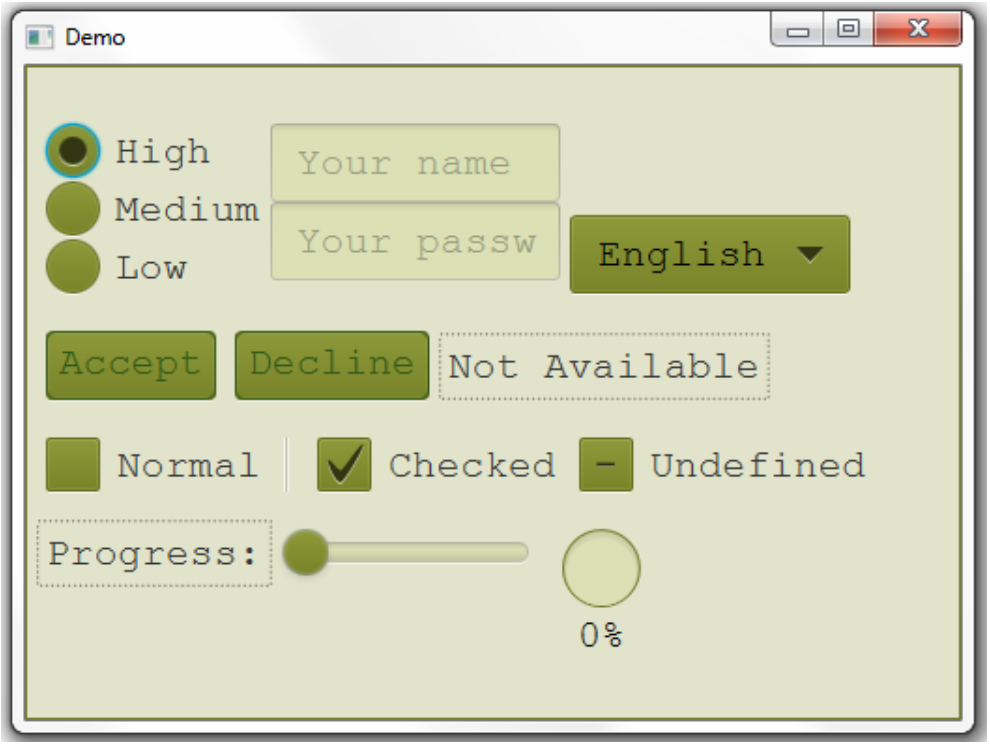


Figure 37–2 Style 2



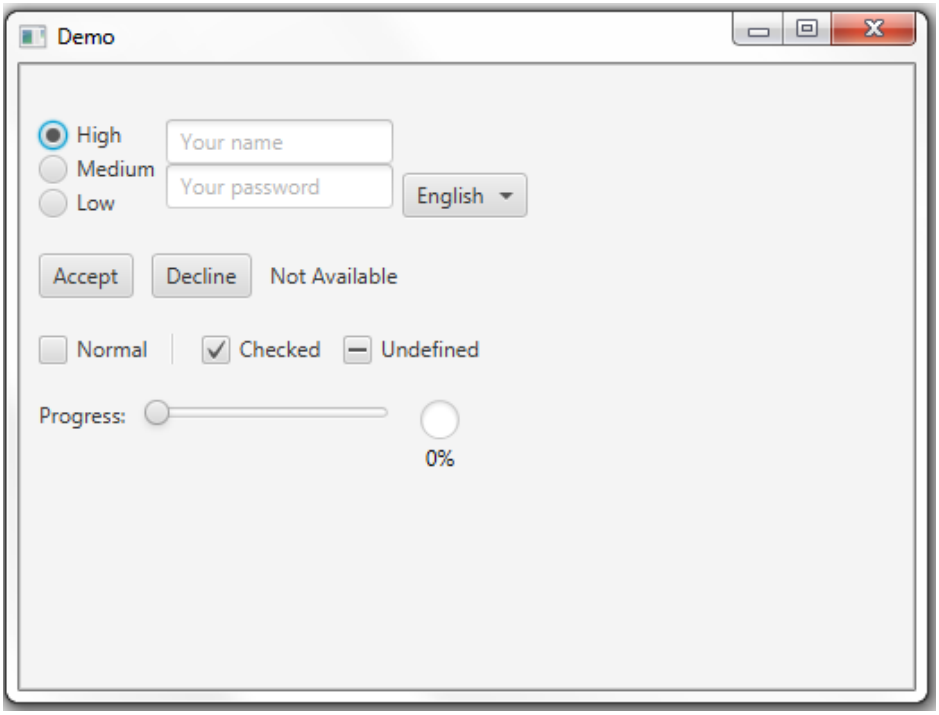
Default Style Sheet

The default style sheet for JavaFX applications is modena.css, which is found in the JavaFX runtime JAR file, jfxrt.jar. This style sheet defines styles for the root node and the UI controls. To view this file, go to the `\jre\lib\ext` directory under the directory in which the Java Development Kit (JDK) is installed. Use the following command to extract the style sheet from the JAR file:

```
jar xf jfxrt.jar com/sun/javafx/scene/control/skin/modena/modena.css
```

Figure 37–3 shows what the sample UI looks like with the default style sheet.

Figure 37–3 Default Style



Creating Style Sheets

You can create one or more of your own style sheets to override the styles in the default style sheet and to add your own styles. Typically style sheets that you create have an extension of `.css` and are located in the same directory as the main class for your JavaFX application.

The style sheet controlStyle1.css provides the skinning shown in Figure 37–1. The style sheet controlStyle2.css provides the skinning shown in Figure 37–2.

Style sheets are applied to `Scene` objects as shown in Example 37–1, where *path* is the directory structure that reflects the location of your style sheet, and *stylesheet* is the name of your style sheet. For example, the path and name of the style sheet for Figure 37–2 is uicontrolcss/controlStyle2.css.

Example 37–1 Adding a Style Sheet

```
Scene scene = new Scene(new Group(), 500, 400);
scene.getStylesheets().add("path/stylesheet.css");
```

Defining Styles

A style definition consists of the name of the style, also called the selector, and a series of rules that set the properties for the style. Rules for a definition are enclosed in braces ({}). Example 37–2 shows the definition for a style named `.custom-button`.

Example 37–2 Sample Style Definition

```
.custom-button {
    -fx-font: 16px "Serif";
```

```
    -fx-padding: 10;
    -fx-background-color: #CCFF99;
}
```

> **Note:** The size of a font can be specified in either points (pt) or pixels (px). A resolution of 96 dots per inch (dpi) is assumed, so 1px = 0.75pt.

Selectors

Several types of styles can be defined. Each type of style has its own convention for selectors.

Style classes correspond to class names. By convention, style class names that consist of more than one word use a hyphen (-) between words. Style class selectors are preceded by a dot (.).

Examples of class selectors:

```
.button
.check-box
.scroll-bar
```

You can also define styles that are associated with a node through the node's ID. The ID is set using the node's `setId()` method. The style name is the ID preceded by a hash symbol (#). For example, a node with the ID `my-button` is skinned with the style `#my-button`.

Examples of ID style selectors:

```
#my-button
#shaded-hbox
```

Compound selectors are also possible. Some classes include elements that can have their own style definition, which are called descendant classes. For example, many UI controls have a descendant class for the label. These definitions are identified by the selector for the class and the selector for the descendant class separated by a space.

Examples of selectors for descendant classes:

```
.check-box .label
.check-box .box
.radio-button .dot
```

Pseudo-classes enable you to customize states of a node, such as when a node has focus. These definitions are identified by the selector for the class and the name for the state separated by a colon (:).

Examples of selectors for pseudo-classes:

```
.radio-button:focused
.radio-button:hover
.scroll-bar:vertical
```

Rules and Properties

The rules for a style definition assign values to properties associated with the class. Rule property names correspond to the names of the properties for a class. The convention for property names with multiple words is to separate the words with a hyphen (-). Property names for styles in JavaFX style sheets are preceded by `-fx-`.

Property names and values are separated by a colon (:). Rules are terminated with a semicolon (;).

Examples of rules:

```
-fx-background-color: #333333;
-fx-text-fill: white;
-fx-alignment: CENTER;
```

The `.root` style class is applied to the root node of the `Scene` instance. Because all nodes in the scene graph are a descendant of the root node, styles in the `.root` style class can be applied to any node.

The `.root` style class includes properties that can be used by other styles to provide consistency in a UI. For example, the property `-fx-focused-base` is defined in the `.root` style. This property is used by styles for other UI controls as the color for the control when it has focus. The following definition shows how this property is used in the style for the class `CheckBox`:

```
.check-box:focused {
    -fx-color: -fx-focused-base;
}
```

Skinning the Scene

You can quickly change the look of your UI just by customizing the `.root` style class. Both of the sample style sheets set the font size and family, the base color from which other colors are derived, and the background color of the scene. Example 37–3 shows the `.root` style from controlStyle2.css.

Example 37–3 Root Style from controlStyle2.css

```
.root{
    -fx-font-size: 16pt;
    -fx-font-family: "Courier New";
    -fx-base: rgb(132, 145, 47);
    -fx-background: rgb(225, 228, 203);
}
```

With just this style, you create the basic look of Figure 37–2. This is possible because the built-in UI controls use the properties set for the root node to derive their own colors and fonts.

Skinning Controls

You can further customize your UI by defining styles for the different controls that you are using. You can override the definitions in the default style sheet or create new class or ID styles. You can also define the style for a node within your code.

Overriding Default Styles

You can override a style in the default style sheet by including the style in your style sheet and assigning it different properties. Example 37–4 shows the style for the `Button` class from controlStyle2.css.

Example 37–4 Override a Style

```
.button{
```

```
    -fx-text-fill: rgb(49, 89, 23);
    -fx-border-color: rgb(49, 89, 23);
    -fx-border-radius: 5;
    -fx-padding: 3 6 6 6;
}
```

The font color, border color, border radius, and padding are picked up from this definition. The color of the button and the font style of the label are picked up from the `.root` definition from Example 37–3. Buttons with this styling look as shown in the following image.

Note: If a class does not have a style defined in the modena.css style sheet, define the style in your style sheet and assign it to each class instance as shown in Example 37–6. For example, layout panes do not have styles defined in the modena.css style sheet. See Styling Layout Panes with CSS for information on creating styles for classes such as `HBox` and `GridPane`.

Creating Class Styles

You can create a class style by adding a definition for it to your style sheet. Example 37–5 defines a new style in controlStyle1.css called `.button1`.

Example 37–5 Define a New Style

```
.button1{
    -fx-text-fill: #006464;
    -fx-background-color: #DFB951;
    -fx-border-radius: 20;
    -fx-background-radius: 20;
    -fx-padding: 5;
}
```

Any button to which this style is added looks as shown in the following image. Note that the font of the label is picked up from the `.root` definition in controlStyle1.css.

To assign this class style to a node, use the `getStyleClass().add()` sequence of methods. Example 37–6 shows the `.button1` style assigned to the Accept button.

Example 37–6 Assign a Class Style

```
Button buttonAccept = new Button("Accept");
buttonAccept.getStyleClass().add("button1");
```

Be aware that adding styles to a node is accumulative. After adding the `.button1` style class to the `buttonAccept` node, the node is rendered using rules from both the `.button` and `.button1` styles.

Creating ID Styles

You can define a style for an individual node by creating a style and assigning the style to the node. The style name is the ID preceded by a hash symbol (#). Example 37–7 creates a definition for a style named `#font-button`.

Example 37–7 Define an ID Style

```
#font-button {
    -fx-font: bold italic 20pt "Arial";
    -fx-effect: dropshadow( one-pass-box , black , 8 , 0.0 , 2 , 0 );
}
```

The button that is assigned the ID `font-button` looks as shown in the following image.

Example 37–8 shows how to assign the ID style to a node.

Example 37–8 Assign an ID Style

```
Button buttonFont = new Button("Font");
buttonFont.setId("font-button");
```

Setting Styles in the Code

You also have the option of setting style properties for a node within the code for your application. Rules set within the code take precedence over styles from a style sheet. Example 37–9 shows how to change the background color and font color for a button.

Example 37–9 Define a Style Inline

```
Button buttonColor = new Button("Color");
buttonColor.setStyle("-fx-background-color: slateblue; -fx-text-fill: white;");
```

The following image shows how the button appears.

Additional Resources

For more in-depth information on JavaFX style sheets, see the JavaFX CSS Reference Guide.

For information on styling the UI Controls, see Using JavaFX UI Controls.

For information on styling layout panes, see Styling Layout Panes with CSS.

For information on styling charts, see Styling Charts with CSS.

38

Styling Charts with CSS

This chapter explains how to change the default appearance of JavaFX charts by applying Cascading Style Sheets (CSS). Learn how to change a chart color scheme, modify its legend or axes, and alter chart symbols.

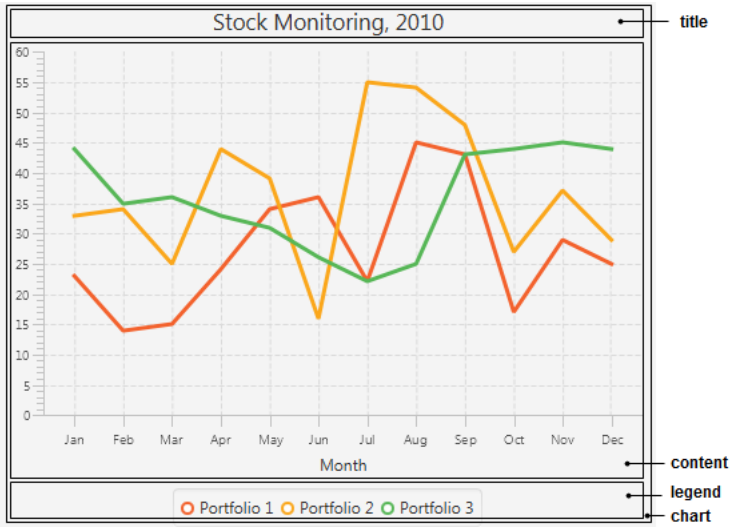
All visual elements of JavaFX charts are defined by the modena style sheet. The JavaFX API has a limited set of methods and properties to alter these visual elements. Oracle recommends that you use the chart-specific CSS properties to implement an alternative look and feel for charts in your JavaFX application.

You can find a complete list of the chart-specific properties in the JavaFX CSS Reference Guide. When you apply CSS styles to your charts, refer to Skinning JavaFX Applications with CSS for implementation details.

Modifying Basic Chart Elements

All JavaFX charts have common properties that can be set though the `.chart`, `.chart-content`, `.chart-title`, and `.chart-legend` CSS classes. Figure 38–1 shows the corresponding areas of the chart.

Figure 38–1 Visual Elements of a Chart



You can change and set the following visual characteristics of these elements:

- Padding and insets

- Background color and image
- Font
- Text fill color

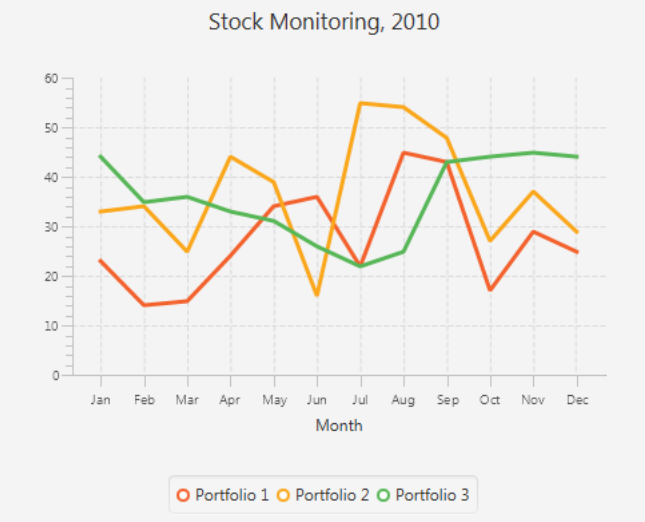
By default, any chart has 5-pixel padding and its content has 10-pixel padding. You can alter these values by using the `-fx-padding` properties of the `.chart` and `.chart-content` CSS classes as shown in Example 38–1.

Example 38–1 Set Chart Padding

```
.chart {
    -fx-padding: 10px;
}
.chart-content {
    -fx-padding: 30px;
}
```

Figure 38–2 shows the view of the line chart after these styles are applied.

Figure 38–2 Setting Chart Top-Level CSS Properties



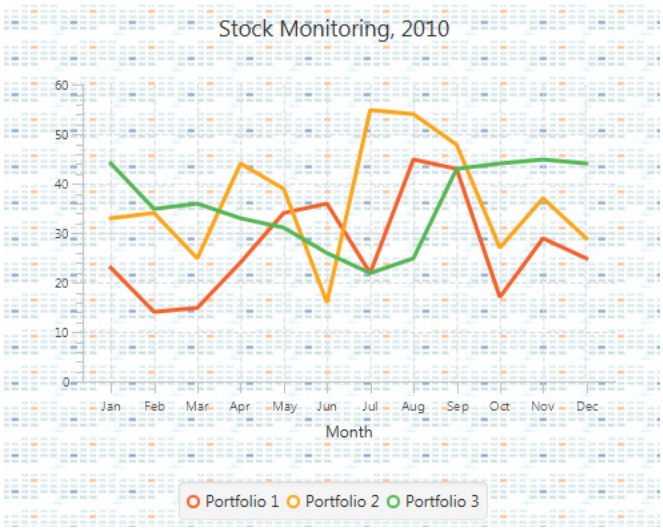
You can define a background color or a background image for the chart. Add the `-fx-background-image` property as shown in Example 38–2.

Example 38–2 Setting a Background Image

```
.chart {
    -fx-padding: 10px;
    -fx-background-image: url("icon.png");
}
.chart-content {
    -fx-padding: 30px;
}
```

Because the icon is smaller than the line chart, the image is repeated to fill the remaining area. Figure 38–3 shows line chart when the background image is applied.

Figure 38–3 Line Chart with a Background Image



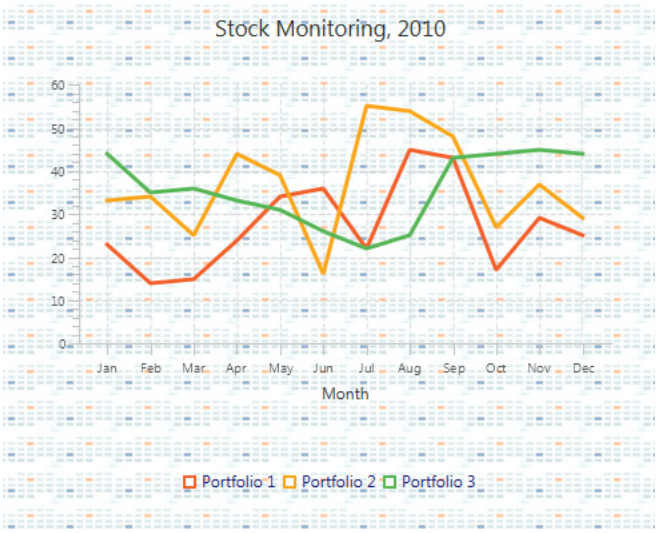
The chart legend for the line chart shown in Figure 38–3 has the default look and feel. You can change its appearance by altering the properties defined in the `.chart-legend` CSS class, as demonstrated in Example 38–3.

Example 38–3 Setting a Chart Legend

```
.chart {
    -fx-padding: 10px;
    -fx-background-image: url("icon.png");
}
.chart-content {
    -fx-padding: 30px;

}
.chart-legend {
   -fx-background-color:  transparent;
   -fx-padding: 20px;
}

.chart-legend-item-symbol{
   -fx-background-radius: 0;
}

.chart-legend-item{
    -fx-text-fill: #191970;
}
```

When you apply these styles, the chart legend is rendered with a transparent background, the labels are painted with dark blue, and the legend symbols change to square, as shown in Figure 38–4.

Figure 38–4 Changing the Chart Legend



By default, legend symbols look like circles, because they are declared as rounded rectangles with a 5-pixel height, 5-pixel width, and 5-pixel radius. When you explicitly set the radius to 0, the circles turn into squares. You can also define the legend symbol by using the `-fx-shape` property. For example, the following line creates a triangle by specifying its SVG path: `-fx-shape: "M5,0 L10,8 L0,8 Z."`

To alter the chart text elements, you should use the corresponding styles as shown in Example 38–4. The `.chart-title` class sets the fill color and the font size for the chart title. The `.axis-label` class define the fill color of the axis label.

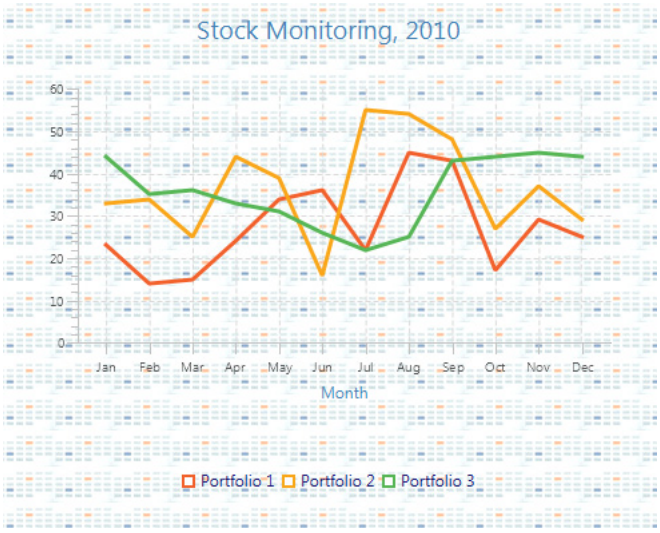
Example 38–4 Changing Color of the Text Elements

```
.chart {
    -fx-padding: 10px;
    -fx-background-image: url("icon.png");
}
.chart-content {
    -fx-padding: 30px;
}

.chart-title {
  -fx-text-fill: #4682b4;
  -fx-font-size: 1.6em;
}

.axis-label {
  -fx-text-fill: #4682b4;
}

.chart-legend {
   -fx-background-color:  transparent;
   -fx-padding: 20px;
}

.chart-legend-item-symbol{
   -fx-background-radius: 0;
}
```

```
.chart-legend-item{
    -fx-text-fill: #191970;
}
```

These modifications result in the appearance shown in Figure 38–5.

Figure 38–5 Line Chart with the Modified Text Elements



Altering Colors of the Chart Plot

When you change the default background color of a chart or set an image as a chart background, the changes do not affect the graph itself. As specified in the modena style sheet, the chart plot of a two-axis chart has a light gray background, and its alternative rows are gray. Use the `-fx-background-color` and `-fx-background-image` properties of the `.chart-plot-background` class to set the background for the chart plot. Example 38–5 defines the background color for the chart plot, the fill color for the alternative rows, and the color of vertical and horizontal grid lines.

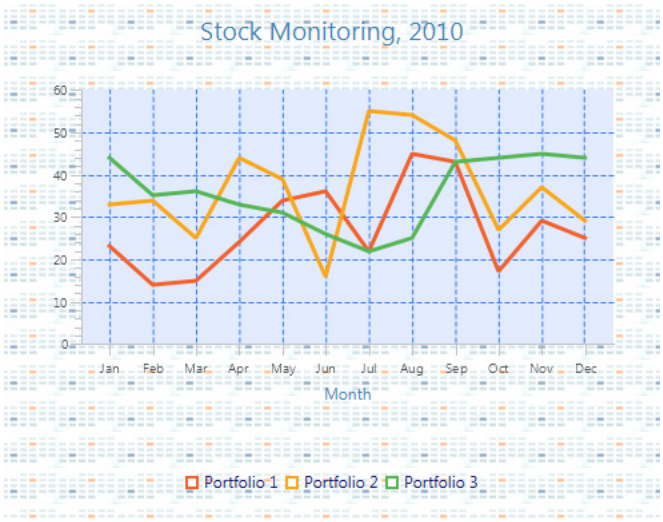
Example 38–5 Setting a Background Color for a Chart Plot

```
.chart {
    -fx-padding: 10px;
    -fx-background-image: url("icon.png");
}
.chart-content {
    -fx-padding: 30px;
}

.chart-title {
  -fx-text-fill: #4682b4;
  -fx-font-size: 1.6em;
}

.axis-label {
  -fx-text-fill: #4682b4;
}

.chart-legend {
   -fx-background-color:  transparent;
```

```
   -fx-padding: 20px;
}

.chart-legend-item-symbol{
   -fx-background-radius: 0;
}

.chart-legend-item{
    -fx-text-fill: #191970;
}

.chart-plot-background {
    -fx-background-color: #e2ecfe;
}
.chart-vertical-grid-lines {
    -fx-stroke: #3278fa;
}
.chart-horizontal-grid-lines {
    -fx-stroke: #3278fa;
}
.chart-alternative-row-fill {
    -fx-fill: #99bcfd;
    -fx-stroke: transparent;
    -fx-stroke-width: 0;
}
```

Figure 38–6 shows the line chart with the modified plot background.

Figure 38–6 Line Chart with an Alternative Plot Color



When you design your chart so that its plot has the same background as the other chart areas, set a transparent background for the plot and alternative rows, as shown in Example 38–6.

Example 38–6 Setting a Transparent Background for the Chart Plot

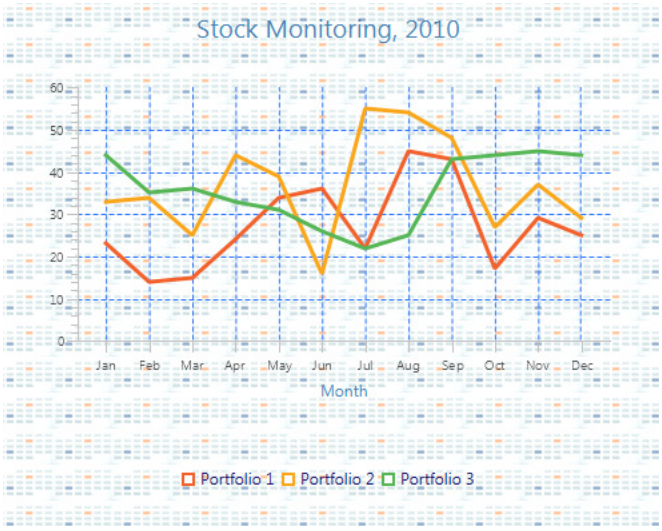
```
.chart {
    -fx-padding: 10px;
    -fx-background-image: url("icon.png");
}
```

```
.chart-content {
    -fx-padding: 30px;
}

.chart-title {
  -fx-text-fill: #4682b4;
  -fx-font-size: 1.6em;
}

.axis-label {
  -fx-text-fill: #4682b4;
}

.chart-legend {
   -fx-background-color:  transparent;
   -fx-padding: 20px;
}

.chart-legend-item-symbol{
   -fx-background-radius: 0;
}

.chart-legend-item{
    -fx-text-fill: #191970;
}

.chart-plot-background {
    -fx-background-color: transparent;
}
.chart-vertical-grid-lines {
    -fx-stroke: #3278fa;
}
.chart-horizontal-grid-lines {
    -fx-stroke: #3278fa;
}
.chart-alternative-row-fill {
    -fx-fill: transparent;
    -fx-stroke: transparent;
    -fx-stroke-width: 0;
}
```

You can make the alternative rows invisible by applying the `setAlternativeRowFillVisible(false)` method to the chart in the JavaFX application.

When the transparent background colors are applied, the chart appears as shown in Figure 38–7.

Figure 38–7 Line Chart with a Transparent Plot Background



Setting the Axes

Although the `Axis` class provides methods and properties to set the tick marks and labels, you can use the corresponding CSS classes and properties to define the appearance for these chart elements.

Consider the bubble chart sample described in the Bubble Chart chapter. Disable the color set for the tick labels in Example 34–4 by either deleting or commenting out the following lines.

- `xAxis.setTickLabelFill(Color.CHOCOLATE);`
- `yAxis.setTickLabelFill(Color.CHOCOLATE);`

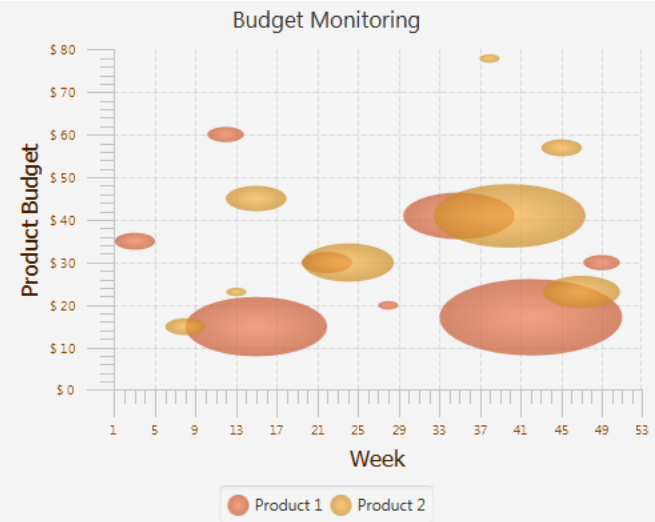
Add the code fragment shown in Example 38–7 to the CSS file of the application.

Example 38–7 Defining Styles for the Chart Axes

```
.axis {
    -fx-font-size: 1.4em;
    -fx-tick-label-fill: #914800;
    -fx-font-family: Tahoma;
    -fx-tick-length: 20;
    -fx-minor-tick-length: 10;
}

.axis-label {
   -fx-text-fill: #462300;
}
```

This style sheet defines the relative font size, font family, and fill colors for the axis labels and tick labels. It also sets the lengths for the tick marks and minor tick marks. When these styles are applied to the chart, it looks as shown in Figure 38–8.

Figure 38–8 Bubble Chart with the Modified Appearance of Its Axes



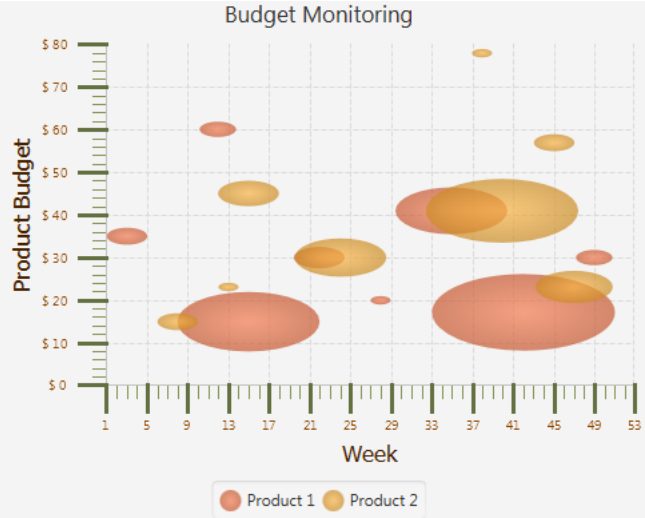
Example 38–7 changes the default values for the length of tick marks and minor tick marks. You can continue changing their appearance by defining a new color scheme, as shown in Example 38–8. This example also sets a 3-pixel width for the basic tick marks, so that they look thicker than minor tick marks.

Example 38–8 Altering Colors of Tick Marks and Minor Tick Marks

```
.axis {
    -fx-font-size: 1.4em;
    -fx-tick-label-fill: #914800;
    -fx-font-family: Tahoma;
    -fx-tick-length: 20;
    -fx-minor-tick-length: 10;
}

.axis-label {
   -fx-text-fill: #462300;
}

.axis-tick-mark {
    -fx-stroke: #637040;
    -fx-stroke-width: 3;
}
.axis-minor-tick-mark {
    -fx-stroke: #859656;
}
```

Figure 38–9 shows how the axes change when you modify the color and width of the chart tick marks.

Figure 38–9 Alternative Color Scheme and Width for the Tick Marks



Setting Chart Colors

Changing the default colors of the charts is the simple way to provide a unique style for your JavaFX application. This section describes some aspects of setting alternative colors for basic types of charts.

By default, the modena style sheet defines eight colors of line that correspond to the first eight series of data. When the number of data series added to the line chart exceeds eight, the color of the extra lines is defined in the `.chart-series-line` CSS class.

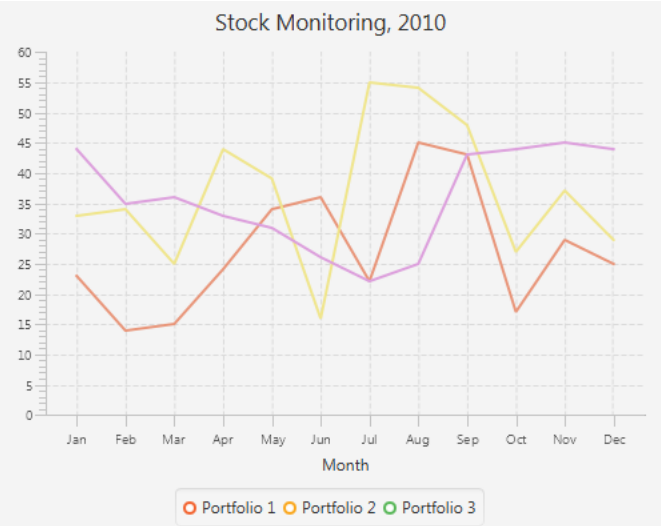
Use the `.chart-series-line` class and `.default-color<x>.chart-series-line` classes to change the style of lines. The style defined in Example 38–9 sets new colors for the lines of the three data series, removes the default effects, and specifies a 2-pixel width.

Example 38–9 Setting Alternative Colors for Three Series in a Line Chart

```
.chart-series-line {
    -fx-stroke-width: 2px;
    -fx-effect: null;
}

.default-color0.chart-series-line { -fx-stroke: #e9967a; }
.default-color1.chart-series-line { -fx-stroke: #f0e68c; }
.default-color2.chart-series-line { -fx-stroke: #dda0dd; }
```

Figure 38–10 shows the line chart after this style has been applied.

Figure 38–10 Line Chart with the Modified Line Colors



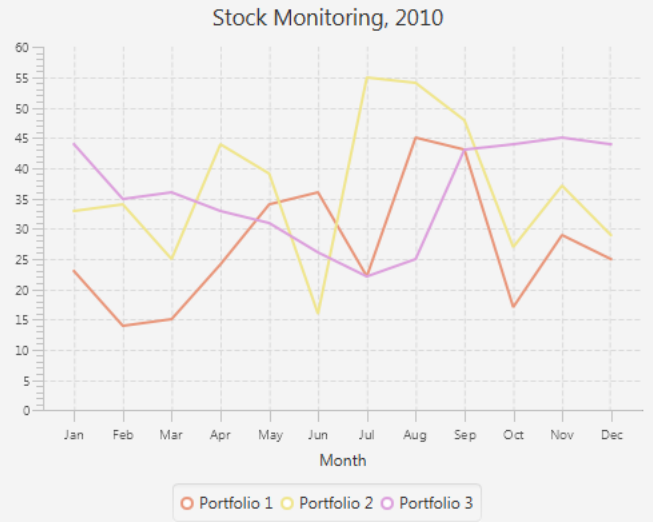
Note that the legend still shows the default colors of the chart series. This is because the corresponding changes were not applied to the chart symbols. Example 38–10 shows how to change the colors of the series in the legend.

Example 38–10 Changing Chart Symbol Color

```
.chart-series-line {
    -fx-stroke-width: 2px;
    -fx-effect: null;
}

.default-color0.chart-series-line { -fx-stroke: #e9967a; }
.default-color1.chart-series-line { -fx-stroke: #f0e68c; }
.default-color2.chart-series-line { -fx-stroke: #dda0dd; }

.default-color0.chart-line-symbol { -fx-background-color: #e9967a, white; }
.default-color1.chart-line-symbol { -fx-background-color: #f0e68c, white; }
.default-color2.chart-line-symbol { -fx-background-color: #dda0dd, white; }
```

Compare Figure 38–10 and Figure 38–11 to observe the change in the chart legend.

Figure 38–11 Line Chart with the Modified Colors of Its Lines and Proper Legend Symbols



When you change colors in a area chart, consider three graphical components: the area for each data series, the corresponding bounding line, and the chart symbol. By default, the modena styles sheet defines a color scheme for eight series of data, including the colors of their areas, lines, and symbols. The default style also sets the basic color for areas, lines, and symbols of additional series.

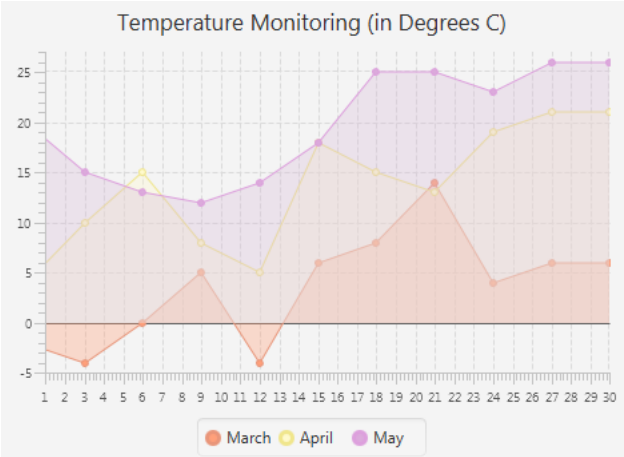
Example 38–11 shows how to change the default color scheme for the areas that correspond to three series of data.

Example 38–11 Creating a New Color Scheme for an Area Chart

```
.default-color0.chart-area-symbol { -fx-background-color: #e9967a, #ffa07a; }
.default-color1.chart-area-symbol { -fx-background-color: #f0e68c, #fffacd; }
.default-color2.chart-area-symbol { -fx-background-color: #dda0dd, #d8bfd8; }

.default-color0.chart-series-area-line { -fx-stroke: #e9967a; }
.default-color1.chart-series-area-line { -fx-stroke: #f0e68c; }
.default-color2.chart-series-area-line { -fx-stroke: #dda0dd; }

.default-color0.chart-series-area-fill { -fx-fill: #ffa07a aa}
.default-color1.chart-series-area-fill { -fx-fill: #fffacd 77; }
.default-color2.chart-series-area-fill { -fx-fill: #d8bfd8 33; }
```

Pay attention to the values in bold. These bold characters set new values for area opacity. By default, all areas have an opacity level of 0.17. Example 38–11 reassigns opacity for the areas so that the first area has the lowest opacity level, and the third area has the highest opacity level. Note that the hexadecimal color syntax with alpha is not a standard W3C CSS format. To conform with the W3C requirements, use the `rgba` CSS function with the fourth parameter as an alpha value. Figure 38–12 shows how these styles change the chart appearance when they are applied.

Figure 38–12 Area Chart with an Alternative Color Scheme



Example 38–12 shows the basic style for all bars in the bar chart defined in the modena style sheet. This style creates a linear gradient for the background color and sets the radius so that all bar edges look rounded.

Example 38–12 Default Style of the Bar Chart

```
.chart-bar {
    -fx-bar-fill: #22bad9;
    -fx-background-color: linear (0%,0%) to (0%,100%)
        stops (0%, derive(-fx-bar-fill,-30%))
        (100%, derive(-fx-bar-fill,-40%)),
        linear (0%,0%) to (0%,100%)
        stops (0%, derive(-fx-bar-fill,80%))
        (100%, derive(-fx-bar-fill, 0%)),
        linear (0%,0%) to (0%,100%)
        stops (0%, derive(-fx-bar-fill,30%))
        (100%, derive(-fx-bar-fill,-10%));
    -fx-background-insets: 0,1,2;
    -fx-background-radius: 5 5 0 0, 4 4 0 0, 3 3 0 0;
}
```

The background settings are not limited to colors, gradients, and effects. You can also set background image for each data series. To implement this approach, first simplify the BarChartSample application as shown in Example 38–13.

Example 38–13 Simplified Bar Chart Sample

```
package barchartsample;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
```

```
import javafx.stage.Stage;

public class BarChartSample extends Application {

    final static String austria = "Austria";
    final static String brazil = "Brazil";
    final static String france = "France";
    final static String italy = "Italy";
    final static String usa = "USA";

    @Override
    public void start(Stage stage) {
        stage.setTitle("Bar Chart Sample");
        final CategoryAxis xAxis = new CategoryAxis();
        final NumberAxis yAxis = new NumberAxis();
        final BarChart<String, Number> bc =
                new BarChart<String, Number>(xAxis, yAxis);
        bc.setTitle("Country Summary");
        xAxis.setLabel("Country");
        xAxis.setTickLabelRotation(90);
        yAxis.setLabel("Value");

        XYChart.Series series1 = new XYChart.Series();
        series1.setName("2003");
        series1.getData().add(new XYChart.Data(austria, 25601.34));
        series1.getData().add(new XYChart.Data(brazil, 20148.82));
        series1.getData().add(new XYChart.Data(france, 10000));
        series1.getData().add(new XYChart.Data(italy, 35407.15));
        series1.getData().add(new XYChart.Data(usa, 11000));

        Scene scene = new Scene(bc, 400, 600);
        bc.getData().add(series1);
        bc.setLegendVisible(false);
        stage.setScene(scene);
        scene.getStylesheets().add("barchartsample/Chart.css");
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Now define the chart style sheet as shown in Example 38–14.

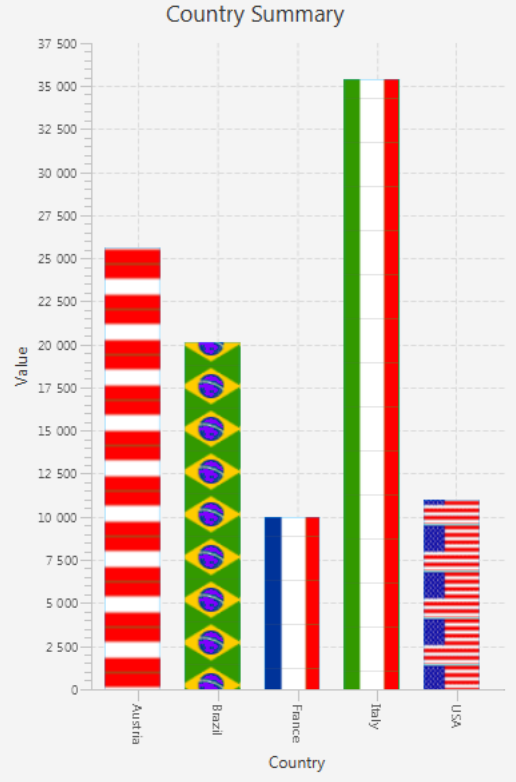
Example 38–14 Adding Images to the Bar Background

```
.chart-bar {
    -fx-background-color: rgba(0,168,355,0.05);
    -fx-border-color: rgba(0,168,355,0.3) rgba(0,168,355,0.3)
        transparent rgba(0,168,355,0.3);
    -fx-background-radius: 0;
    -fx-background-position: left center;
}

.data0.chart-bar {
    -fx-background-image: url("austria.png");
}
.data1.chart-bar {
    -fx-background-image: url("brazil.png");
```

```
}
.data2.chart-bar {
    -fx-background-image: url("france.png");
}
.data3.chart-bar {
    -fx-background-image: url("italy.png");
}
.data4.chart-bar {
    -fx-background-image: url("usa.png");
}
```

This style sets a background image for each data series and defines the position of the image within the bar. Figure 38–13 shows how the BarChartSample looks after the new styles are applied.

Figure 38–13 Using Images to Fill a Bar Chart



When you build pie charts in your JavaFX application, you typically need to set alternative colors for the pie chart slices. You can redefine the default color scheme by setting the `.default-color<x>.chart-pie` CSS classes. Example 38–15 implements this task.

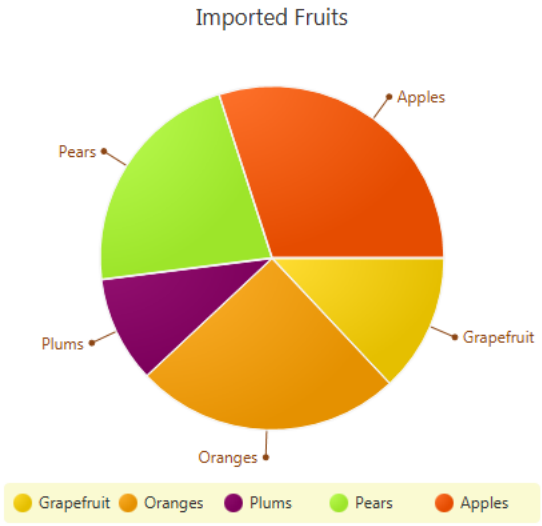
Example 38–15 Setting Colors of a Pie Chart

```
.default-color0.chart-pie { -fx-pie-color: #ffd700; }
.default-color1.chart-pie { -fx-pie-color: #ffa500; }
.default-color2.chart-pie { -fx-pie-color: #860061; }
.default-color3.chart-pie { -fx-pie-color: #adff2f; }
.default-color4.chart-pie { -fx-pie-color: #ff5700; }
```

```
.chart-pie-label-line {
    -fx-stroke: #8b4513;
    -fx-fill: #8b4513;
}

.chart-pie-label {
    -fx-fill: #8b4513;
    -fx-font-size: 1em;
}
.chart-legend {
   -fx-background-color:  #fafad2;
   -fx-stroke: #daa520;
}
```

The pie chart colors now resemble the colors of fruit that they represent. Additionally, the style sheet in Example 38–15 sets alternative colors for the labels and the chart legend. You can observe the new style in Figure 38–14.

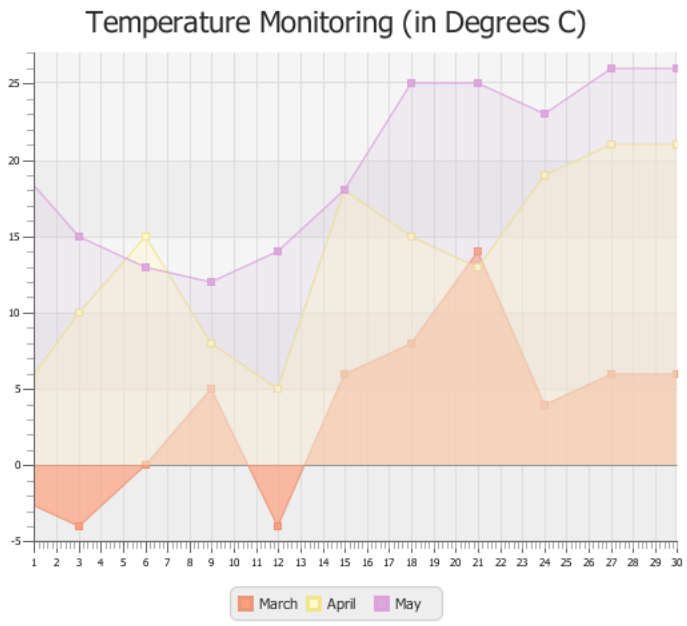
Figure 38–14 Pie Chart with the Redefined Colors of Its Slices



Changing Chart Symbols

Although symbols to display in the chart legend are defined in the modena style sheet, you can change their appearance by modifying the default color scheme and symbol shape. Example 38–11 changes the colors of the area chart symbols. You can add the following line to change the symbol shape to a square: `.chart-area-symbol{-fx-background-radius: 0;}`. By default, the background radius is 5 pixels. Changing the background radius to 0, turns the circle into a square. This change applies to all series of data, as shown in Figure 38–15.

Figure 38–15 Area Chart with Modified Chart Symbols



In scatter charts, all data is represented by set of points. Each data series has its special symbol. By default, the modena style defines seven symbols for seven series of data and the basic symbol that it uses for other data series. Example 38–16 shows the default styles for the scatter charts.

Example 38–16 Styles for a Scatter Chart Defined in the Modena Style Sheet

```
.chart-symbol { /* solid circle */
    -fx-background-color: CHART_COLOR_1;
    -fx-background-radius: 5px;
    -fx-padding: 5px;
}
.default-color1.chart-symbol { /* solid square */
    -fx-background-color: CHART_COLOR_2;
    -fx-background-radius: 0;
}
.default-color2.chart-symbol { /* solid diamond */
    -fx-background-color: CHART_COLOR_3;
    -fx-background-radius: 0;
    -fx-padding: 7px 5px 7px 5px;
    -fx-shape: "M5,0 L10,9 L5,18 L0,9 Z";
}
.default-color3.chart-symbol { /* cross */
    -fx-background-color: CHART_COLOR_4;
    -fx-background-radius: 0;
    -fx-background-insets: 0;
    -fx-shape: "M2,0 L5,4 L8,0 L10,0 L10,2 L6,5 L10,8 L10,10 L8,10 L5,6 L2,10
L0,10 L0,8 L4,5 L0,2 L0,0 Z";
}
.default-color4.chart-symbol { /* solid triangle */
    -fx-background-color: CHART_COLOR_5;
    -fx-background-radius: 0;
    -fx-background-insets: 0;
    -fx-shape: "M5,0 L10,8 L0,8 Z";
}
```

```
.default-color5.chart-symbol { /* hollow circle */
    -fx-background-color: CHART_COLOR_6, white;
    -fx-background-insets: 0, 2;
    -fx-background-radius: 5px;
    -fx-padding: 5px;
}
.default-color6.chart-symbol { /* hollow square */
    -fx-background-color: CHART_COLOR_7, white;
    -fx-background-insets: 0, 2;
    -fx-background-radius: 0;
}
.default-color7.chart-symbol { /* hollow diamond */
    -fx-background-color: CHART_COLOR_8, white;
    -fx-background-radius: 0;
    -fx-background-insets: 0, 2.5;
    -fx-padding: 7px 5px 7px 5px;
    -fx-shape: "M5,0 L10,9 L5,18 L0,9 Z";
}
```

You can use these CSS classes and the available CSS properties to change the symbols of scatter charts, or you can invent your own symbols to represent data.

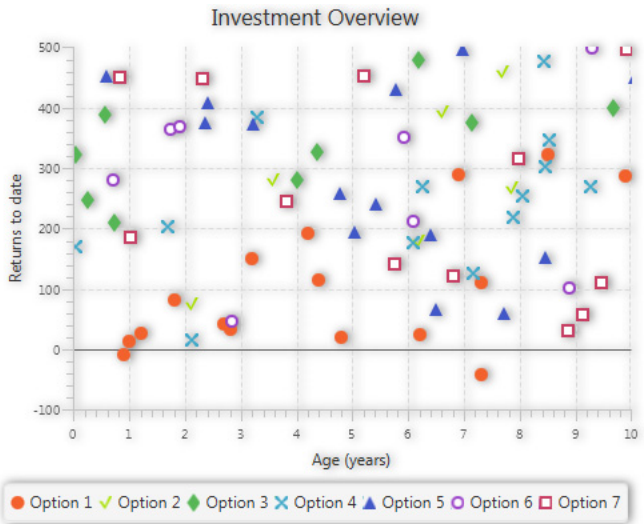
Use the `.default-color1.chart-symbol` CSS class to change the default color and shape of the symbol for the second data series as shown in Example 38–17.

Example 38–17 Redefining the Shape of the Second Series of Data

```
.default-color1.chart-symbol {
    -fx-background-color: #a9e200;
    -fx-shape: "M0,4 L2,4 L4,8 L7,0 L9,0 L4,11 Z";
}
```

When this style is applied to the scatter chart, as shown in Figure 38–16, the points of the second series appear as check marks. The points of other series appear according to the default styles.

Figure 38–16 Scatter Chart with New Check Mark Symbol to Designate the Second Data Series



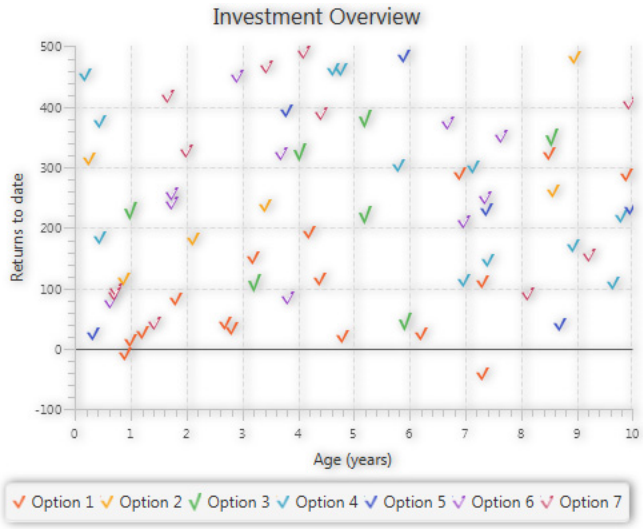
Use the `.chart-symbol` class to set a new chart symbol for the all data series in the scatter chart, as shown in Example 38–18.

Example 38–18 Defining an Alternative Symbol for a Scatter Chart

```
.chart-symbol{
    -fx-shape: "M0,4 L2,4 L4,8 L7,0 L9,0 L4,11 Z";
}
```

Figure 38–17 shows a scatter chart with seven series of data. Each series is represented by a check mark of different color. The color of each series is derived from the modena style sheet.

Figure 38–17 Scatter Chart with a Check Sign as a Chart Symbol



In conclusion, when you need to style a chart in your JavaFX application, consider the following steps:

- Add a .css file to your JavaFX application.
- Identify the graphical elements of the chart you need to change.
- Determine the corresponding CSS classes.
- Set the properties of the selected CSS classes specifying the values to attain the required appearance.

Refer to Styling UI Controls with CSS for additional information about how to style your JavaFX application with CSS.

Part V

Working with Text in JavaFX Applications

This document explains how to add text and text effects to JavaFX applications. It includes demo samples that illustrate how to apply single effects and a chain of effects to text nodes.

The document contains the following chapters:

39

Using Text in JavaFX

This chapter explains how to add text to your JavaFX applications.

It also includes code samples to illustrate the APIs being used.

Introduction

The graphical content of JavaFX applications consists of objects organized in a tree-like structure called a **scene graph**. A single element in the scene graph is called a **node**. Nodes can handle different types of content, including text. Nodes can be transformed and animated. You can also apply various effects to nodes. Using features common to all node types enables you to provide sophisticated text content that meets the demands of modern rich Internet applications (RIAs).

The JavaFX SDK provides the `javafx.scene.text.Text` class that is used to display text. The `Text` class inherits from the `Node` class. For this reason, you can apply effects, animation, and transformations to text nodes in the same way as to any other nodes. Because the `Node` class inherits from the `Shape` class, you can set a stroke or apply a fill setting to text nodes in the same way as to any shape.

Adding Text

To add a text object to your application, use any of the constructors shown in Example 39–1 through Example 39–3.

Example 39–1

```
Text t = new Text();
t.setText("This is a text sample");
```

Example 39–2

```
Text t = new Text("This is a text sample");
```

Example 39–3

```
Text t = new Text (10, 20, "This is a text sample");
```

The constructor in Example 39–3 creates a text object at the coordinates specified with the first two parameters.

Setting Text Font and Color

When adding text, you can also set some of its properties. To set the font, you can use an instance of the `javafx.scene.text.Font` class. The `Font.font()` method enables you to specify the font family name and size. You can also set the text color as shown in Example 39–4.

Example 39–4

```
t.setText("This is a text sample");
t.setFont(Font.font ("Verdana", 20));
t.setFill(Color.RED);
```

Alternatively, you may want to use a system font, which varies depending on the platform. For this purpose, call the `Font.getDefault()` method.

In the production code, Oracle recommends that you set the styles using cascading style sheets (CSS). For example, to be able to apply a linear gradient fill to your text objects, add the style with the required rules to your CSS as shown in Example 39–5.

Example 39–5

```
#fancytext {
    -fx-font: 100px Tahoma;
    -fx-fill: linear-gradient(from 0% 0% to 100% 200%, repeat, aqua 0%, red 50%);
    -fx-stroke: black;
    -fx-stroke-width: 1;
}
```

In your code, create a text object and apply the style from CSS as shown in Example 39–6.

Example 39–6

```
Text t = new Text ("Stroke and Fill");
t.setId("fancytext");
```

This code creates the text shown in Figure 39–1.

Figure 39–1

For more details about using CSS in JavaFX applications, see Skinning JavaFX Applications with CSS.

Making Text Bold or Italic

To make the text look bold, use the `FontWeight` constant of the `font` method as shown in Example 39–7.

Example 39–7

```
t.setFont(Font.font("Verdana", FontWeight.BOLD, 70));
```

To display text in italic, use the `FontPosture` constant as shown in Example 39–8.

Example 39–8

```
t.setFont(Font.font("Verdana", FontPosture.ITALIC, 20));
```

Using Custom Fonts

If you need to use a unique font that might not be installed on another computer, you can include a TrueType font (.ttf) or an OpenType (.otf) in your JavaFX application.

To include a TrueType or OpenType font as a custom font, use the following procedure:

1. Create a `resources/fonts` folder in your project folder.
2. Copy your font files to the `fonts` subfolder in your project.
3. In your source code, load the custom font as shown in Example 39–9.

Example 39–9

```
text.setFont(Font.loadFont("file:resources/fonts/isadoracyr.ttf", 120));
```

This code provides the font for the text shown in Figure 39–2.

Figure 39–2

Setting LCD Text Support

LCD (liquid crystal display) text is an anti-aliased text that takes advantage of the properties of LCD panels to render smoother text. You can take advantage of the LCD text on the text nodes by using the API shown in Example 39–10.

Example 39–10

```
text.setFontSmoothingType(FontSmoothingType.LCD));
```

Alternatively, you can provide this setting in a .css file by using the syntax shown in Example 39–11.

Example 39–11

```
.text { -fx-font-smoothing-type: lcd; }
```

Rich Text and Bidirectional Support

You can create several `Text` nodes and lay them out in a single text flow by using the `TextFlow` layout pane. The `TextFlow` object employs the text and font of each `Text` node but ignores the wrapping width and the x and y properties of its children. The `TextFlow` object uses its own width and text alignment to determine the location of each child. Example 39–12 shows three `Text` nodes that have different fonts and text laid out in a `TextFlow` pane.

Example 39–12

```
String family = "Helvetica";
double size = 50;

TextFlow textFlow = new TextFlow();
textFlow.setLayoutX(40);
textFlow.setLayoutY(40);
Text text1 = new Text("Hello ");
text1.setFont(Font.font(family, size));
text1.setFill(Color.RED);
Text text2 = new Text("Bold");
text2.setFill(Color.ORANGE);
text2.setFont(Font.font(family, FontWeight.BOLD, size));
Text text3 = new Text(" World");
text3.setFill(Color.GREEN);
text3.setFont(Font.font(family, FontPosture.ITALIC, size));
textFlow.getChildren().addAll(text1, text2, text3);

Group group = new Group(textFlow);
Scene scene = new Scene(group, 500, 150, Color.WHITE);
stage.setTitle("Hello Rich Text");
stage.setScene(scene);
stage.show();
```

This code provides the output shown in Figure 39–3.

Figure 39–3

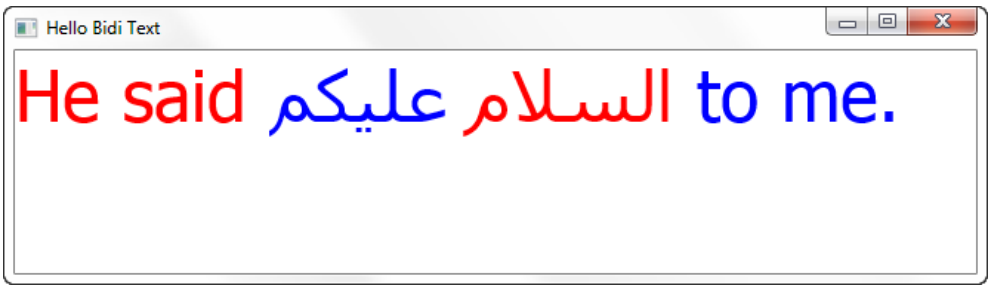
You can embed objects such as shapes and images together with the text nodes in a `TextFlow` object or create text with bidirectional support as shown in Example 39–13.

Example 39–13

```
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.scene.text.TextFlow;
import javafx.stage.Stage;

public class JavaFXBidiText extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        TextFlow textFlow = new TextFlow();
        Font font = new Font("Tahoma", 48);

        Text text1 = new Text("He said \u0627\u0644\u0633\u0644\u0627\u0645");
        text1.setFill(Color.RED);
        text1.setFont(font);
        Text text2 = new Text(" \u0639\u0644\u064a\u0643\u0645 to me.");
        text2.setFill(Color.BLUE);
        text2.setFont(font);
        textFlow.getChildren().addAll(text1, text2);

        Group group = new Group(textFlow);
        Scene scene = new Scene(group, 650, 150, Color.WHITE);
        stage.setTitle("Hello Bidi Text");
        stage.setScene(scene);
        stage.show();
    }
```

This code creates the output shown in Figure 39–4.

Figure 39–4



This example shows how the content of a single `Text` node can be divided and placed in different locations on the `TextFlow` object due to bidirectional reordering.

The default orientation of the `TextFlow` object is left-to-right with Arabic words to be rendered right-to-left. There are two Arabic words to be rendered together: the red word is rendered first at the rightmost position and is followed to the left by the blue word.

You can change the default orientation of the `TextFlow` object by calling

```
textFlow.setNodeOrientation(NodeOrientation.RIGHT_TO_LEFT);
```

Note that both the `Text` and `TextFlow` objects support bidirectional reordering as defined by the Unicode Consortium in the Bidirectional Algorithm Annex #9 at

http://www.unicode.org/reports/tr9/

40

Applying Effects to Text

In this chapter you learn how to apply single effects and a chain of effects to text nodes. It includes demo samples to illustrate the APIs being used.

The JavaFX SDK provides a wide set of effects that reside in the `javafx.scene.effect` package. For a complete set of available effects, see the API documentation. You can see some of the effects in action in the `TextEffects` demo application. This application displays text nodes with various effects. Download the `TextEffectsSample.zip` file, extract the files, save them on your computer, and open a NetBeans project in the NetBeans IDE.

Perspective Effect

The `PerspectiveTansform` class enables you to imitate a three-dimensional effect for your two-dimensional content. The perspective transformation can map an arbitrary quadrilateral into another quadrilateral. An input for this transformation is your node. To define the perspective transformation, you specify the x and y coordinates of output locations of all four corners. In the `TextEffects` application, the PerspectiveTransform effect is set for a group consisting of a rectangle and text as shown in Example 40–1.

Example 40–1

```
PerspectiveTransform pt = new PerspectiveTransform();
pt.setUlx(10.0f);
pt.setUly(10.0f);
pt.setUrx(310.0f);
pt.setUry(40.0f);
pt.setLrx(310.0f);
pt.setLry(60.0f);
pt.setLlx(10.0f);
pt.setLly(90.0f);

g.setEffect(pt);
g.setCache(true);

Rectangle r = new Rectangle();
r.setX(10.0f);
r.setY(10.0f);
r.setWidth(280.0f);
r.setHeight(80.0f);
r.setFill(Color.BLUE);

Text t = new Text();
t.setX(20.0f);
```

```
t.setY(65.0f);
t.setText("Perspective");
t.setFill(Color.YELLOW);
t.setFont(Font.font(null, FontWeight.BOLD, 36));

g.getChildren().add(r);
g.getChildren().add(t);
return g;
```

Using the `setCache(true)` property enables you to improve performance when rendering the node.

You can see the result of the perspective transformation in Figure 40–1.

Figure 40–1 Text with a Perspective Effect

Blur Effect

The `GaussianBlur` class provides a blur effect based on a Gaussian convolution kernel.

Example 40–2 shows a text node with an applied Gaussian blur effect implemented in the `TextEffects` application.

Example 40–2

```
Text t2 = new Text();
t2.setX(10.0f);
t2.setY(140.0f);
t2.setCache(true);
t2.setText("Blurry Text");
t2.setFill(Color.RED);
t2.setFont(Font.font(null, FontWeight.BOLD, 36));
t2.setEffect(new GaussianBlur());
return t2;
```

You can see the result of the Gaussian blur effect in Figure 40–2.

Figure 40–2 Text with a Blur Effect

Drop Shadow Effect

To implement a drop shadow effect, use the `DropShadow` class. You can specify a color and an offset for the shadow of your text. In the `TextEffects` application, the drop

shadow effect is applied to the red text and provides a three-pixel gray shadow. You can see the code in Example 40–3.

Example 40–3

```
DropShadow ds = new DropShadow();
ds.setOffsetY(3.0f);
ds.setColor(Color.color(0.4f, 0.4f, 0.4f));

Text t = new Text();
t.setEffect(ds);
t.setCache(true);
t.setX(10.0f);
t.setY(270.0f);
t.setFill(Color.RED);
t.setText("JavaFX drop shadow...");
t.setFont(Font.font(null, FontWeight.BOLD, 32));
```

You can see the result of the applied effect in Figure 40–3.

Figure 40–3 Text with a Drop Shadow Effect

Inner Shadow Effect

An inner shadow effect renders a shadow inside the edges of your content. For text content, you can specify a color and an offset. See how the inner shadow effect with four-pixel offsets in the x and y directions is applied to a text node in Example 40–4.

Example 40–4

```
InnerShadow is = new InnerShadow();
is.setOffsetX(4.0f);
is.setOffsetY(4.0f);

Text t = new Text();
t.setEffect(is);
t.setX(20);
t.setY(100);
t.setText("InnerShadow");
t.setFill(Color.YELLOW);
t.setFont(Font.font(null, FontWeight.BOLD, 80));

t.setTranslateX(300);
t.setTranslateY(300);

return t;
```

The result of the applied effect is shown in Figure 40–4.

Figure 40–4 Text with an Inner Shadow Effect

Reflection

The `Reflection` class enables you to display the reflected version of your text below the original text. You can adjust the view of your reflected text by providing additional parameters such as a bottom opacity value, a fraction of the input to be visible in the reflection, an offset between the input text and its reflection, and a top opacity value. For details, see the API documentation.

The reflection effect is implemented in the `TextEffects` application as shown in Example 40–5.

Example 40–5

```
Text t = new Text();
t.setX(10.0f);
t.setY(50.0f);
t.setCache(true);
t.setText("Reflections on JavaFX...");
t.setFill(Color.RED);
t.setFont(Font.font(null, FontWeight.BOLD, 30));

Reflection r = new Reflection();
r.setFraction(0.7f);

t.setEffect(r);

t.setTranslateY(400);
return t;
```

You can see the text node with the reflection effect in Figure 40–5.

Figure 40–5 Text with a Reflection Effect

Combining Several Effects

In the previous section you learned how to apply a single effect to a text node. To further enrich your text content, you can compose several effects and apply a chain of effects to achieve a specific visual result. Consider the `NeonSign` application shown in Figure 40–6.

Figure 40–6 The Neon Sign Application Window

The graphical scene of the `NeonSign` application consists of the following elements:

- An image of a brick wall used as a background
- A rectangle that provides a radial gradient fill
- A text node with a chain of effects
- A text field used for entering text data

The background is set in an external .css file.

This application uses a binding mechanism to set the text content on the text node. The text property of the text node is bound to the text property of the text field as shown in Example 40–6.

Example 40–6

```
Text text = new Text();
TextField textField = new TextField();
textField.setText("Neon Sign");
text.textProperty().bind(textField.textProperty());
```

You can type in the text field and view the changed contents of the text node.

A chain of effects is applied to the text node. The primary effect is a blend effect that uses the `MULTIPLY` mode to mix two inputs together: a drop shadow effect and another blend effect, `blend1`. Similarly, the `blend1` effect combines a drop shadow effect (`ds1`) and a blend effect (`blend2`). The `blend2` effect combines two inner shadow effects. Using this chain of effects and varying color parameters enables you to apply subtle and sophisticated color patterns to text objects. See the code for the chain of effects in Example 40–7.

Example 40–7

```
Blend blend = new Blend();
```

```
blend.setMode(BlendMode.MULTIPLY);

DropShadow ds = new DropShadow();
ds.setColor(Color.rgb(254, 235, 66, 0.3));
ds.setOffsetX(5);
ds.setOffsetY(5);
ds.setRadius(5);
ds.setSpread(0.2);

blend.setBottomInput(ds);

DropShadow ds1 = new DropShadow();
ds1.setColor(Color.web("#f13a00"));
ds1.setRadius(20);
ds1.setSpread(0.2);

Blend blend2 = new Blend();
blend2.setMode(BlendMode.MULTIPLY);

InnerShadow is = new InnerShadow();
is.setColor(Color.web("#feeb42"));
is.setRadius(9);
is.setChoke(0.8);
blend2.setBottomInput(is);

InnerShadow is1 = new InnerShadow();
is1.setColor(Color.web("#f13a00"));
is1.setRadius(5);
is1.setChoke(0.4);
blend2.setTopInput(is1);

Blend blend1 = new Blend();
blend1.setMode(BlendMode.MULTIPLY);
blend1.setBottomInput(ds1);
blend1.setTopInput(blend2);

blend.setTopInput(blend1);

text.setEffect(blend);
```

In this article, you learned how to add text and apply various effects to text content. For a complete set of available effects, see the API documentation.

If you need to implement a text editing area in your JavaFX application, use the `HTMLEditor` component. For more information about the `HTMLEditor` control, see HTML Editor.

Application Files

Source Code

- `TextEffects.java`
- `NeonSign.java`

NetBeans Projects

- `TextEffectsSample.zip`
- `NeonSignSample.zip`

Part VI

Source Code for the UI Components Tutorials

The following tables list the demo applications in this document with their associated source code files.

UI Control Samples

Chapter	Source File	NetBeans Project File
Label	LabelSample.java	UIControlSamples.zip
Button	ButtonSample.java	UIControlSamples.zip
Radio Button	RadioButtonSample.java	UIControlSamples.zip
Toggle Button	ToggleButtonSample.java	UIControlSamples.zip
Checkbox	CheckboxSample.java	UIControlSamples.zip
Choice Box	ChoiceBoxSample.java	UIControlSamples.zip
Text Field	TextFieldSample.java	UIControlSamples.zip
Password Field	PasswordField.java	UIControlSamples.zip
Scrollbar	ScrollBarSample.java	UIControlSamples.zip
Scroll Pane	ScrollPaneSample.java	UIControlSamples.zip
List View	ListViewSample.java	UIControlSamples.zip
Table View	TableViewSample.java	UIControlSamples.zip
Tree View	TreeViewSample.java	UIControlSamples.zip
Tree Table View	TreeTableViewSample.java	UIControlSamples.zip
Combo Box	ComboBoxSample.java	UIControlSamples.zip
Separator	SeparatorSample.java	UIControlSamples.zip
Slider	SliderSample.java	UIControlSamples.zip
Progress Bar, Progress Indicator	ProgressSample.java	UIControlSamples.zip
Hyperlink	HyperlinkSample.java	UIControlSamples.zip
Hyperlink	HTMLEditorSample.java	UIControlSamples.zip
HTML Editor	HTMLEditorSample.java	UIControlSamples.zip
Tooltip	TooltipSample.java	UIControlSamples.zip

Chapter	Source File	NetBeans Project File
Accordion and Titled Pane	TitledPaneSample.java	UIControlSamples.zip
Menu Controls	MenuSample.java	UIControlSamples.zip
Color Picker	ColorPickerSample.java	UIControlSamples.zip
Date Picker	DatePickerSample.java	UIControlSamples.zip
Pagination	PaginationSample.java	UIControlSamples.zip
File Chooser	FileChooserSample.java	UIControlSamples.zip

Chart Samples

Chapter	Source File	NetBeans Project Files
Pie Chart	PieChartSample.java	PieChartSample.zip
Line Chart	LineChartSample.java	LineChartSample.zip
Area Chart	AreaChartSample.java	AreaChartSample.zip
Bubble Chart	BubbleChartSample.java	BubbleChartSample.zip
Scatter Chart	ScatterChartSample.java	ScatterChartSample.zip
Bar Chart	BarChartSample.java	BarChartSample.zip

CSS Samples

Chapter	Source Files	NetBeans Project Files
Styling UI Controls with CSS	DownloadButton.java	UIControlCSS.zip
Styling UI Controls with CSS	DownloadButtonStyle1.css	UIControlCSS.zip
Styling UI Controls with CSS	DownloadButtonStyle2.css	UIControlCSS.zip
Styling UI Controls with CSS	StyleStage.java	UIControlCSS.zip
Styling UI Controls with CSS	UIControlCSS.java	UIControlCSS.zip
Styling UI Controls with CSS	controlStyle1.css	UIControlCSS.zip
Styling UI Controls with CSS	controlStyle2.css	UIControlCSS.zip

Text Samples

Chapter	Source File	NetBeans Project Files
Applying Effects to Text	TextEffects.java	TextEffectsSample.zip
Applying Effects to Text	NeonSign.java	NeonSignSample.zip

A

UI Control Samples

This appendix lists code for the following JavaFX samples:

- LabelSample.java
- ButtonSample.java
- RadioButtonSample.java
- ToggleButtonSample.java
- CheckboxSample.java
- ChoiceBoxSample.java
- TextFieldSample.java
- PasswordField.java
- ScrollBarSample.java
- ScrollPaneSample.java
- ListViewSample.java
- TableViewSample.java
- TreeViewSample.java
- TreeTableViewSample.java
- ComboBoxSample.java
- SeparatorSample.java
- SliderSample.java
- ProgressSample.java
- HyperlinkSample.java
- HyperlinkWebViewSample.java
- HTMLEditorSample.java
- TooltipSample.java
- TitledPaneSample.java
- MenuSample.java
- ColorPickerSample.java
- DatePickerSample.java
- PaginationSample.java

- FileChooserSample.java

For the descriptions of this source files, see Using JavaFX UI Controls.

LabelSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package labelsample;

import javafx.scene.Group;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.TextAlignment;
import javafx.stage.Stage;

public class LabelSample extends Application {

Label label3 = new Label("A label that needs to be wrapped");

    public static void main(String[] args) {
        launch(args);
    }
```

```
    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Label Sample");
        stage.setWidth(420);
        stage.setHeight(180);


        HBox hbox = new HBox();
        Image image = new Image(getClass().getResourceAsStream("labels.jpg"));

        Label label1 = new Label("Search");
        label1.setGraphic(new ImageView(image));
        label1.setFont(new Font("Arial", 30));
        label1.setTextFill(Color.web("#0076a3"));
        label1.setTextAlignment(TextAlignment.JUSTIFY);

        Label label2 = new Label ("Values");
        label2.setFont(Font.font("Cambria", 32));
        label2.setRotate(270);
        label2.setTranslateY(50);

        label3.setWrapText(true);
        label3.setTranslateY(50);
        label3.setPrefWidth(100);

        label3.setOnMouseEntered((MouseEvent e) -> {
            label3.setScaleX(1.5);
            label3.setScaleY(1.5);
        });

        label3.setOnMouseExited((MouseEvent e) -> {
            label3.setScaleX(1);
            label3.setScaleY(1);
        });

        hbox.setSpacing(10);
        hbox.getChildren().add((label1));
        hbox.getChildren().add(label2);
        hbox.getChildren().add(label3);
        ((Group)scene.getRoot()).getChildren().add(hbox);

        stage.setScene(scene);
        stage.show();
    }
}
```

ButtonSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
```

```
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package buttonsample;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class ButtonSample extends Application {

    private static final Color color = Color.web("#464646");
    Button button3 = new Button("Decline");
    DropShadow shadow = new DropShadow();
    Label label = new Label();


    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Button Sample");
        stage.setWidth(300);
        stage.setHeight(190);
        scene.getStylesheets().add("buttonsample/ControlStyle.css");
```

```
label.setFont(Font.font("Times New Roman", 22));
label.setTextFill(color);

Image imageDecline = new Image(getClass().getResourceAsStream("not.png"));
Image imageAccept = new Image(getClass().getResourceAsStream("ok.png"));

VBox vbox = new VBox();
vbox.setLayoutX(20);
vbox.setLayoutY(20);
HBox hbox1 = new HBox();
HBox hbox2 = new HBox();

Button button1 = new Button("Accept", new ImageView(imageAccept));
button1.getStyleClass().add("button1");
button1.setOnAction((ActionEvent e) -> {
    label.setText("Accepted");
});


Button button2 = new Button("Accept");
button2.setOnAction((ActionEvent e) -> {
    label.setText("Accepted");
});

button3.setOnAction((ActionEvent e) -> {
    label.setText("Declined");
});

button3.addEventHandler(MouseEvent.MOUSE_ENTERED, (MouseEvent e) -> {
    button3.setEffect(shadow);
});

button3.addEventHandler(MouseEvent.MOUSE_EXITED, (MouseEvent e) -> {
    button3.setEffect(null);
});


hbox1.getChildren().add(button2);
hbox1.getChildren().add(button3);
hbox1.getChildren().add(label);
hbox1.setSpacing(10);
hbox1.setAlignment(Pos.BOTTOM_CENTER);

Button button4 = new Button();
button4.setGraphic(new ImageView(imageAccept));
button4.setOnAction((ActionEvent e) -> {
    label.setText("Accepted");
});


Button button5 = new Button();
button5.setGraphic(new ImageView(imageDecline));
button5.setOnAction((ActionEvent e) -> {
    label.setText("Declined");
});

hbox2.getChildren().add(button4);
hbox2.getChildren().add(button5);
hbox2.setSpacing(25);
```

```
        vbox.getChildren().add(button1);
        vbox.getChildren().add(hbox1);
        vbox.getChildren().add(hbox2);
        vbox.setSpacing(10);
        ((Group)scene.getRoot()).getChildren().add(vbox);

        stage.setScene(scene);
        stage.show();
    }
}
```

RadioButtonSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package radiobuttonsample;

import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Insets;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
```

```
public class RadioButtonSample extends Application {

    final ImageView icon = new ImageView();

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Radio Button Sample");
        stage.setWidth(250);
        stage.setHeight(150);

        final ToggleGroup group = new ToggleGroup();

        RadioButton rb1 = new RadioButton("Home");
        rb1.setToggleGroup(group);
        rb1.setUserData("Home");

        RadioButton rb2 = new RadioButton("Calendar");
        rb2.setToggleGroup(group);
        rb2.setUserData("Calendar");

        RadioButton rb3 = new RadioButton("Contacts");
        rb3.setToggleGroup(group);
        rb3.setUserData("Contacts");

        group.selectedToggleProperty().addListener(
            (ObservableValue<? extends Toggle> ov, Toggle old_toggle,
            Toggle new_toggle) -> {
                if (group.getSelectedToggle() != null) {
                    final Image image = new Image(
                        getClass().getResourceAsStream(
                        group.getSelectedToggle().getUserData().toString() +
                        ".jpg"
                        )
                        );
                icon.setImage(image);
            }
        });

        HBox hbox = new HBox();
        VBox vbox = new VBox();

        vbox.getChildren().add(rb1);
        vbox.getChildren().add(rb2);
        vbox.getChildren().add(rb3);
        vbox.setSpacing(10);

        hbox.getChildren().add(vbox);
        hbox.getChildren().add(icon);
        hbox.setSpacing(50);
        hbox.setPadding(new Insets(20, 10, 10, 20));

        ((Group) scene.getRoot()).getChildren().add(hbox);
        stage.setScene(scene);
        stage.show();
    }
```

```
}
```

ToggleButtonSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package togglebuttonsample;

import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.Toggle;
import javafx.scene.control.ToggleButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

public class ToggleButtonSample extends Application {

    public static void main(String[] args) {
        launch(args);
    }
```

```
@Override
public void start(Stage stage) {
    stage.setTitle("Toggle Button Sample");
    stage.setWidth(250);
    stage.setHeight(180);

    HBox hbox = new HBox();
    VBox vbox = new VBox();

    Scene scene = new Scene(new Group(vbox));
    stage.setScene(scene);
    scene.getStylesheets().add("togglebuttonsample/ControlStyle.css");

    Rectangle rect = new Rectangle();
    rect.setHeight(50);
    rect.setFill(Color.WHITE);
    rect.setStroke(Color.DARKGRAY);
    rect.setStrokeWidth(2);
    rect.setArcHeight(10);
    rect.setArcWidth(10);

    final ToggleGroup group = new ToggleGroup();

    group.selectedToggleProperty().addListener(
        (ObservableValue<? extends Toggle> ov,
        Toggle toggle, Toggle new_toggle) -> {
        if (new_toggle == null)
            rect.setFill(Color.WHITE);
        else
            rect.setFill((Color) group.getSelectedToggle().getUserData());
    });

    ToggleButton tb1 = new ToggleButton("Minor");
    tb1.setToggleGroup(group);
    tb1.setUserData(Color.LIGHTGREEN);
    tb1.setSelected(true);
    tb1.getStyleClass().add("toggle-button1");

    ToggleButton tb2 = new ToggleButton("Major");
    tb2.setToggleGroup(group);
    tb2.setUserData(Color.LIGHTBLUE);
    tb2.getStyleClass().add("toggle-button2");

    ToggleButton tb3 = new ToggleButton("Critical");
    tb3.setToggleGroup(group);
    tb3.setUserData(Color.SALMON);
    tb3.getStyleClass().add("toggle-button3");



    hbox.getChildren().addAll(tb1, tb2, tb3);

    vbox.getChildren().add(new Label("Priority:"));
    vbox.getChildren().add(hbox);
    vbox.getChildren().add(rect);
    vbox.setPadding(new Insets(20, 10, 10, 20));


    stage.show();
    rect.setWidth(hbox.getWidth());
```

```
    }
}
```

CheckboxSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package checkboxsample;

import javafx.application.Application;
import javafx.scene.layout.StackPane;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

public class CheckboxSample extends Application {

    Rectangle rect = new Rectangle(90, 30);
    final String[] names = new String[]{"Security", "Project", "Chart"};
```

```
    final Image[] images = new Image[names.length];
    final ImageView[] icons = new ImageView[names.length];
    final CheckBox[] cbs = new CheckBox[names.length];

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Checkbox Sample");
        stage.setWidth(250);
        stage.setHeight(150);

        rect.setArcHeight(10);
        rect.setArcWidth(10);
        rect.setFill(Color.rgb(41, 41, 41));

        for (int i = 0; i < names.length; i++) {
            final Image image = images[i] =
                new Image(getClass().getResourceAsStream(names[i] + ".png"));
            final ImageView icon = icons[i] = new ImageView();
            final CheckBox cb = cbs[i] = new CheckBox(names[i]);
            cb.selectedProperty().addListener(
                    (ObservableValue<? extends Boolean> ov,
                    Boolean old_val, Boolean new_val) -> {
                        icon.setImage(new_val ? image : null);
            });
        }

        VBox vbox = new VBox();
        vbox.getChildren().addAll(cbs);
        vbox.setSpacing(5);

        HBox hbox = new HBox();
        hbox.getChildren().addAll(icons);
        hbox.setPadding(new Insets(0, 0, 0, 5));

        StackPane stack = new StackPane();

        stack.getChildren().add(rect);
        stack.getChildren().add(hbox);
        StackPane.setAlignment(rect, Pos.TOP_CENTER);

        HBox root = new HBox();
        root.getChildren().add(vbox);
        root.getChildren().add(stack);
        root.setSpacing(40);
        root.setPadding(new Insets(20, 10, 10, 20));

        ((Group) scene.getRoot()).getChildren().add(root);

        stage.setScene(scene);
        stage.show();
    }
}
```

ChoiceBoxSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package choiceboxsample;

import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.Label;
import javafx.scene.control.Tooltip;
import javafx.scene.layout.HBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class ChoiceBoxSample extends Application {

    Rectangle rect = new Rectangle(150, 30);
    final Label label = new Label("Hello");

    public static void main(String[] args) {
        launch(args);
    }
```

```
    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        scene.setFill(Color.ALICEBLUE);
        stage.setScene(scene);
        stage.show();

        stage.setTitle("ChoiceBox Sample");
        stage.setWidth(300);
        stage.setHeight(200);

        label.setFont(Font.font("Arial", 25));
        label.setLayoutX(40);

       final String[] greetings = new String[]{"Hello", "Hola", "??????", "??",
           "?????"};
       final ChoiceBox cb = new ChoiceBox(FXCollections.observableArrayList(
           "English", "Español", "????????", "????", "???")
       );

       cb.getSelectionModel().selectedIndexProperty().addListener(
           (ObservableValue<? extends Number> ov,
                Number old_val, Number new_val) -> {
                    label.setText(greetings[new_val.intValue()]);
        });

        cb.setTooltip(new Tooltip("Select the language"));
        cb.setValue("English");
        HBox hb = new HBox();
        hb.getChildren().addAll(cb, label);
        hb.setSpacing(30);
        hb.setAlignment(Pos.CENTER);
        hb.setPadding(new Insets(10, 0, 0, 10));

        ((Group) scene.getRoot()).getChildren().add(hb);
    }
}
```

TextFieldSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
```

```
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package textfieldsample;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class TextFieldSample extends Application {

    @Override
    public void start(Stage stage) {
        Group root = new Group();
        Scene scene = new Scene(root, 300, 150);
        stage.setScene(scene);
        stage.setTitle("Text Field Sample");

        GridPane grid = new GridPane();
        grid.setPadding(new Insets(10, 10, 10, 10));
        grid.setVgap(5);
        grid.setHgap(5);

        scene.setRoot(grid);

        final TextField name = new TextField();
        name.setPromptText("Enter your first name.");
        GridPane.setConstraints(name, 0, 0);
        grid.getChildren().add(name);

        final TextField lastName = new TextField();
        lastName.setPromptText("Enter your last name.");
        GridPane.setConstraints(lastName, 0, 1);
        grid.getChildren().add(lastName);

        final TextField comment = new TextField();
        comment.setPromptText("Enter your comment.");
        GridPane.setConstraints(comment, 0, 2);
        grid.getChildren().add(comment);

        Button submit = new Button("Submit");
        GridPane.setConstraints(submit, 1, 0);
```

```
        grid.getChildren().add(submit);

        Button clear = new Button("Clear");
        GridPane.setConstraints(clear, 1, 1);
        grid.getChildren().add(clear);

        final Label label = new Label();
        GridPane.setConstraints(label, 0, 3);
        GridPane.setColumnSpan(label, 2);
        grid.getChildren().add(label);

        submit.setOnAction((ActionEvent e) -> {
            if (
                    (comment.getText() != null && !comment.getText().isEmpty())
                    ) {
                label.setText(name.getText() + " " +
                        lastName.getText() + ", "
                        + "thank you for your comment!");
            } else {
                label.setText("You have not left a comment.");
            }
        });

        clear.setOnAction((ActionEvent e) -> {
            name.clear();
            lastName.clear();
            comment.clear();
            label.setText(null);
        });

        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

PasswordField.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
```

```
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package passwordfieldsample;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class PasswordFiledSample extends Application {

    final Label message = new Label("");

    @Override
    public void start(Stage stage) {
        Group root = new Group();
        Scene scene = new Scene(root, 260, 80);
        stage.setScene(scene);
        stage.setTitle("Password Field Sample");

        VBox vb = new VBox();
        vb.setPadding(new Insets(10, 0, 0, 10));
        vb.setSpacing(10);
        HBox hb = new HBox();
        hb.setSpacing(10);
        hb.setAlignment(Pos.CENTER_LEFT);

        Label label = new Label("Password");
        final PasswordField pb = new PasswordField();
        pb.setText("Your password");

        pb.setOnAction((ActionEvent e) -> {
            if (!pb.getText().equals("T2f$Ay!")) {
                message.setText("Your password is incorrect!");
                message.setTextFill(Color.rgb(210, 39, 30));
            } else {
                message.setText("Your password has been confirmed");
                message.setTextFill(Color.rgb(21, 117, 84));
            }
            pb.clear();
        });
```

```
        hb.getChildren().addAll(label, pb);
        vb.getChildren().addAll(hb, message);

        scene.setRoot(vb);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

ScrollBarSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package scrollbarsample;

import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Orientation;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.ScrollBar;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.VBox;
```

```
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class ScrollBarSample extends Application {

    final ScrollBar sc = new ScrollBar();
    final Image[] images = new Image[5];
    final ImageView[] pics = new ImageView[5];
    final VBox vb = new VBox();
    DropShadow shadow = new DropShadow();

    @Override
    public void start(Stage stage) {
        Group root = new Group();
        Scene scene = new Scene(root, 180, 180);
        scene.setFill(Color.BLACK);
        stage.setScene(scene);
        stage.setTitle("Scrollbar");
        root.getChildren().addAll(vb, sc);

        shadow.setColor(Color.GREY);
        shadow.setOffsetX(2);
        shadow.setOffsetY(2);

        vb.setLayoutX(5);
        vb.setSpacing(10);

        sc.setLayoutX(scene.getWidth()-sc.getWidth());
        sc.setMin(0);
        sc.setOrientation(Orientation.VERTICAL);
        sc.setPrefHeight(180);
        sc.setMax(360);

        for (int i = 0; i < 5; i++) {
            final Image image = images[i] =
                new Image(getClass().getResourceAsStream("fw" +(i+1)+ ".jpg"));
            final ImageView pic = pics[i] =
                new ImageView(images[i]);
            pic.setEffect(shadow);
            vb.getChildren().add(pics[i]);
        }



      sc.valueProperty().addListener((ObservableValue<? extends Number> ov,
            Number old_val, Number new_val) -> {
                vb.setLayoutY(-new_val.doubleValue());
        });

        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

ScrollPaneSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package scrollpanesample;

import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ScrollPane;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class ScrollPaneSample extends Application {

    final ScrollPane sp = new ScrollPane();
    final Image[] images = new Image[5];
    final ImageView[] pics = new ImageView[5];
    final VBox vb = new VBox();
    final Label fileName = new Label();
    final String [] imageNames = new String [] {"fw1.jpg", "fw2.jpg",
        "fw3.jpg", "fw4.jpg", "fw5.jpg"};

    @Override
    public void start(Stage stage) {
        VBox box = new VBox();
```

```
        Scene scene = new Scene(box, 180, 180);
        stage.setScene(scene);
        stage.setTitle("ScrollPaneSample");
        box.getChildren().addAll(sp, fileName);
        VBox.setVgrow(sp, Priority.ALWAYS);

        fileName.setLayoutX(30);
        fileName.setLayoutY(160);

        Image roses = new Image(getClass().getResourceAsStream("roses.jpg"));
        sp.setContent(new ImageView(roses));
        sp.setHbarPolicy(ScrollPane.ScrollBarPolicy.NEVER);



        for (int i = 0; i < 5; i++) {
            images[i] = new Image(getClass().getResourceAsStream(imageNames[i]));
            pics[i] = new ImageView(images[i]);
            pics[i].setFitWidth(100);
            pics[i].setPreserveRatio(true);
            vb.getChildren().add(pics[i]);
        }

        sp.setVmax(440);
        sp.setPrefSize(115, 150);
        sp.setContent(vb);
        sp.vvalueProperty().addListener((ObservableValue<? extends Number> ov,
            Number old_val, Number new_val) -> {
                fileName.setText(imageNames[(new_val.intValue() - 1)/100]);
        });
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

ListViewSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
```

```
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package listviewsample;

import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ListCell;
import javafx.scene.control.ListView;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class ListViewSample extends Application {

    ListView<String> list = new ListView<>();
    ObservableList<String> data = FXCollections.observableArrayList(
            "chocolate", "salmon", "gold", "coral", "darkorchid",
            "darkgoldenrod", "lightsalmon", "black", "rosybrown", "blue",
            "blueviolet", "brown");
    final Label label = new Label();

    @Override
    public void start(Stage stage) {
        VBox box = new VBox();
        Scene scene = new Scene(box, 200, 200);
        stage.setScene(scene);
        stage.setTitle("ListViewSample");
        box.getChildren().addAll(list, label);
        VBox.setVgrow(list, Priority.ALWAYS);

        label.setLayoutX(10);
        label.setLayoutY(115);
        label.setFont(Font.font("Verdana", 20));

        list.setItems(data);

        list.setCellFactory((ListView<String> l) -> new ColorRectCell());

        list.getSelectionModel().selectedItemProperty().addListener(
            (ObservableValue<? extends String> ov, String old_val,
                String new_val) -> {
                    label.setText(new_val);
```

```
                    label.setTextFill(Color.web(new_val));
            });
            stage.show();
        }

        static class ColorRectCell extends ListCell<String> {
            @Override
            public void updateItem(String item, boolean empty) {
                super.updateItem(item, empty);
                Rectangle rect = new Rectangle(100, 20);
                if (item != null) {
                    rect.setFill(Color.web(item));
                    setGraphic(rect);
                } else {
                    setGraphic(null);
                }
            }
        }

        public static void main(String[] args) {
            launch(args);
        }
    }
```

TableViewSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package tableviewsample;
```

```
import javafx.application.Application;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableColumn.CellEditEvent;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.control.cell.TextFieldTableCell;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class TableViewSample extends Application {

    private final TableView<Person> table = new TableView<>();
    private final ObservableList<Person> data =
            FXCollections.observableArrayList(
            new Person("Jacob", "Smith", "jacob.smith@example.com"),
            new Person("Isabella", "Johnson", "isabella.johnson@example.com"),
            new Person("Ethan", "Williams", "ethan.williams@example.com"),
            new Person("Emma", "Jones", "emma.jones@example.com"),
            new Person("Michael", "Brown", "michael.brown@example.com"));
    final HBox hb = new HBox();

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Table View Sample");
        stage.setWidth(450);
        stage.setHeight(550);

        final Label label = new Label("Address Book");
        label.setFont(new Font("Arial", 20));

        table.setEditable(true);

        TableColumn<Person, String> firstNameCol =
            new TableColumn<>("First Name");
        firstNameCol.setMinWidth(100);
        firstNameCol.setCellValueFactory(
            new PropertyValueFactory<>("firstName"));

        firstNameCol.setCellFactory(TextFieldTableCell.<Person>forTableColumn());
        firstNameCol.setOnEditCommit(
            (CellEditEvent<Person, String> t) -> {
                ((Person) t.getTableView().getItems().get(
```

```
                t.getTablePosition().getRow())
                ).setFirstName(t.getNewValue());
});


TableColumn<Person, String> lastNameCol =
    new TableColumn<>("Last Name");
lastNameCol.setMinWidth(100);
lastNameCol.setCellValueFactory(
    new PropertyValueFactory<>("lastName"));
lastNameCol.setCellFactory(TextFieldTableCell.<Person>forTableColumn());
lastNameCol.setOnEditCommit(
    (CellEditEvent<Person, String> t) -> {
        ((Person) t.getTableView().getItems().get(
                t.getTablePosition().getRow())
                ).setLastName(t.getNewValue());
});

TableColumn<Person, String> emailCol = new TableColumn<>("Email");
emailCol.setMinWidth(200);
emailCol.setCellValueFactory(
    new PropertyValueFactory<>("email"));
emailCol.setCellFactory(TextFieldTableCell.<Person>forTableColumn());
emailCol.setOnEditCommit(
    (CellEditEvent<Person, String> t) -> {
        ((Person) t.getTableView().getItems().get(
                t.getTablePosition().getRow())
                ).setEmail(t.getNewValue());
});

table.setItems(data);
table.getColumns().addAll(firstNameCol, lastNameCol, emailCol);

final TextField addFirstName = new TextField();
addFirstName.setPromptText("First Name");
addFirstName.setMaxWidth(firstNameCol.getPrefWidth());
final TextField addLastName = new TextField();
addLastName.setMaxWidth(lastNameCol.getPrefWidth());
addLastName.setPromptText("Last Name");
final TextField addEmail = new TextField();
addEmail.setMaxWidth(emailCol.getPrefWidth());
addEmail.setPromptText("Email");

final Button addButton = new Button("Add");
addButton.setOnAction((ActionEvent e) -> {
    data.add(new Person(
            addFirstName.getText(),
            addLastName.getText(),
            addEmail.getText()));
    addFirstName.clear();
    addLastName.clear();
    addEmail.clear();
});

hb.getChildren().addAll(addFirstName, addLastName, addEmail, addButton);
hb.setSpacing(3);

final VBox vbox = new VBox();
vbox.setSpacing(5);
vbox.setPadding(new Insets(10, 0, 0, 10));
```

```
            vbox.getChildren().addAll(label, table, hb);

            ((Group) scene.getRoot()).getChildren().addAll(vbox);

            stage.setScene(scene);
            stage.show();
        }

        public static class Person {

            private final SimpleStringProperty firstName;
            private final SimpleStringProperty lastName;
            private final SimpleStringProperty email;

            private Person(String fName, String lName, String email) {
                this.firstName = new SimpleStringProperty(fName);
                this.lastName = new SimpleStringProperty(lName);
                this.email = new SimpleStringProperty(email);
            }

            public String getFirstName() {
                return firstName.get();
            }

            public void setFirstName(String fName) {
                firstName.set(fName);
            }

            public String getLastName() {
                return lastName.get();
            }

            public void setLastName(String fName) {
                lastName.set(fName);
            }

            public String getEmail() {
                return email.get();
            }

            public void setEmail(String fName) {
                email.set(fName);
            }
        }
    }
```

TreeViewSample.java

```
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package treeviewsample;

import java.util.Arrays;
import java.util.List;
import javafx.application.Application;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.control.TreeCell;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

import javafx.beans.property.SimpleStringProperty;
import javafx.event.ActionEvent;
import javafx.scene.control.ContextMenu;
import javafx.scene.control.MenuItem;
import javafx.scene.layout.VBox;

public class TreeViewSample extends Application {

    private final Node rootIcon =
        new ImageView(new Image(getClass().getResourceAsStream("root.png")));
    private final Image depIcon =
        new Image(getClass().getResourceAsStream("department.png"));
    List<Employee> employees = Arrays.<Employee>asList(
            new Employee("Jacob Smith", "Accounts Department"),
            new Employee("Isabella Johnson", "Accounts Department"),
            new Employee("Ethan Williams", "Sales Department"),
            new Employee("Emma Jones", "Sales Department"),
            new Employee("Michael Brown", "Sales Department"),
            new Employee("Anna Black", "Sales Department"),
            new Employee("Rodger York", "Sales Department"),
            new Employee("Susan Collins", "Sales Department"),
```

```
        new Employee("Mike Graham", "IT Support"),
        new Employee("Judy Mayer", "IT Support"),
        new Employee("Gregory Smith", "IT Support"));
TreeItem<String> rootNode =
    new TreeItem<>("MyCompany Human Resources", rootIcon);

public static void main(String[] args) {
    Application.launch(args);
}

@Override
public void start(Stage stage) {
    rootNode.setExpanded(true);
    for (Employee employee : employees) {
        TreeItem<String> empLeaf = new TreeItem<>(employee.getName());
        boolean found = false;
        for (TreeItem<String> depNode : rootNode.getChildren()) {
            if (depNode.getValue().contentEquals(employee.getDepartment())){
                depNode.getChildren().add(empLeaf);
                found = true;
                break;
            }
        }
        if (!found) {
            TreeItem depNode = new TreeItem(employee.getDepartment(),
                new ImageView(depIcon)
            );
            rootNode.getChildren().add(depNode);
            depNode.getChildren().add(empLeaf);
        }
    }

    stage.setTitle("Tree View Sample");
    VBox box = new VBox();
    final Scene scene = new Scene(box, 400, 300);
    scene.setFill(Color.LIGHTGRAY);

    TreeView<String> treeView = new TreeView<>(rootNode);
    treeView.setShowRoot(true);
    treeView.setEditable(true);
    treeView.setCellFactory((TreeView<String> p) ->
        new TextFieldTreeCellImpl());

    box.getChildren().add(treeView);
    stage.setScene(scene);
    stage.show();
}

private final class TextFieldTreeCellImpl extends TreeCell<String> {

    private TextField textField;
    private final ContextMenu addMenu = new ContextMenu();

    public TextFieldTreeCellImpl() {
        MenuItem addMenuItem = new MenuItem("Add Employee");
        addMenu.getItems().add(addMenuItem);
        addMenuItem.setOnAction((ActionEvent t) -> {
            TreeItem newEmployee =
                    new TreeItem<>("New Employee");
            getTreeItem().getChildren().add(newEmployee);
```

```
                });
            }

            @Override
            public void startEdit() {
                super.startEdit();

                if (textField == null) {
                    createTextField();
                }
                setText(null);
                setGraphic(textField);
                textField.selectAll();
            }

            @Override
            public void cancelEdit() {
                super.cancelEdit();

                setText((String) getItem());
                setGraphic(getTreeItem().getGraphic());
            }

            @Override
            public void updateItem(String item, boolean empty) {
                super.updateItem(item, empty);

                if (empty) {
                    setText(null);
                    setGraphic(null);
                } else {
                    if (isEditing()) {
                        if (textField != null) {
                            textField.setText(getString());
                        }
                        setText(null);
                        setGraphic(textField);
                    } else {
                        setText(getString());
                        setGraphic(getTreeItem().getGraphic());
                        if (!getTreeItem().isLeaf()&&getTreeItem().getParent()!=
null){
                            setContextMenu(addMenu);
                        }
                    }
                }
            }

            private void createTextField() {
                textField = new TextField(getString());
                textField.setOnKeyReleased((KeyEvent t) -> {
                    if (t.getCode() == KeyCode.ENTER) {
                        commitEdit(textField.getText());
                    } else if (t.getCode() == KeyCode.ESCAPE) {
                        cancelEdit();
                    }
                });

            }
```

```
        private String getString() {
            return getItem() == null ? "" : getItem().toString();
        }
    }

    public static class Employee {

        private final SimpleStringProperty name;
        private final SimpleStringProperty department;

        private Employee(String name, String department) {
            this.name = new SimpleStringProperty(name);
            this.department = new SimpleStringProperty(department);
        }

        public String getName() {
            return name.get();
        }

        public void setName(String fName) {
            name.set(fName);
        }

        public String getDepartment() {
            return department.get();
        }

        public void setDepartment(String fName) {
            department.set(fName);
        }
    }
}
```

TreeTableViewSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
```

```
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package treetableviewsample;

import java.util.Arrays;
import java.util.List;
import javafx.application.Application;
import javafx.beans.property.ReadOnlyStringWrapper;
import javafx.beans.property.SimpleStringProperty;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.SelectionMode;
import javafx.scene.control.TreeTableColumn;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeSortMode;
import javafx.scene.control.TreeTableView;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class TreeTableViewSample extends Application {

    List<Employee> employees = Arrays.<Employee>asList(
        new Employee("Ethan Williams", "ethan.williams@example.com"),
        new Employee("Emma Jones", "emma.jones@example.com"),
        new Employee("Michael Brown", "michael.brown@example.com"),
        new Employee("Anna Black", "anna.black@example.com"),
        new Employee("Rodger York", "roger.york@example.com"),
        new Employee("Susan Collins", "susan.collins@example.com"));
    private final ImageView depIcon = new ImageView (
            new Image(getClass().getResourceAsStream("department.png"))
    );

    final TreeItem<Employee> root =
        new TreeItem<>(new Employee("Sales Department", ""), depIcon);

    public static void main(String[] args) {
        Application.launch(TreeTableViewSample.class, args);
    }

    @Override
    public void start(Stage stage) {
        root.setExpanded(true);
        employees.stream().forEach((employee) -> {
            root.getChildren().add(new TreeItem<>(employee));
        });
        stage.setTitle("Tree Table View Sample");
        final Scene scene = new Scene(new Group(), 875, 700);
        scene.setFill(Color.LIGHTGRAY);
        Group sceneRoot = (Group) scene.getRoot();

        TreeTableColumn<Employee, String> empColumn =
            new TreeTableColumn<>("Employee");
```

```
        empColumn.setPrefWidth(150);
        empColumn.setCellValueFactory(
            (TreeTableColumn.CellDataFeatures<Employee, String> param) ->
            new ReadOnlyStringWrapper(param.getValue().getValue().getName())
        );
        TreeTableColumn<Employee, String> emailColumn =
            new TreeTableColumn<>("Email");
        emailColumn.setPrefWidth(190);
        emailColumn.setCellValueFactory(
            (TreeTableColumn.CellDataFeatures<Employee, String> param) ->
            new ReadOnlyStringWrapper(param.getValue().getValue().getEmail())
        );
        emailColumn.setSortType(TreeTableColumn.SortType.ASCENDING);

        TreeTableView<Employee> treeTableView = new TreeTableView<>(root);
        treeTableView.getColumns().setAll(empColumn, emailColumn);
        treeTableView.setSortMode(TreeSortMode.ALL_DESCENDANTS);

treeTableView.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
        treeTableView.getSelectionModel().setCellSelectionEnabled(true);
        sceneRoot.getChildren().add(treeTableView);

        stage.setScene(scene);
        stage.show();
    }

    public class Employee {

        private SimpleStringProperty name;
        private SimpleStringProperty email;
        public SimpleStringProperty nameProperty() {
            if (name == null) {
                name = new SimpleStringProperty(this, "name");
            }
            return name;
        }
        public SimpleStringProperty emailProperty() {
            if (email == null) {
                email = new SimpleStringProperty(this, "email");
            }
            return email;
        }
        private Employee(String name, String email) {
            this.name = new SimpleStringProperty(name);
            this.email = new SimpleStringProperty(email);
        }
        public String getName() {
            return name.get();
        }
        public void setName(String fName) {
            name.set(fName);
        }
        public String getEmail() {
            return email.get();
        }
        public void setEmail(String fName) {
            email.set(fName);
        }
    }
}
```

ComboBoxSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package comboboxsample;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.util.Callback;

public class ComboBoxSample extends Application {
        public static void main(String[] args) {
        launch(args);
    }

    final Button button = new Button ("Send");
    final Label notification = new Label ();
    final TextField subject = new TextField("");
    final TextArea text = new TextArea ("");
```

```
String address = " ";

@Override public void start(Stage stage) {
    stage.setTitle("ComboBoxSample");
    Scene scene = new Scene(new Group(), 500, 270);

    final ComboBox emailComboBox = new ComboBox();
    emailComboBox.getItems().addAll(
        "jacob.smith@example.com",
        "isabella.johnson@example.com",
        "ethan.williams@example.com",
        "emma.jones@example.com",
        "michael.brown@example.com"
    );

    emailComboBox.setPromptText("Email address");
    emailComboBox.setEditable(true);
    emailComboBox.setOnAction((Event ev) -> {
        address =
            emailComboBox.getSelectionModel().getSelectedItem().toString();
    });


    final ComboBox priorityComboBox = new ComboBox();
    priorityComboBox.getItems().addAll(
        "Highest",
        "High",
        "Normal",
        "Low",
        "Lowest"
    );
    priorityComboBox.setValue("Normal");
    priorityComboBox.setCellFactory(
        new Callback<ListView<String>, ListCell<String>>() {
            @Override public ListCell<String> call(ListView<String> param) {
                final ListCell<String> cell = new ListCell<String>() {
                    {
                        super.setPrefWidth(100);
                    }
                    @Override public void updateItem(String item,
                        boolean empty) {
                            super.updateItem(item, empty);
                            if (item != null) {
                                setText(item);
                            if (item.contains("High")) {
                                        setTextFill(Color.RED);
                                    }
                                    else if (item.contains("Low")){
                                        setTextFill(Color.GREEN);
                                    }
                                    else {
                                        setTextFill(Color.BLACK);
                                    }
                                }
                                else {
                                    setText(null);
                                }
                            }
                    };
```

```
                return cell;
            }

        });


        button.setOnAction((ActionEvent e) -> {
            if (emailComboBox.getValue() != null &&
                    !emailComboBox.getValue().toString().isEmpty()){
                notification.setText("Your message was successfully sent"
                        + " to " + address);
                emailComboBox.setValue(null);
                if (priorityComboBox.getValue() != null &&
                        !priorityComboBox.getValue().toString().isEmpty()){
                    priorityComboBox.setValue(null);
                }
                subject.clear();
                text.clear();
            }
            else {
                notification.setText("You have not selected a recipient!");
            }
        });

        GridPane grid = new GridPane();
        grid.setVgap(4);
        grid.setHgap(10);
        grid.setPadding(new Insets(5, 5, 5, 5));
        grid.add(new Label("To: "), 0, 0);
        grid.add(emailComboBox, 1, 0);
        grid.add(new Label("Priority: "), 2, 0);
        grid.add(priorityComboBox, 3, 0);
        grid.add(new Label("Subject: "), 0, 1);
        grid.add(subject, 1, 1, 3, 1);
        grid.add(text, 0, 2, 4, 1);
        grid.add(button, 0, 3);
        grid.add (notification, 1, 3, 3, 1);

        Group root = (Group)scene.getRoot();
        root.getChildren().add(grid);
        stage.setScene(scene);
        stage.show();

    }
}
```

SeparatorSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
```

```
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package separatorsample;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Orientation;
import javafx.geometry.VPos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.GridPane;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class SeparatorSample extends Application {

    Label caption = new Label("Weather Forecast");
    Label friday = new Label("Friday");
    Label saturday = new Label("Saturday");
    Label sunday = new Label("Sunday");

    @Override
    public void start(Stage stage) {
        Group root = new Group();
        Scene scene = new Scene(root, 500, 200);
        stage.setScene(scene);
        stage.setTitle("Separator Sample");

        GridPane grid = new GridPane();
        grid.setPadding(new Insets(10, 10, 10, 10));
        grid.setVgap(2);
        grid.setHgap(5);

        scene.setRoot(grid);
        scene.getStylesheets().add("separatorsample/controlStyle.css");

        Image cloudImage = new Image(
            getClass().getResourceAsStream("cloud.jpg")
```

```
);
Image sunImage = new Image(
    getClass().getResourceAsStream("sun.jpg")
);

caption.setFont(Font.font("Verdana", 20));

GridPane.setConstraints(caption, 0, 0);
GridPane.setColumnSpan(caption, 8);
grid.getChildren().add(caption);

final Separator sepHor = new Separator();
sepHor.setValignment(VPos.CENTER);
GridPane.setConstraints(sepHor, 0, 1);
GridPane.setColumnSpan(sepHor, 7);
grid.getChildren().add(sepHor);

friday.setFont(Font.font("Verdana", 18));
GridPane.setConstraints(friday, 0, 2);
GridPane.setColumnSpan(friday, 2);
grid.getChildren().add(friday);

final Separator sepVert1 = new Separator();
sepVert1.setOrientation(Orientation.VERTICAL);
sepVert1.setValignment(VPos.CENTER);
sepVert1.setPrefHeight(80);
GridPane.setConstraints(sepVert1, 2, 2);
GridPane.setRowSpan(sepVert1, 2);
grid.getChildren().add(sepVert1);

saturday.setFont(Font.font("Verdana", 18));
GridPane.setConstraints(saturday, 3, 2);
GridPane.setColumnSpan(saturday, 2);
grid.getChildren().add(saturday);

final Separator sepVert2 = new Separator();
sepVert2.setOrientation(Orientation.VERTICAL);
sepVert2.setValignment(VPos.CENTER);
sepVert2.setPrefHeight(80);
GridPane.setConstraints(sepVert2, 5, 2);
GridPane.setRowSpan(sepVert2, 2);
grid.getChildren().add(sepVert2);

sunday.setFont(Font.font("Verdana", 18));
GridPane.setConstraints(sunday, 6, 2);
GridPane.setColumnSpan(sunday, 2);
grid.getChildren().add(sunday);

final ImageView cloud = new ImageView(cloudImage);
GridPane.setConstraints(cloud, 0, 3);
grid.getChildren().add(cloud);

final Label t1 = new Label("16");
t1.setFont(Font.font("Verdana", 20));
GridPane.setConstraints(t1, 1, 3);
grid.getChildren().add(t1);

final ImageView sun1 = new ImageView(sunImage);
GridPane.setConstraints(sun1, 3, 3);
grid.getChildren().add(sun1);
```

```
        final Label t2 = new Label("18");
        t2.setFont(Font.font("Verdana", 20));
        GridPane.setConstraints(t2, 4, 3);
        grid.getChildren().add(t2);

        final ImageView sun2 = new ImageView(sunImage);
        GridPane.setConstraints(sun2, 6, 3);
        grid.getChildren().add(sun2);

        final Label t3 = new Label("20");
        t3.setFont(Font.font("Verdana", 20));
        GridPane.setConstraints(t3, 7, 3);
        grid.getChildren().add(t3);

        stage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

SliderSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package slidersample;

import javafx.application.Application;
```

```
import javafx.beans.value.ObservableValue;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.Slider;
import javafx.scene.effect.SepiaTone;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.GridPane;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class SliderSample extends Application {

    final Slider opacityLevel = new Slider(0, 1, 1);
    final Slider sepiaTone = new Slider(0, 1, 1);
    final Slider scaling = new Slider (0.5, 1, 1);
    final Image image  = new Image(getClass().getResourceAsStream(
        "cappuccino.jpg")
    );

    final Label opacityCaption = new Label("Opacity Level:");
    final Label sepiaCaption = new Label("Sepia Tone:");
    final Label scalingCaption = new Label("Scaling Factor:");

    final Label opacityValue = new Label(
        Double.toString(opacityLevel.getValue()));
    final Label sepiaValue = new Label(
        Double.toString(sepiaTone.getValue()));
    final Label scalingValue = new Label(
        Double.toString(scaling.getValue()));

    final static Color textColor = Color.BLACK;
    final static SepiaTone sepiaEffect = new SepiaTone();

    @Override
    public void start(Stage stage) {
        Group root = new Group();
        Scene scene = new Scene(root, 600, 400);
        stage.setScene(scene);
        stage.setTitle("Slider Sample");

        GridPane grid = new GridPane();
        grid.setPadding(new Insets(10, 10, 10, 10));
        grid.setVgap(10);
        grid.setHgap(70);

        final ImageView cappuccino = new ImageView (image);
        cappuccino.setEffect(sepiaEffect);
        GridPane.setConstraints(cappuccino, 0, 0);
        GridPane.setColumnSpan(cappuccino, 3);
        grid.getChildren().add(cappuccino);
        scene.setRoot(grid);

        opacityCaption.setTextFill(textColor);
        GridPane.setConstraints(opacityCaption, 0, 1);
        grid.getChildren().add(opacityCaption);
```

```
            opacityLevel.valueProperty().addListener((
                ObservableValue<? extends Number> ov,
                Number old_val, Number new_val) -> {
                    cappuccino.setOpacity(new_val.doubleValue());
                    opacityValue.setText(String.format("%.2f", new_val));
            });

            GridPane.setConstraints(opacityLevel, 1, 1);
            grid.getChildren().add(opacityLevel);

            opacityValue.setTextFill(textColor);
            GridPane.setConstraints(opacityValue, 2, 1);
            grid.getChildren().add(opacityValue);

            sepiaCaption.setTextFill(textColor);
            GridPane.setConstraints(sepiaCaption, 0, 2);
            grid.getChildren().add(sepiaCaption);

            sepiaTone.valueProperty().addListener((
                ObservableValue<? extends Number> ov, Number old_val,
                Number new_val) -> {
                    sepiaEffect.setLevel(new_val.doubleValue());
                    sepiaValue.setText(String.format("%.2f", new_val));
            });
            GridPane.setConstraints(sepiaTone, 1, 2);
            grid.getChildren().add(sepiaTone);

            sepiaValue.setTextFill(textColor);
            GridPane.setConstraints(sepiaValue, 2, 2);
            grid.getChildren().add(sepiaValue);

            scalingCaption.setTextFill(textColor);
            GridPane.setConstraints(scalingCaption, 0, 3);
            grid.getChildren().add(scalingCaption);

            scaling.valueProperty().addListener((
                ObservableValue<? extends Number> ov, Number old_val,
                Number new_val) -> {
                    cappuccino.setScaleX(new_val.doubleValue());
                    cappuccino.setScaleY(new_val.doubleValue());
                    scalingValue.setText(String.format("%.2f", new_val));
            });
            GridPane.setConstraints(scaling, 1, 3);
            grid.getChildren().add(scaling);

            scalingValue.setTextFill(textColor);
            GridPane.setConstraints(scalingValue, 2, 3);
            grid.getChildren().add(scalingValue);

            stage.show();
        }

        public static void main(String[] args) {
            launch(args);
        }
    }
```

ProgressSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package progresssample;

import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.ProgressBar;
import javafx.scene.control.ProgressIndicator;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class ProgressSample  extends Application {

final Float[] values = new Float[] {-1.0f, 0f, 0.6f, 1.0f};
final Label [] labels = new Label[values.length];
final ProgressBar[] pbs = new ProgressBar[values.length];
final ProgressIndicator[] pins = new ProgressIndicator[values.length];
final HBox hbs [] = new HBox [values.length];

    @Override
    public void start(Stage stage) {
        Group root = new Group();
        Scene scene = new Scene(root, 300, 250);
        stage.setScene(scene);
```

```
        stage.setTitle("Progress Controls");

        for (int i = 0; i < values.length; i++) {
            final Label label = labels[i] = new Label();
            label.setText("progress:" + values[i]);

            final ProgressBar pb = pbs[i] = new ProgressBar();
            pb.setProgress(values[i]);

            final ProgressIndicator pin = pins[i] = new ProgressIndicator();
            pin.setProgress(values[i]);
            final HBox hb = hbs[i] = new HBox();
            hb.setSpacing(5);
            hb.setAlignment(Pos.CENTER);
            hb.getChildren().addAll(label, pb, pin);
        }

        final VBox vb = new VBox();
        vb.setSpacing(5);
        vb.getChildren().addAll(hbs);
        scene.setRoot(vb);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

HyperlinkSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
```

```
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package hyperlinksample;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class HyperlinkSample extends Application {

    final static String[] imageFiles = new String[]{
        "product.png",
        "education.png",
        "partners.png",
        "support.png"
    };
    final static String[] captions = new String[]{
        "Products",
        "Education",
        "Partners",
        "Support"
    };
    final ImageView selectedImage = new ImageView();
    final ScrollPane list = new ScrollPane();
    final Hyperlink[] hpls = new Hyperlink[captions.length];
    final Image[] images = new Image[imageFiles.length];

    public static void main(String[] args) {
        launch(HyperlinkSample.class, args);
    }

    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Hyperlink Sample");
        stage.setWidth(300);
        stage.setHeight(200);

        selectedImage.setLayoutX(100);
        selectedImage.setLayoutY(10);

        for (int i = 0; i < captions.length; i++) {
            final Hyperlink hpl = hpls[i] = new Hyperlink(captions[i]);
            final Image image = images[i] =
                new Image(getClass().getResourceAsStream(imageFiles[i]));
            hpl.setOnAction((ActionEvent e) -> {
                selectedImage.setImage(image);
            });
        }

        final Button button = new Button("Refresh links");
        button.setOnAction((ActionEvent e) -> {
            for (int i = 0; i < captions.length; i++) {
```

```
                hpls[i].setVisited(false);
                selectedImage.setImage(null);
            }
        });

        VBox vbox = new VBox();
        vbox.getChildren().addAll(hpls);
        vbox.getChildren().add(button);
        vbox.setSpacing(5);

        ((Group) scene.getRoot()).getChildren().addAll(vbox, selectedImage);
        stage.setScene(scene);
        stage.show();
    }
}
```

HyperlinkWebViewSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package hyperlinkwebviewsample;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Pos;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
```

```
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
import javafx.stage.Stage;

public class HyperlinkWebViewSample extends Application {

    final static String[] imageFiles = new String[]{
        "product.png",
        "education.png",
        "partners.png",
        "support.png"
    };
    final static String[] captions = new String[]{
        "Products",
        "Education",
        "Partners",
        "Support"
    };

    final static String[] urls = new String[]{
        "http://www.oracle.com/us/products/index.html",
        "http://education.oracle.com/",
        "http://www.oracle.com/partners/index.html",
        "http://www.oracle.com/us/support/index.html"
    };

    final ImageView selectedImage = new ImageView();
    final Hyperlink[] hpls = new Hyperlink[captions.length];
    final Image[] images = new Image[imageFiles.length];

    public static void main(String[] args){
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        VBox vbox = new VBox();
        Scene scene = new Scene(vbox);
        stage.setTitle("Hyperlink Sample");
        stage.setWidth(570);
        stage.setHeight(550);

        selectedImage.setLayoutX(100);
        selectedImage.setLayoutY(10);

        final WebView browser = new WebView();
        final WebEngine webEngine = browser.getEngine();

        for (int i = 0; i < captions.length; i++) {
            final Hyperlink hpl = hpls[i] = new Hyperlink(captions[i]);
            final Image image = images[i] =
                new Image(getClass().getResourceAsStream(imageFiles[i]));
            hpl.setGraphic(new ImageView (image));
            hpl.setFont(Font.font("Arial", 14));
            final String url = urls[i];
```

```
            hpl.setOnAction((ActionEvent e) -> {
                webEngine.load(url);
            });
        }

        HBox hbox = new HBox();
        hbox.setAlignment(Pos.BASELINE_CENTER);
        hbox.getChildren().addAll(hpls);
        vbox.getChildren().addAll(hbox, browser);
        VBox.setVgrow(browser, Priority.ALWAYS);

        stage.setScene(scene);
        stage.show();
    }
}
```

HTMLEditorSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package htmleditorsample;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.*;
```

```
import javafx.scene.layout.VBox;
import javafx.scene.web.HTMLEditor;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebView;
import javafx.stage.Stage;

public class HTMLEditorSample extends Application {
    private final String INITIAL_TEXT = "Lorem ipsum dolor sit "
            + "amet, consectetur adipiscing elit. Nam tortor felis, pulvinar "
            + "in scelerisque cursus, pulvinar at ante. Nulla consequat"
            + "congue lectus in sodales. Nullam eu est a felis ornare "
            + "bibendum et nec tellus. Vivamus non metus tempus augue auctor "
            + "ornare. Duis pulvinar justo ac purus adipiscing pulvinar. "
            + "Integer congue faucibus dapibus. Integer id nisl ut elit "
            + "aliquam sagittis gravida eu dolor. Etiam sit amet ipsum "
            + "sem.";

    @Override
    public void start(Stage stage) {
        stage.setTitle("HTMLEditor Sample");
        stage.setWidth(650);
        stage.setHeight(500);
        Scene scene = new Scene(new Group());

        VBox root = new VBox();
        root.setPadding(new Insets(8, 8, 8, 8));
        root.setSpacing(5);
        root.setAlignment(Pos.BOTTOM_LEFT);

        final HTMLEditor htmlEditor = new HTMLEditor();
        htmlEditor.setPrefHeight(245);
        htmlEditor.setHtmlText(INITIAL_TEXT);

        final WebView browser = new WebView();
        final WebEngine webEngine = browser.getEngine();


        ScrollPane scrollPane = new ScrollPane();
        scrollPane.getStyleClass().add("noborder-scroll-pane");
        scrollPane.setStyle("-fx-background-color: white");
        scrollPane.setContent(browser);
        scrollPane.setFitToWidth(true);
        scrollPane.setPrefHeight(180);

        Button showHTMLButton = new Button("Load Content in Browser");
        root.setAlignment(Pos.CENTER);
        showHTMLButton.setOnAction((ActionEvent arg0) -> {
            webEngine.loadContent(htmlEditor.getHtmlText());
        });

        root.getChildren().addAll(htmlEditor, showHTMLButton, scrollPane);
        scene.setRoot(root);

        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
```

```
}
```

TooltipSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package tooltipsample;

import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.control.Label;
import javafx.scene.control.Tooltip;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;


public class TooltipSample extends Application {

    final static String[] rooms = new String[]{
        "Accommodation (BB)",
        "Half Board",
        "Late Check-out",
        "Extra Bed"
    };
```

```
    final static Integer[] rates = new Integer[]{
        100, 20, 10, 30
    };
    final CheckBox[] cbs = new CheckBox[rooms.length];
    final Label total = new Label("Total: $0");
    Integer sum = 0;

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());        stage.setTitle("Tooltip
Sample");
        stage.setWidth(330);
        stage.setHeight(150);

        total.setFont(new Font("Arial", 20));

        for (int i = 0; i < rooms.length; i++) {
            final CheckBox cb = cbs[i] = new CheckBox(rooms[i]);
            final Integer rate = rates[i];
            final Tooltip tooltip = new Tooltip("$" + rates[i].toString());
            tooltip.setFont(new Font("Arial", 16));
            cb.setTooltip(tooltip);
    cb.selectedProperty().addListener(
                (ObservableValue<? extends Boolean> ov, Boolean old_val,
                Boolean new_val) -> {
                    if (cb.isSelected()) {
                        sum = sum + rate;
                    } else {
                        sum = sum - rate;
                    }
                    total.setText("Total: $" + sum.toString());
                }
            );
        }

        VBox vbox = new VBox();
        vbox.getChildren().addAll(cbs);
        vbox.setSpacing(5);
        HBox root = new HBox();
        root.getChildren().add(vbox);
        root.getChildren().add(total);
        root.setSpacing(40);
        root.setPadding(new Insets(20, 10, 10, 20));

        ((Group) scene.getRoot()).getChildren().add(root);

        stage.setScene(scene);
        stage.show();
    }
}
```

TitledPaneSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
```

```
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package titledpanesample;

import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Accordion;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.control.TitledPane;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;

public class TitledPaneSample extends Application {
    final String[] imageNames = new String[]{"Apples", "Flowers", "Leaves"};
    final Image[] images = new Image[imageNames.length];
    final ImageView[] pics = new ImageView[imageNames.length];
    final TitledPane[] tps = new TitledPane[imageNames.length];
    final Label label = new Label("N/A");

    public static void main(String[] args) {
        launch(args);
    }

    @Override public void start(Stage stage) {
        stage.setTitle("TitledPane");
```

```
        Scene scene = new Scene(new Group(), 450, 250);

        TitledPane gridTitlePane = new TitledPane();
        GridPane grid = new GridPane();
        grid.setVgap(4);
        grid.setPadding(new Insets(5, 5, 5, 5));
        grid.add(new Label("First Name: "), 0, 0);
        grid.add(new TextField(), 1, 0);
        grid.add(new Label("Last Name: "), 0, 1);
        grid.add(new TextField(), 1, 1);
        grid.add(new Label("Email: "), 0, 2);
        grid.add(new TextField(), 1, 2);
        grid.add(new Label("Attachment: "), 0, 3);
        grid.add(label,1, 3);
        gridTitlePane.setText("Grid");
        gridTitlePane.setContent(grid);

        final Accordion accordion = new Accordion ();

        for (int i = 0; i < imageNames.length; i++) {
            images[i] =
                new Image(getClass().getResourceAsStream(imageNames[i]+".jpg"));
            pics[i] = new ImageView(images[i]);
            tps[i] = new TitledPane(imageNames[i],pics[i]);
        }
        accordion.getPanes().addAll(tps);


        accordion.expandedPaneProperty().addListener(
            (ObservableValue<? extends TitledPane> ov, TitledPane old_val,
            TitledPane new_val) -> {
                if (new_val != null) {
                    label.setText(accordion.getExpandedPane().getText()
                            + ".jpg");
                }
        });

        HBox hbox = new HBox(10);
        hbox.setPadding(new Insets(20, 0, 0, 20));
        hbox.getChildren().setAll(gridTitlePane, accordion);

        Group root = (Group)scene.getRoot();
        root.getChildren().add(hbox);
        stage.setScene(scene);
        stage.show();
    }
}
```

MenuSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
```

```
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package menusample;

import java.util.AbstractMap.SimpleEntry;
import java.util.Map.Entry;
import javafx.application.Application;
import javafx.beans.value.ObservableValue;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.effect.DropShadow;
import javafx.scene.effect.Effect;
import javafx.scene.effect.Glow;
import javafx.scene.effect.SepiaTone;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.*;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.TextAlignment;
import javafx.stage.Stage;

public class MenuSample extends Application {

    final PageData[] pages = new PageData[] {
        new PageData("Apple",
            "The apple is the pomaceous fruit of the apple tree, species Malus "
            + "domestica in the rose family (Rosaceae). It is one of the most "
            + "widely cultivated tree fruits, and the most widely known of "
            + "the many members of genus Malus that are used by humans. "
            + "The tree originated in Western Asia, where its wild ancestor, "
            + "the Alma, is still found today.",
            "Malus domestica"),
        new PageData("Hawthorn",
```

```
            "The hawthorn is a large genus of shrubs and trees in the rose
family,"
            + "Rosaceae, native to temperate regions of the Northern Hemisphere "
            + "in Europe, Asia and North America. The name hawthorn was "
            + "originally applied to the species native to northern Europe, "
            + "especially the Common Hawthorn C. monogyna, and the unmodified "
            + "name is often so used in Britain and Ireland.",
            "Crataegus monogyna"),
        new PageData("Ivy",
            "The ivy is a flowering plant in the grape family (Vitaceae) native to
"
            + " eastern Asia in Japan, Korea, and northern and eastern China. "
            + "It is a deciduous woody vine growing to 30 m tall or more given "
            + "suitable support,  attaching itself by means of numerous small "
            + "branched tendrils tipped with sticky disks.",
            "Parthenocissus tricuspidata"),
        new PageData("Quince",
            "The quince is the sole member of the genus Cydonia and is native to "
            + "warm-temperate southwest Asia in the Caucasus region. The "
            + "immature fruit is green with dense grey-white pubescence, most "
            + "of which rubs off before maturity in late autumn when the fruit "
            + "changes color to yellow with hard, strongly perfumed flesh.",
            "Cydonia oblonga")
    };

    final String[] viewOptions = new String[] {
        "Title",
        "Binomial name",
        "Picture",
        "Description"
    };

    final Entry<String, Effect>[] effects = new Entry[] {
        new SimpleEntry<>("Sepia Tone", new SepiaTone()),
        new SimpleEntry<>("Glow", new Glow()),
        new SimpleEntry<>("Shadow", new DropShadow())
    };

    final ImageView pic = new ImageView();
    final Label name = new Label();
    final Label binName = new Label();
    final Label description = new Label();
    private int currentIndex = -1;

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        stage.setTitle("Menu Sample");
        Scene scene = new Scene(new VBox(), 400, 350);
        scene.setFill(Color.OLDLACE);

        name.setFont(new Font("Verdana Bold", 22));
        binName.setFont(new Font("Arial Italic", 10));
        pic.setFitHeight(150);
        pic.setPreserveRatio(true);
        description.setWrapText(true);
        description.setTextAlignment(TextAlignment.JUSTIFY);
```

```
shuffle();

MenuBar menuBar = new MenuBar();

// --- Graphical elements
final VBox vbox = new VBox();
vbox.setAlignment(Pos.CENTER);
vbox.setSpacing(10);
vbox.setPadding(new Insets(0, 10, 0, 10));
vbox.getChildren().addAll(name, binName, pic, description);

// --- Menu File
Menu menuFile = new Menu("File");
MenuItem add = new MenuItem("Shuffle",
    new ImageView(new Image("menusample/new.png")));
add.setOnAction((ActionEvent t) -> {
    shuffle();
    vbox.setVisible(true);
});

MenuItem clear = new MenuItem("Clear");
clear.setAccelerator(KeyCombination.keyCombination("Ctrl+X"));
clear.setOnAction((ActionEvent t) -> {
    vbox.setVisible(false);
});

MenuItem exit = new MenuItem("Exit");
exit.setOnAction((ActionEvent t) -> {
    System.exit(0);
});

menuFile.getItems().addAll(add, clear, new SeparatorMenuItem(), exit);

// --- Menu Edit
Menu menuEdit = new Menu("Edit");
Menu menuEffect = new Menu("Picture Effect");

final ToggleGroup groupEffect = new ToggleGroup();
for (Entry<String, Effect> effect : effects) {
    RadioMenuItem itemEffect = new RadioMenuItem(effect.getKey());
    itemEffect.setUserData(effect.getValue());
    itemEffect.setToggleGroup(groupEffect);
    menuEffect.getItems().add(itemEffect);
}

final MenuItem noEffects = new MenuItem("No Effects");
noEffects.setDisable(true);
noEffects.setOnAction((ActionEvent t) -> {
    pic.setEffect(null);
    groupEffect.getSelectedToggle().setSelected(false);
    noEffects.setDisable(true);
});

groupEffect.selectedToggleProperty().addListener(
    (ObservableValue<? extends Toggle> ov, Toggle old_toggle,
    Toggle new_toggle) -> {
        if (groupEffect.getSelectedToggle() != null) {
            Effect effect =
                    (Effect)
```

```
groupEffect.getSelectedToggle().getUserData();
                    pic.setEffect(effect);
                    noEffects.setDisable(false);
                } else {
                    noEffects.setDisable(true);
                }
        });

        menuEdit.getItems().addAll(menuEffect, noEffects);

        // --- Menu View
        Menu menuView = new Menu("View");
        CheckMenuItem titleView = createMenuItem ("Title", name);
        CheckMenuItem binNameView = createMenuItem ("Binomial name", binName);
        CheckMenuItem picView = createMenuItem ("Picture", pic);
        CheckMenuItem descriptionView = createMenuItem (
            "Decsription", description);

        menuView.getItems().addAll(titleView, binNameView, picView,
            descriptionView);
        menuBar.getMenus().addAll(menuFile, menuEdit, menuView);


        // --- Context Menu
        final ContextMenu cm = new ContextMenu();
        MenuItem cmItem1 = new MenuItem("Copy Image");
        cmItem1.setOnAction((ActionEvent e) -> {
            Clipboard clipboard = Clipboard.getSystemClipboard();
            ClipboardContent content = new ClipboardContent();
            content.putImage(pic.getImage());
            clipboard.setContent(content);
        });

        cm.getItems().add(cmItem1);
        pic.addEventHandler(MouseEvent.MOUSE_CLICKED, (MouseEvent e) -> {
            if (e.getButton() == MouseButton.SECONDARY)
                cm.show(pic, e.getScreenX(), e.getScreenY());
        });

        ((VBox) scene.getRoot()).getChildren().addAll(menuBar, vbox);

        stage.setScene(scene);
        stage.show();
    }

    private void shuffle() {
        int i = currentIndex;
        while (i == currentIndex) {
            i = (int) (Math.random() * pages.length);
        }
        pic.setImage(pages[i].image);
        name.setText(pages[i].name);
        binName.setText("(" + pages[i].binNames + ")");
        description.setText(pages[i].description);
        currentIndex = i;
    }

    private static CheckMenuItem createMenuItem (String title, final Node node){
        CheckMenuItem cmi = new CheckMenuItem(title);
        cmi.setSelected(true);
```

```
        cmi.selectedProperty().addListener(
            (ObservableValue<? extends Boolean> ov, Boolean old_val,
            Boolean new_val) -> {
                node.setVisible(new_val);
        });
        return cmi;
    }

    private class PageData {
        public String name;
        public String description;
        public String binNames;
        public Image image;
        public PageData(String name, String description, String binNames) {
            this.name = name;
            this.description = description;
            this.binNames = binNames;
            image = new Image(getClass().getResourceAsStream(name + ".jpg"));
        }
    }
}
```

ColorPickerSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package colorpickersample;

import javafx.application.Application;
```

```
import javafx.scene.Scene;
import javafx.scene.control.ColorPicker;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.event.ActionEvent;
import javafx.scene.control.ComboBox;
import javafx.scene.control.ToolBar;
import javafx.scene.effect.DropShadow;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.SVGPath;
import javafx.stage.Stage;

public class ColorPickerSample extends Application {

    ImageView logo = new ImageView(
        new Image(getClass().getResourceAsStream("OracleLogo.png"))
    );

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        stage.setTitle("ColorPickerSample");

        Scene scene = new Scene(new VBox(20), 300, 300);
        //scene.getStylesheets().add("colorpickersample/ControlStyle.css");
        scene.setFill(Color.web("#ccffcc"));
        VBox box = (VBox) scene.getRoot();

        ToolBar tb = new ToolBar();
        box.getChildren().add(tb);

        final ComboBox logoSamples = new ComboBox();
        logoSamples.setPromptText("Logo");
        logoSamples.setValue("Oracle");
        logoSamples.getItems().addAll(
                "Oracle",
                "Java",
                "JavaFX",
                "Cup");

        logoSamples.valueProperty().addListener(new ChangeListener<String>() {
            @Override
          public void changed(ObservableValue ov, String t, String t1) {
                logo.setImage(
                    new Image(getClass().getResourceAsStream(t1+"Logo.png"))
                );
            }
        });


        final ColorPicker colorPicker = new ColorPicker();
        colorPicker.setValue(Color.CORAL);
        tb.getItems().addAll(logoSamples, colorPicker);
```

```
        StackPane stack = new StackPane();
        box.getChildren().add(stack);

        final SVGPath svg = new SVGPath();
        svg.setContent("M70,50 L90,50 L120,90 L150,50 L170,50"
            + "L210,90 L180,120 L170,110 L170,200 L70,200 L70,110 L60,120 L30,90"
            + "L70,50");
        svg.setStroke(Color.DARKGREY);
        svg.setStrokeWidth(2);
        svg.setEffect(new DropShadow());
        svg.setFill(colorPicker.getValue());
        stack.getChildren().addAll(svg, logo);

        colorPicker.setOnAction((ActionEvent t) -> {
            svg.setFill(colorPicker.getValue());
        });

        stage.setScene(scene);
        stage.show();
    }
}
```

DatePickerSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package datepickersample;
```

```
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;
import java.util.Locale;
import javafx.application.Application;
import javafx.geometry.HPos;
import javafx.scene.Scene;
import javafx.scene.control.DateCell;
import javafx.scene.control.DatePicker;
import javafx.scene.control.Label;
import javafx.scene.control.Tooltip;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import javafx.util.Callback;

import java.time.chrono.*;

public class DatePickerSample extends Application {

    private Stage stage;
    private DatePicker checkInDatePicker;
    private DatePicker checkOutDatePicker;

    public static void main(String[] args) {
        Locale.setDefault(Locale.US);
        launch(args);
    }

    @Override
    public void start(Stage stage) {
        this.stage = stage;
        stage.setTitle("DatePickerSample ");
        initUI();
        stage.show();
    }

    private void initUI() {
        VBox vbox = new VBox(20);
        vbox.setStyle("-fx-padding: 10;");
        Scene scene = new Scene(vbox, 400, 400);
        stage.setScene(scene);

        checkInDatePicker = new DatePicker();
        checkOutDatePicker = new DatePicker();
        checkInDatePicker.setValue(LocalDate.now());

        final Callback<DatePicker, DateCell> dayCellFactory =
            new Callback<DatePicker, DateCell>() {
                @Override
                public DateCell call(final DatePicker datePicker) {
                    return new DateCell() {
                        @Override
                        public void updateItem(LocalDate item, boolean empty) {
                            super.updateItem(item, empty);

                            if (item.isBefore(
                                    checkInDatePicker.getValue().plusDays(1))
                                ) {
                                setDisable(true);
                    setStyle("-fx-background-color: #ffc0cb;");
```

```
                                }
                                long p = ChronoUnit.DAYS.between(
                                        checkInDatePicker.getValue(), item
                                );
                                setTooltip(new Tooltip(
                                    "You're about to stay for " + p + " days")
                                );
                            }
                        };
                    }
                };

                checkOutDatePicker.setDayCellFactory(dayCellFactory);
                checkOutDatePicker.setValue(checkInDatePicker.getValue().plusDays(1));
                checkInDatePicker.setChronology(ThaiBuddhistChronology.INSTANCE);
                checkOutDatePicker.setChronology(HijrahChronology.INSTANCE);

                GridPane gridPane = new GridPane();
                gridPane.setHgap(10);
                gridPane.setVgap(10);

                Label checkInlabel = new Label("Check-In Date:");
                gridPane.add(checkInlabel, 0, 0);
                GridPane.setHalignment(checkInlabel, HPos.LEFT);

                gridPane.add(checkInDatePicker, 0, 1);

                Label checkOutlabel = new Label("Check-Out Date:");
                gridPane.add(checkOutlabel, 0, 2);
                GridPane.setHalignment(checkOutlabel, HPos.LEFT);

                gridPane.add(checkOutDatePicker, 0, 3);

                vbox.getChildren().add(gridPane);

            }
        }
```

PaginationSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
```

```
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package paginationsample;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Pagination;
import javafx.scene.control.Label;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class PaginationSample extends Application {

    private Pagination pagination;
    String[] fonts = new String[]{};

    public static void main(String[] args) throws Exception {
        launch(args);
    }

    public int itemsPerPage() {
        return 15;
    }

    public VBox createPage(int pageIndex) {
        VBox box = new VBox(5);
        int page = pageIndex * itemsPerPage();
        for (int i = page; i < page + itemsPerPage(); i++) {
            Label font = new Label(fonts[i]);
            box.getChildren().add(font);
        }
        return box;
    }

    @Override
    public void start(final Stage stage) throws Exception {
        fonts = Font.getFamilies().toArray(fonts);

        pagination = new Pagination(fonts.length/itemsPerPage(), 0);
        pagination.getStyleClass().add(Pagination.STYLE_CLASS_BULLET);
        pagination.setPageFactory((Integer pageIndex) -> createPage(pageIndex));
        AnchorPane anchor = new AnchorPane();
        AnchorPane.setTopAnchor(pagination, 10.0);
        AnchorPane.setRightAnchor(pagination, 10.0);
        AnchorPane.setBottomAnchor(pagination, 10.0);
        AnchorPane.setLeftAnchor(pagination, 10.0);
        anchor.getChildren().addAll(pagination);
```

```
        Scene scene = new Scene(anchor);
        stage.setScene(scene);
        stage.setTitle("PaginationSample");
        scene.getStylesheets().add("paginationsample/ControlStyle.css");
        stage.show();
    }
}
```

FileChooserSample.java

```
/*
 * Copyright (c) 2012, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package filechoosersample;

import java.awt.Desktop;
import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
```

```
import javafx.stage.Stage;

public final class FileChooserSample extends Application {

    private final Desktop desktop = Desktop.getDesktop();

    @Override
    public void start(final Stage stage) {
        stage.setTitle("File Chooser Sample");

        final FileChooser fileChooser = new FileChooser();
        final Button openButton = new Button("Open a Picture...");
        final Button openMultipleButton = new Button("Open Pictures...");

        openButton.setOnAction(
                (final ActionEvent e) -> {
                    configureFileChooser(fileChooser);
                    File file = fileChooser.showOpenDialog(stage);
                    if (file != null) {
                        openFile(file);
                    }
                });

        openMultipleButton.setOnAction(
                (final ActionEvent e) -> {
                    configureFileChooser(fileChooser);
                    List<File> list
                    = fileChooser.showOpenMultipleDialog(stage);
                    if (list != null) {
                        list.stream().forEach((file) -> {
                            openFile(file);
                        });
                    }
                });

        final GridPane inputGridPane = new GridPane();

        GridPane.setConstraints(openButton, 0, 1);
        GridPane.setConstraints(openMultipleButton, 1, 1);
        inputGridPane.setHgap(6);
        inputGridPane.setVgap(6);
        inputGridPane.getChildren().addAll(openButton, openMultipleButton);

        final Pane rootGroup = new VBox(12);
        rootGroup.getChildren().addAll(inputGridPane);
        rootGroup.setPadding(new Insets(12, 12, 12, 12));

        stage.setScene(new Scene(rootGroup));
        stage.show();
    }

    public static void main(String[] args) {
        Application.launch(args);
    }

    private static void configureFileChooser(
            final FileChooser fileChooser) {
        fileChooser.setTitle("View Pictures");
        fileChooser.setInitialDirectory(
                new File(System.getProperty("user.home"))
```

```
        );
        fileChooser.getExtensionFilters().addAll(
                new FileChooser.ExtensionFilter("All Images", "*.*"),
                new FileChooser.ExtensionFilter("JPG", "*.jpg"),
                new FileChooser.ExtensionFilter("PNG", "*.png")
        );
    }

    private void openFile(File file) {
        try {
            desktop.open(file);
        } catch (IOException ex) {
            Logger.getLogger(FileChooserSample.class.getName()).log(
                    Level.SEVERE, null, ex
            );
        }
    }
}
```

B

Chart Samples

This appendix lists code for the following JavaFX samples:

- PieChartSample.java
- LineChartSample.java
- AreaChartSample.java
- BubbleChartSample.java
- ScatterChartSample.java
- BarChartSample.java

For the descriptions of this source files, see Working with JavaFX Charts.

PieChartSample.java

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
```

```
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package piechartsample;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.scene.chart.*;
import javafx.scene.Group;
import javafx.scene.control.Label;
import javafx.scene.input.MouseEvent;

public class PieChartSample extends Application {

    @Override
    public void start(Stage stage) {
        Scene scene = new Scene(new Group());
        stage.setTitle("Imported Fruits");
        stage.setWidth(500);
        stage.setHeight(500);

        ObservableList<PieChart.Data> pieChartData =
                FXCollections.observableArrayList(
                new PieChart.Data("Grapefruit", 13),
                new PieChart.Data("Oranges", 25),
                new PieChart.Data("Plums", 10),
                new PieChart.Data("Pears", 22),
                new PieChart.Data("Apples", 30));

        final PieChart chart = new PieChart(pieChartData);
        chart.setTitle("Imported Fruits");
        final Label caption = new Label("");
        caption.setTextFill(Color.DARKORANGE);
        caption.setStyle("-fx-font: 24 arial;");
        chart.getData().stream().forEach((data) ->{
            data.getNode().addEventHandler(MouseEvent.MOUSE_PRESSED,
            (MouseEvent e) -> {
                caption.setTranslateX(e.getSceneX());
                caption.setTranslateY(e.getSceneY());
                caption.setText(String.valueOf(data.getPieValue())
                        + "%");
            });
        });

        ((Group) scene.getRoot()).getChildren().addAll(chart, caption);
        stage.setScene(scene);
        //scene.getStylesheets().add("piechartsample/Chart.css");
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

LineChartSample.java

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package linechartsample;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.LineChart;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;


public class LineChartSample extends Application {

    @Override public void start(Stage stage) {
        stage.setTitle("Line Chart Sample");
        final CategoryAxis xAxis = new CategoryAxis();
        final NumberAxis yAxis = new NumberAxis();
         xAxis.setLabel("Month");
        final LineChart<String,Number> lineChart =
                new LineChart<>(xAxis,yAxis);

        lineChart.setTitle("Stock Monitoring, 2010");

        lineChart.setCreateSymbols(false);
        lineChart.setAlternativeRowFillVisible(false);
        XYChart.Series series1 = new XYChart.Series();
```

```
        series1.setName("Portfolio 1");

        series1.getData().add(new XYChart.Data("Jan", 23));
        series1.getData().add(new XYChart.Data("Feb", 14));
        series1.getData().add(new XYChart.Data("Mar", 15));
        series1.getData().add(new XYChart.Data("Apr", 24));
        series1.getData().add(new XYChart.Data("May", 34));
        series1.getData().add(new XYChart.Data("Jun", 36));
        series1.getData().add(new XYChart.Data("Jul", 22));
        series1.getData().add(new XYChart.Data("Aug", 45));
        series1.getData().add(new XYChart.Data("Sep", 43));
        series1.getData().add(new XYChart.Data("Oct", 17));
        series1.getData().add(new XYChart.Data("Nov", 29));
        series1.getData().add(new XYChart.Data("Dec", 25));

        XYChart.Series series2 = new XYChart.Series();
        series2.setName("Portfolio 2");
        series2.getData().add(new XYChart.Data("Jan", 33));
        series2.getData().add(new XYChart.Data("Feb", 34));
        series2.getData().add(new XYChart.Data("Mar", 25));
        series2.getData().add(new XYChart.Data("Apr", 44));
        series2.getData().add(new XYChart.Data("May", 39));
        series2.getData().add(new XYChart.Data("Jun", 16));
        series2.getData().add(new XYChart.Data("Jul", 55));
        series2.getData().add(new XYChart.Data("Aug", 54));
        series2.getData().add(new XYChart.Data("Sep", 48));
        series2.getData().add(new XYChart.Data("Oct", 27));
        series2.getData().add(new XYChart.Data("Nov", 37));
        series2.getData().add(new XYChart.Data("Dec", 29));

        XYChart.Series series3 = new XYChart.Series();
        series3.setName("Portfolio 3");
        series3.getData().add(new XYChart.Data("Jan", 44));
        series3.getData().add(new XYChart.Data("Feb", 35));
        series3.getData().add(new XYChart.Data("Mar", 36));
        series3.getData().add(new XYChart.Data("Apr", 33));
        series3.getData().add(new XYChart.Data("May", 31));
        series3.getData().add(new XYChart.Data("Jun", 26));
        series3.getData().add(new XYChart.Data("Jul", 22));
        series3.getData().add(new XYChart.Data("Aug", 25));
        series3.getData().add(new XYChart.Data("Sep", 43));
        series3.getData().add(new XYChart.Data("Oct", 44));
        series3.getData().add(new XYChart.Data("Nov", 45));
        series3.getData().add(new XYChart.Data("Dec", 44));

        Scene scene  = new Scene(lineChart);
        lineChart.getData().addAll(series1, series2, series3);
       //scene.getStylesheets().add("linechartsample/Chart.css");
        stage.setScene(scene);
        stage.show();
    }


    public static void main(String[] args) {
        launch(args);
    }
}
```

AreaChartSample.java

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package areachartsample;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.AreaChart;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;

public class AreaChartSample extends Application {

    @Override public void start(Stage stage) {
        stage.setTitle("Area Chart Sample");
        final NumberAxis xAxis = new NumberAxis(1, 30, 1);
        final NumberAxis yAxis = new NumberAxis(-5, 27, 5);
        final AreaChart<Number,Number> ac =
            new AreaChart<>(xAxis,yAxis);
        xAxis.setForceZeroInRange(true);

        ac.setTitle("Temperature Monitoring (in Degrees C)");

        XYChart.Series series1 = new XYChart.Series();
        series1.setName("March");
        series1.getData().add(new XYChart.Data(0, -2));
        series1.getData().add(new XYChart.Data(3, -4));
        series1.getData().add(new XYChart.Data(6, 0));
```

```
            series1.getData().add(new XYChart.Data(9, 5));
            series1.getData().add(new XYChart.Data(12, -4));
            series1.getData().add(new XYChart.Data(15, 6));
            series1.getData().add(new XYChart.Data(18, 8));
            series1.getData().add(new XYChart.Data(21, 14));
            series1.getData().add(new XYChart.Data(24, 4));
            series1.getData().add(new XYChart.Data(27, 6));
            series1.getData().add(new XYChart.Data(30, 6));

            XYChart.Series series2 = new XYChart.Series();
            series2.setName("April");
            series2.getData().add(new XYChart.Data(0, 4));
            series2.getData().add(new XYChart.Data(3, 10));
            series2.getData().add(new XYChart.Data(6, 15));
            series2.getData().add(new XYChart.Data(9, 8));
            series2.getData().add(new XYChart.Data(12, 5));
            series2.getData().add(new XYChart.Data(15, 18));
            series2.getData().add(new XYChart.Data(18, 15));
            series2.getData().add(new XYChart.Data(21, 13));
            series2.getData().add(new XYChart.Data(24, 19));
            series2.getData().add(new XYChart.Data(27, 21));
            series2.getData().add(new XYChart.Data(30, 21));

            XYChart.Series series3 = new XYChart.Series();
            series3.setName("May");
            series3.getData().add(new XYChart.Data(0, 20));
            series3.getData().add(new XYChart.Data(3, 15));
            series3.getData().add(new XYChart.Data(6, 13));
            series3.getData().add(new XYChart.Data(9, 12));
            series3.getData().add(new XYChart.Data(12, 14));
            series3.getData().add(new XYChart.Data(15, 18));
            series3.getData().add(new XYChart.Data(18, 25));
            series3.getData().add(new XYChart.Data(21, 25));
            series3.getData().add(new XYChart.Data(24, 23));
            series3.getData().add(new XYChart.Data(27, 26));
            series3.getData().add(new XYChart.Data(30, 26));

            Scene scene  = new Scene(ac,800,600);
            //scene.getStylesheets().add("areachartsample/Chart.css");
            ac.setHorizontalZeroLineVisible(true);
            ac.getData().addAll(series1, series2, series3);
            stage.setScene(scene);
            stage.show();
        }

        public static void main(String[] args) {
            launch(args);
        }
    }
```

BubbleChartSample.java

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
```

```
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */


package bubblechartsample;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.BubbleChart;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.scene.paint.Color;
import javafx.stage.Stage;


public class BubbleChartSample extends Application {

    @Override public void start(Stage stage) {
        stage.setTitle("Bubble Chart Sample");
        final NumberAxis xAxis = new NumberAxis(1, 53, 4);
        final NumberAxis yAxis = new NumberAxis(0, 80, 10);
        final BubbleChart<Number,Number> blc =
            new BubbleChart<>(xAxis,yAxis);
        xAxis.setLabel("Week");
        xAxis.setTickLabelFill(Color.CHOCOLATE);
        xAxis.setMinorTickCount(4);

        yAxis.setLabel("Product Budget");
        yAxis.setTickLabelFill(Color.CHOCOLATE);
        yAxis.setTickLabelGap(10);
        yAxis.setTickLabelFormatter(
            new NumberAxis.DefaultFormatter(yAxis,"$ ",null)
        );

        blc.setTitle("Budget Monitoring");

        XYChart.Series series1 = new XYChart.Series();
        series1.setName("Product 1");
        series1.getData().add(new XYChart.Data(3, 35, 2));
```

```
        series1.getData().add(new XYChart.Data(12, 60, 1.8));
        series1.getData().add(new XYChart.Data(15, 15, 7));
        series1.getData().add(new XYChart.Data(22, 30, 2.5));
        series1.getData().add(new XYChart.Data(28, 20, 1));
        series1.getData().add(new XYChart.Data(35, 41, 5.5));
        series1.getData().add(new XYChart.Data(42, 17, 9));
        series1.getData().add(new XYChart.Data(49, 30, 1.8));

        XYChart.Series series2 = new XYChart.Series();
        series2.setName("Product 2");
        series2.getData().add(new XYChart.Data(8, 15, 2));
        series2.getData().add(new XYChart.Data(13, 23, 1));
        series2.getData().add(new XYChart.Data(15, 45, 3));
        series2.getData().add(new XYChart.Data(24, 30, 4.5));
        series2.getData().add(new XYChart.Data(38, 78, 1));
        series2.getData().add(new XYChart.Data(40, 41, 7.5));
        series2.getData().add(new XYChart.Data(45, 57, 2));
        series2.getData().add(new XYChart.Data(47, 23, 3.8));

        Scene scene  = new Scene(blc);
        blc.getData().addAll(series1, series2);
        //scene.getStylesheets().add("bubblechartsample/Chart.css");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

ScatterChartSample.java

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
```

```
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */


package scatterchartsample;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.ScatterChart;
import javafx.scene.chart.XYChart;
import javafx.scene.control.Button;
import javafx.scene.effect.DropShadow;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.stage.Stage;


public class ScatterChartSample extends Application {

    @Override public void start(Stage stage) {
        stage.setTitle("Scatter Chart Sample");
        final NumberAxis xAxis = new NumberAxis(0, 10, 1);
        final NumberAxis yAxis = new NumberAxis(-100, 500, 100);
        final ScatterChart<Number,Number> sc =
            new ScatterChart<>(xAxis,yAxis);
        xAxis.setLabel("Age (years)");
        yAxis.setLabel("Returns to date");
        sc.setTitle("Investment Overview");

        XYChart.Series series1 = new XYChart.Series();

        series1.setName("Option 1");
        series1.getData().add(new XYChart.Data(4.2, 193.2));
        series1.getData().add(new XYChart.Data(2.8, 33.6));
        series1.getData().add(new XYChart.Data(6.2, 24.8));
        series1.getData().add(new XYChart.Data(1, 14));
        series1.getData().add(new XYChart.Data(1.2, 26.4));
        series1.getData().add(new XYChart.Data(4.4, 114.4));
        series1.getData().add(new XYChart.Data(8.5, 323));
        series1.getData().add(new XYChart.Data(6.9, 289.8));
        series1.getData().add(new XYChart.Data(9.9, 287.1));
        series1.getData().add(new XYChart.Data(0.9, -9));
        series1.getData().add(new XYChart.Data(3.2, 150.8));
        series1.getData().add(new XYChart.Data(4.8, 20.8));
        series1.getData().add(new XYChart.Data(7.3, -42.3));
        series1.getData().add(new XYChart.Data(1.8, 81.4));
        series1.getData().add(new XYChart.Data(7.3, 110.3));
        series1.getData().add(new XYChart.Data(2.7, 41.2));

        sc.setPrefSize(500, 400);
        sc.getData().addAll(series1);
        Scene scene  = new Scene(new Group());
```

```
        final VBox vbox = new VBox();
        final HBox hbox = new HBox();

        final Button add = new Button("Add Series");
        add.setOnAction((ActionEvent e) -> {
            if (sc.getData() == null)
                sc.setData(FXCollections.<XYChart.Series<Number,
                        Numbe>>observableArrayList());
            ScatterChart.Series<Number, Number> series =
                    new ScatterChart.Series<>();
            series.setName("Option "+(sc.getData().size()+1));
            for (int i=0; i<100; i++) series.getData().add(
                new ScatterChart.Data<>(Math.random()*100, Math.random()*500));
            sc.getData().add(series);
        });

        final Button remove = new Button("Remove Series");
        remove.setOnAction((ActionEvent e) -> {
            if (!sc.getData().isEmpty())
                sc.getData().remove((int)(
                    Math.random()*(sc.getData().size()-1)));
        });

        final DropShadow shadow = new DropShadow();
        shadow.setOffsetX(2);
        shadow.setColor(Color.GREY);
        sc.setEffect(shadow);

        hbox.setSpacing(10);
        hbox.getChildren().addAll(add, remove);

        vbox.getChildren().addAll(sc, hbox);
        hbox.setPadding(new Insets(10, 10, 10, 50));

        ((Group)scene.getRoot()).getChildren().add(vbox);
        //scene.getStylesheets().add("scatterchartsample/Chart.css");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

BarChartSample.java

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
```

```
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */


package barchartsample;

import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.scene.Scene;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.stage.Stage;
import javafx.util.Duration;

public class BarChartSample extends Application {

    final static String austria = "Austria";
    final static String brazil = "Brazil";
    final static String france = "France";
    final static String italy = "Italy";
    final static String usa = "USA";

    @Override
    public void start(Stage stage) {
        stage.setTitle("Bar Chart Sample");
        final NumberAxis xAxis = new NumberAxis();
        final CategoryAxis yAxis = new CategoryAxis();
        final BarChart<Number, String> bc =
                new BarChart<>(xAxis, yAxis);
        bc.setTitle("Country Summary");
        xAxis.setLabel("Value");
        xAxis.setTickLabelRotation(90);
        yAxis.setLabel("Country");

        XYChart.Series series1 = new XYChart.Series();
        series1.setName("2003");
        series1.getData().add(new XYChart.Data(25601.34, austria));
        series1.getData().add(new XYChart.Data(20148.82, brazil));
        series1.getData().add(new XYChart.Data(10000, france));
```

```
        series1.getData().add(new XYChart.Data(35407.15, italy));
        series1.getData().add(new XYChart.Data(12000, usa));

        XYChart.Series series2 = new XYChart.Series();
        series2.setName("2004");
        series2.getData().add(new XYChart.Data(57401.85, austria));
        series2.getData().add(new XYChart.Data(41941.19, brazil));
        series2.getData().add(new XYChart.Data(45263.37, france));
        series2.getData().add(new XYChart.Data(117320.16, italy));
        series2.getData().add(new XYChart.Data(14845.27, usa));

        XYChart.Series series3 = new XYChart.Series();
        series3.setName("2005");
        series3.getData().add(new XYChart.Data(45000.65, austria));
        series3.getData().add(new XYChart.Data(44835.76, brazil));
        series3.getData().add(new XYChart.Data(18722.18, france));
        series3.getData().add(new XYChart.Data(17557.31, italy));
        series3.getData().add(new XYChart.Data(92633.68, usa));

        Timeline tl = new Timeline();
        tl.getKeyFrames().add(new KeyFrame(Duration.millis(500),
                (ActionEvent actionEvent) -> {
                    bc.getData().stream().forEach((series) -> {
                        series.getData().stream().forEach((data) -> {
                            data.setXValue(Math.random() * 1000);
                        });
                });
        }));
        tl.setCycleCount(Animation.INDEFINITE);
        tl.setAutoReverse(true);
        tl.play();

        Scene scene = new Scene(bc, 800, 600);
        bc.getData().addAll(series1, series2, series3);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

C

CSS Samples

This appendix lists code for the following JavaFX samples:

- DownloadButton.java
- DownloadButtonStyle1.css
- DownloadButtonStyle2.css
- StyleStage.java
- UIControlCSS.java
- controlStyle1.css
- controlStyle2.css

For the descriptions of this source files, see Skinning JavaFX Applications with CSS.

DownloadButton.java

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
```

```
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
package uicontrolcss;


import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
import uicontrolcss.StyleStage.SceneCreator;



/**
 *
 * @author Alexander Kouznetsov
 */
public class DownloadButton extends Application implements SceneCreator {

    public static void main(String[] args) {
        Application.launch(args);
    }

    @Override
    public void start(Stage stage) {

        StyleStage styleStage = new StyleStage(stage);
        styleStage.add("Style1", "DownloadButtonStyle1.css");
        styleStage.add("Style2", "DownloadButtonStyle2.css");
        stage.show();
        styleStage.setSceneCreator(this);
    }

    @Override
    public Scene createScene() {
        Button download = new Button("Download");
        download.getStyleClass().add("button1");

        Button go = new Button("Go");

        Button submit = new Button("Submit");
        submit.getStyleClass().add("button2");

        HBox hBox = new HBox(10);
        hBox.getChildren().addAll(download, go, submit);

        return new Scene(hBox, 400, 100);
    }

}
```

DownloadButtonStyle1.css

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
```

```
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/*
    Document   : controlStyle
*/

.button {
    -fx-text-fill: #e4f3fc;
    -fx-font: bold 20pt "Tahoma";
    -fx-padding: 10;
    -fx-color: #2d4b8e
}

.button:hover{
    -fx-color: #395bae;
}
```

DownloadButtonStyle2.css

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
```

```
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/*
    Document   : controlStyleLong
*/

.button {
    -fx-text-fill: #e4f3fc;
    -fx-font: bold 20pt "Tahoma";
    -fx-padding: 10;
    -fx-color: #2d4b8e
}

.button:hover{
    -fx-color: #395bae;
}

.button1 {
    -fx-font: 20pt "Verdana";
    -fx-text-fill: #5086bb;
    -fx-padding: 10;
    -fx-background-color: linear-gradient(#cbd0d7, white);
    -fx-background-radius: 23;
    -fx-border-radius: 23;
    -fx-border-color: #767676;

}

.button1:hover {
    -fx-background-color: linear-gradient(white, #cbd0d7);
}

.button2 {
    -fx-background-color: #a0b616;
    -fx-text-fill: white;
    -fx-font: bold 22pt "Courier New";
    -fx-padding: 10;
    -fx-background-radius: 10;
    -fx-border-radius: 10;
    -fx-border-style: dotted;
    -fx-border-color: #67644e;
    -fx-border-width: 2;
}

.button2:pressed {
    -fx-scale-x: 0.8;
```

```
    -fx-scale-y: 0.8;
}
```

StyleStage.java

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
package uicontrolcss;


import javafx.beans.Observable;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;
import javafx.beans.value.ObservableValue;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.ToggleButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
import javafx.stage.WindowEvent;


/**
 *
 * @author Alexander Kouznetsov
 */
public class StyleStage {

    private final StylePanel stylePanel;
```

```
    private final Stage stage;
    private Stage demoStage;

    public StyleStage(Stage stage) {
        this.stage = stage;
        stylePanel = new StylePanel();

        Scene scene = new Scene(stylePanel);

        stage.setScene(scene);
        stage.setTitle("Choose style");

        stage.setOnCloseRequest((WindowEvent t) -> {
            demoStage.close();
        });
    }

    public void add(String name, String styleSheetName) {
        stylePanel.add(name, styleSheetName);
    }

    public void setSceneCreator(final SceneCreator sceneCreator) {
        if (demoStage == null) {
            demoStage = new Stage();
            demoStage.setTitle("Demo");
            demoStage.setX(stage.getX());
            demoStage.setY(stage.getY() + stage.getHeight());
        }
        demoStage.setScene(sceneCreator.createScene());
        demoStage.show();
        stylePanel.selected.addListener((ObservableValue<? extends String> ov,
            String t, String t1) -> {
            demoStage.setScene(sceneCreator.createScene());

            if (t1 != null) {
                demoStage.getScene().getStylesheets().setAll(
                        UIControlCSS.class.getResource(t1).toString());
            }
        });
    }

    public static interface SceneCreator {
        Scene createScene();
    }
}

class StylePanel extends HBox {

    public StringProperty selected = new SimpleStringProperty();

    ToggleGroup stylesheetToggleGroup = new ToggleGroup();

    public StylePanel() {
        super(5);

        StyleButton defaultStylesheetButton = new StyleButton("Default", null);
        defaultStylesheetButton.setSelected(true);
        defaultStylesheetButton.setToggleGroup(stylesheetToggleGroup);

        setPadding(new Insets(0, 0, 30, 0));
```

```
        setAlignment(Pos.BOTTOM_LEFT);
        getChildren().addAll(defaultStylesheetButton);
    }

    public void add(String name, String styleSheetName) {
        StyleButton styleButton = new StyleButton(name, styleSheetName);
        styleButton.setToggleGroup(stylesheetToggleGroup);
        getChildren().addAll(styleButton);
    }

    class StyleButton extends ToggleButton {

        public StyleButton(String text, final String styleSheetName) {
            super(text);
            selectedProperty().addListener((Observable ov) -> {
                selected.set(styleSheetName);
            });
        }
    }
}
```

UIControlCSS.java

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
package uicontrolcss;


import javafx.application.Application;
import javafx.beans.binding.Bindings;
import javafx.collections.FXCollections;
```

```
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Orientation;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.ProgressIndicator;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.Separator;
import javafx.scene.control.Slider;
import javafx.scene.control.TextField;
import javafx.scene.control.ToggleGroup;
import javafx.scene.control.Tooltip;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import uicontrolcss.StyleStage.SceneCreator;



/**
 *
 * @author Alexander Kouznetsov
 */
public class UIControlCSS extends Application implements SceneCreator {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Application.launch(args);
    }

    @Override
    public void start(Stage stage) {
        StyleStage styleStage = new StyleStage(stage);
        styleStage.add("controlStyle1", "controlStyle1.css");
        styleStage.add("controlStyle2", "controlStyle2.css");
        stage.show();
        styleStage.setSceneCreator(this);
    }

    @Override
    public Scene createScene() {

        ToggleGroup toggleGroup = new ToggleGroup();

        RadioButton radioButton1 = new RadioButton("High");
        radioButton1.setToggleGroup(toggleGroup);
        radioButton1.setSelected(true);

        RadioButton radioButton2 = new RadioButton("Medium");
        radioButton2.setToggleGroup(toggleGroup);

        RadioButton radioButton3 = new RadioButton("Low");
```

```
radioButton3.setToggleGroup(toggleGroup);

VBox vBox1 = new VBox(2);
vBox1.getChildren().addAll(radioButton1, radioButton2, radioButton3);

TextField textField = new TextField();
textField.setPrefColumnCount(10);
textField.setPromptText("Your name");

PasswordField passwordField = new PasswordField();
passwordField.setPrefColumnCount(10);
passwordField.setPromptText("Your password");

VBox vBox2 = new VBox();
vBox2.getChildren().addAll(textField, passwordField);

ChoiceBox<String> choiceBox = new ChoiceBox<>(
    FXCollections.observableArrayList("English", "???????",
        "Fran\u00E7ais"));
choiceBox.setTooltip(new Tooltip("Your language"));
choiceBox.getSelectionModel().select(0);

HBox hBox1 = new HBox(5);
hBox1.setAlignment(Pos.BOTTOM_LEFT);
hBox1.getChildren().addAll(vBox1, vBox2, choiceBox);

final Label label1 = new Label("Not Available");
label1.getStyleClass().add("borders");

Button button1 = new Button("Accept");
button1.getStyleClass().add("button1");
button1.setOnAction((ActionEvent t) -> {
    label1.setText("Accepted");
});

Button button2 = new Button("Decline");
button2.getStyleClass().add("button2");
button2.setOnAction((ActionEvent t) -> {
    label1.setText("Declined");
});

HBox hBox2 = new HBox(10);
hBox2.setAlignment(Pos.CENTER_LEFT);
hBox2.getChildren().addAll(button1, button2, label1);

CheckBox checkBox1 = new CheckBox("Normal");

Separator separator = new Separator(Orientation.VERTICAL);
separator.setPrefSize(1, 15);

CheckBox checkBox2 = new CheckBox("Checked");
checkBox2.setSelected(true);

CheckBox checkBox3 = new CheckBox("Undefined");
checkBox3.setIndeterminate(true);
checkBox3.setAllowIndeterminate(true);

HBox hBox3 = new HBox(12);
hBox3.getChildren().addAll(checkBox1, separator, checkBox2, checkBox3);
```

```
        Label label2 = new Label("Progress:");
        label2.getStyleClass().add("borders");

        Slider slider = new Slider();

        ProgressIndicator progressIndicator = new ProgressIndicator(0);
        progressIndicator.progressProperty().bind(Bindings.divide(
            slider.valueProperty(), slider.maxProperty()));

        HBox hBox4 = new HBox(10);
        hBox4.getChildren().addAll(label2, slider, progressIndicator);

        final VBox vBox = new VBox(20);
        vBox.setPadding(new Insets(30, 10, 30, 10));
        vBox.setAlignment(Pos.TOP_LEFT);
        vBox.getChildren().setAll(hBox1, hBox2, hBox3, hBox4);

        ScrollPane scrollPane = new ScrollPane();
        scrollPane.setContent(vBox);

        return new Scene(scrollPane, 500, 350);
    }
}
```

controlStyle1.css

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/*
    Document   : controlStyle1
```

```
*/

.root{
    -fx-font-size: 14pt;
    -fx-font-family: "Tahoma";
    -fx-base: #DFB951;
    -fx-background: #A78732;
    -fx-focus-color: #B6A678;
}

.button1{
    -fx-text-fill: #006464;
    -fx-background-color: #DFB951;
    -fx-border-radius: 20;
    -fx-background-radius: 20;
    -fx-padding: 5;
}

.button2{
    -fx-text-fill: #c10000;
    -fx-background-color: #DFB951;
    -fx-border-radius: 20;
    -fx-background-radius: 20;
    -fx-padding: 5;
}

.slider{
    -fx-border-color: white;
    -fx-border-style: dashed;
    -fx-border-width: 2;
}
```

controlStyle2.css

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
```

```
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/*
    Document   : controlStyle2
*/

.root{
    -fx-font-size: 16pt;
    -fx-font-family: "Courier New";
    -fx-base: rgb(132, 145, 47);
    -fx-background: rgb(225, 228, 203);
}

.button{
    -fx-text-fill: rgb(49, 89, 23);
    -fx-border-color: rgb(49, 89, 23);
    -fx-border-radius: 5;
    -fx-padding: 3 6 6 6;
}

.borders{
    -fx-border-color: rgb(103, 100, 78);
    -fx-border-style: dotted;
    -fx-border-width: 1.5;
    -fx-border-insets: -5;
}
```

D

Text Samples

This appendix lists code for the following JavaFX samples:

- TextEffects.java
- NeonSign.java

For the descriptions of the source files, see Applying Effects to Text.

TextEffects.java

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */



package texteffects;
```

```
import javafx.application.Application;
import javafx.collections.ObservableList;
import javafx.geometry.VPos;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.effect.*;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;


public class TextEffects extends Application {

    Stage stage;
    Scene scene;

    @Override
    public void start(Stage stage) {
        stage.show();

        scene = new Scene(new Group(), 900, 600);
        ObservableList content = ((Group)scene.getRoot()).getChildren();

        /// Perspective
        content.add(perspective());
        /// DropShadow
        content.add(dropShadow());
        /// Bloom
        content.add(bloom());
        /// BoxBlur
        content.add(boxBlur());
        /// DisplacementMap
        content.add(displacementMap());
        /// InnerShadow
        content.add(innerShadow());
        /// Lighting
        content.add(lighting());
        /// MotionBlur
        content.add(motionBlur());
        /// Reflection
        content.add(reflection());
        /// GaussianBlur
        content.add(gaussianBlur());
        /// DistantLight
        content.add(distantLight());
        stage.setScene(scene);
        stage.setTitle("Text Effects");
    }

    static Node distantLight() {
        Light.Distant light = new Light.Distant();
        light.setAzimuth(-135.0f);
        light.setElevation(30.0f);

        Lighting l = new Lighting();
```

```
        l.setLight(light);
        l.setSurfaceScale(5.0f);

        final Text t = new Text();
        t.setText("DistantLight");
        t.setFill(Color.RED);
        t.setFont(Font.font(null, FontWeight.BOLD, 70));
        t.setX(10.0f);
        t.setY(10.0f);
        t.setTextOrigin(VPos.TOP);

        t.setEffect(l);

        final Rectangle r = new Rectangle();
        r.setFill(Color.BLACK);

        Group g = new Group();
        g.getChildren().add(r);
        g.getChildren().add(t);

        g.setTranslateY(470);

        return g;
    }

    static Node perspective() {
        Group g = new Group();
        PerspectiveTransform pt = new PerspectiveTransform();
        pt.setUlx(10.0f);
        pt.setUly(10.0f);
        pt.setUrx(310.0f);
        pt.setUry(40.0f);
        pt.setLrx(310.0f);
        pt.setLry(60.0f);
        pt.setLlx(10.0f);
        pt.setLly(90.0f);

        g.setEffect(pt);
        g.setCache(true);

        Rectangle r = new Rectangle();
        r.setX(10.0f);
        r.setY(10.0f);
        r.setWidth(280.0f);
        r.setHeight(80.0f);
        r.setFill(Color.BLUE);

        Text t = new Text();
        t.setX(20.0f);
        t.setY(65.0f);
        t.setText("Perspective");
        t.setFill(Color.YELLOW);
        t.setFont(Font.font(null, FontWeight.BOLD, 36));

        g.getChildren().add(r);
        g.getChildren().add(t);
        return g;
    }

    static Node gaussianBlur() {
```

```
        Text t2 = new Text();
        t2.setX(10.0f);
        t2.setY(140.0f);
        t2.setCache(true);
        t2.setText("Blurry Text");
        t2.setFill(Color.RED);
        t2.setFont(Font.font(null, FontWeight.BOLD, 36));
        t2.setEffect(new GaussianBlur());
        return t2;
    }

    static Node reflection() {
        Text t = new Text();
        t.setX(10.0f);
        t.setY(50.0f);
        t.setCache(true);
        t.setText("Reflections on JavaFX...");
        t.setFill(Color.RED);
        t.setFont(Font.font(null, FontWeight.BOLD, 30));

        Reflection r = new Reflection();
        r.setFraction(0.7f);

        t.setEffect(r);

        t.setTranslateY(400);
        return t;
    }

    static Node motionBlur() {
        Text t = new Text();
        t.setX(100.0f);
        t.setY(100.0f);
        t.setText("Motion");
        t.setFill(Color.RED);
        t.setFont(Font.font(null, FontWeight.BOLD, 60));

        MotionBlur mb = new MotionBlur();
        mb.setRadius(15.0f);
        mb.setAngle(-30.0f);

        t.setEffect(mb);

        t.setTranslateX(300);
        t.setTranslateY(150);

        return t;
    }

    static Node lighting() {
        Light.Distant light = new Light.Distant();
        light.setAzimuth(-135.0f);

        Lighting l = new Lighting();
        l.setLight(light);
        l.setSurfaceScale(5.0f);

        Text t = new Text();
        t.setText("Lighting!");
        t.setFill(Color.RED);
```

```
        t.setFont(Font.font(null, FontWeight.BOLD, 70));
        t.setX(10.0f);
        t.setY(10.0f);
        t.setTextOrigin(VPos.TOP);

        t.setEffect(l);

        t.setTranslateX(0);
        t.setTranslateY(320);

        return t;
    }

    static Node innerShadow() {
        InnerShadow is = new InnerShadow();
        is.setOffsetX(4.0f);
        is.setOffsetY(4.0f);

        Text t = new Text();
        t.setEffect(is);
        t.setX(20);
        t.setY(100);
        t.setText("InnerShadow");
        t.setFill(Color.YELLOW);
        t.setFont(Font.font(null, FontWeight.BOLD, 80));

        t.setTranslateX(350);
        t.setTranslateY(300);

        return t;
    }

    static Node displacementMap() {
        int w = 220;
        int h = 100;
        FloatMap map = new FloatMap();
        map.setWidth(w);
        map.setHeight(h);

        for (int i = 0; i < w; i++) {
            double v = (Math.sin(i / 50.0 * Math.PI) - 0.5) / 40.0;
            for (int j = 0; j < h; j++) {
                map.setSamples(i, j, 0.0f, (float) v);
            }
        }

        Group g = new Group();
        DisplacementMap dm = new DisplacementMap();
        dm.setMapData(map);

        Rectangle r = new Rectangle();
        r.setX(80.0f);
        r.setY(40.0f);
        r.setWidth(w);
        r.setHeight(h);
        r.setFill(Color.BLUE);

        g.getChildren().add(r);

        Text t = new Text();
```

```
        t.setX(100.0f);
        t.setY(80.0f);
        t.setText("Wavy Text");
        t.setFill(Color.YELLOW);
        t.setFont(Font.font(null, FontWeight.BOLD, 36));

        g.getChildren().add(t);

        g.setEffect(dm);
        g.setCache(true);

        g.setTranslateX(300);
        g.setTranslateY(180);

        return g;
    }

    static Node boxBlur() {
        Text t = new Text();
        t.setText("Blurry Text!");
        t.setFill(Color.RED);
        t.setFont(Font.font(null, FontWeight.BOLD, 36));
        t.setX(10);
        t.setY(40);

        BoxBlur bb = new BoxBlur();
        bb.setWidth(15);
        bb.setHeight(15);
        bb.setIterations(3);

        t.setEffect(bb);
        t.setTranslateX(350);
        t.setTranslateY(100);

        return t;
    }


    static Node dropShadow() {
        DropShadow ds = new DropShadow();
        ds.setOffsetY(3.0f);
        ds.setColor(Color.color(0.4f, 0.4f, 0.4f));

        Text t = new Text();
        t.setEffect(ds);
        t.setCache(true);
        t.setX(10.0f);
        t.setY(270.0f);
        t.setFill(Color.RED);
        t.setText("JavaFX drop shadow...");
        t.setFont(Font.font(null, FontWeight.BOLD, 32));

        return t;
    }

    static Node bloom() {
        Group g = new Group();

        Rectangle r = new Rectangle();
        r.setX(10);
```

```
        r.setY(10);
        r.setWidth(160);
        r.setHeight(80);
        r.setFill(Color.DARKBLUE);

        Text t = new Text();
        t.setText("Bloom!");
        t.setFill(Color.YELLOW);
        t.setFont(Font.font(null, FontWeight.BOLD, 36));
        t.setX(25);
        t.setY(65);

        g.setCache(true);
        g.setEffect(new Bloom());
        g.getChildren().add(r);
        g.getChildren().add(t);
        g.setTranslateX(350);
        return g;
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Application.launch (args);
    }
}
```

NeonSign.java

```
/*
 * Copyright (c) 2011, 2014, Oracle and/or its affiliates.
 * All rights reserved. Use is subject to license terms.
 *
 * This file is available and licensed under the following license:
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the distribution.
 *  - Neither the name of Oracle nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
```

```
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package neonsign;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.effect.Blend;
import javafx.scene.effect.BlendMode;
import javafx.scene.effect.DropShadow;
import javafx.scene.effect.InnerShadow;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;
import javafx.stage.Stage;


public class NeonSign extends Application {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Application.launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Text Effects: Neon Sign");
        double width = 600;
        double height = 300;
        Pane root = new Pane();
        Scene scene = new Scene(root, width, height);

        Rectangle rect = new Rectangle (width, height);
        rect.getStyleClass().add("myrect");

        Text text = new Text();
        text.setId("fancytext");

        text.setX(20);
        text.setY(150);
        Blend blend = new Blend();
        blend.setMode(BlendMode.MULTIPLY);

        DropShadow ds = new DropShadow();
        ds.setColor(Color.rgb(254, 235, 66, 0.3));
        ds.setOffsetX(5);
        ds.setOffsetY(5);
        ds.setRadius(5);
        ds.setSpread(0.2);

        blend.setBottomInput(ds);

        DropShadow ds1 = new DropShadow();
        ds1.setColor(Color.web("#f13a00"));
        ds1.setRadius(20);
```

```
        ds1.setSpread(0.2);

        Blend blend2 = new Blend();
        blend2.setMode(BlendMode.MULTIPLY);

        InnerShadow is = new InnerShadow();
        is.setColor(Color.web("#feeb42"));
        is.setRadius(9);
        is.setChoke(0.8);
        blend2.setBottomInput(is);

        InnerShadow is1 = new InnerShadow();
        is1.setColor(Color.web("#f13a00"));
        is1.setRadius(5);
        is1.setChoke(0.4);
        blend2.setTopInput(is1);

        Blend blend1 = new Blend();
        blend1.setMode(BlendMode.MULTIPLY);
        blend1.setBottomInput(ds1);
        blend1.setTopInput(blend2);

        blend.setTopInput(blend1);

        text.setEffect(blend);

        TextField textField = new TextField();
        textField.setText("Neon Sign");
        text.textProperty().bind(textField.textProperty());
        textField.setPrefColumnCount(40);
        textField.setLayoutX(50);
        textField.setLayoutY(260);

        root.getChildren().addAll(rect,text,textField);
        primaryStage.setScene(scene);

scene.getStylesheets().add(NeonSign.class.getResource("brickStyle.css").toExternal
Form());
        primaryStage.show();
    }
}
```