Oracle® Database

Get Started with Java Development

23c
F47017-02
September 2023

Oracle Database Get Started with Java Development, 23c

F47017-02

Primary Author: Tulika Das

Contributing Authors: Apoorva Srinivas, Tanmay Choudhury

Contributors: Kuassi Mensah, Nirmala Sundarappa

Contents

List of Tables

Preface

This preface discusses the intended audience and conventions used in this book. It also includes a list of related Oracle documents that you can refer to for more information.

Audience

This guide is intended for application developers using Java to access and modify data in Oracle Database. This guide illustrates how to perform these tasks using a simple Java Database Connectivity (JDBC) driver to connect to the database. The application also uses Universal Connection Pool (UCP) which is a Java Connection Pool intended to optimize database resources and improve performance.

This guide can be read by anyone with an interest in Java programming, but it assumes at least some prior knowledge of the following:

- Java
- Oracle databases

Related Documents

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

- *Oracle Database JDBC Developer's Guide*
- *Oracle Database Java Developer's Guide*
- *Oracle Universal Connection Pool Developer's Guide*

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.
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.

1

Aims and Objectives of This Book

Java is a popular language among developers that is used to build various enterprise solutions.

This guide will help you understand all Java products used to build a Java application. You will learn how to model a Java Web application using MVC design pattern, Oracle JDBC Thin driver, Universal Connection Pool (UCP), and Java in the Database (using embedded OJVM).

In the next few chapters, you will create a Java web application — 'HR Web application'. This application helps the HR team of AnyCo Corporation to lookup or modify details of a specific employee, or all employees, delete an employee, or apply a salary raise to all employees.

The HR application has two users:

- **hrstaff**
- **hradmin**

Each user has a different set of roles and privileges.

This Chapter contains the following topics:

- Architecture of the Web Application
- Components of the Application
- Objectives and Tasks

1.1 Architecture of the HR Web Application

The HR Web application uses the MVC (Model, View, Controller) architecture and the latest tools and technologies. A Model View Controllder (MVC) is a design pattern that is easy-to-use. It separates the web application into three simple parts (Model-View-Controller).

The **Model** stores the data or the information that the web application is meant to operate on. It does not include any information about the user-interface.

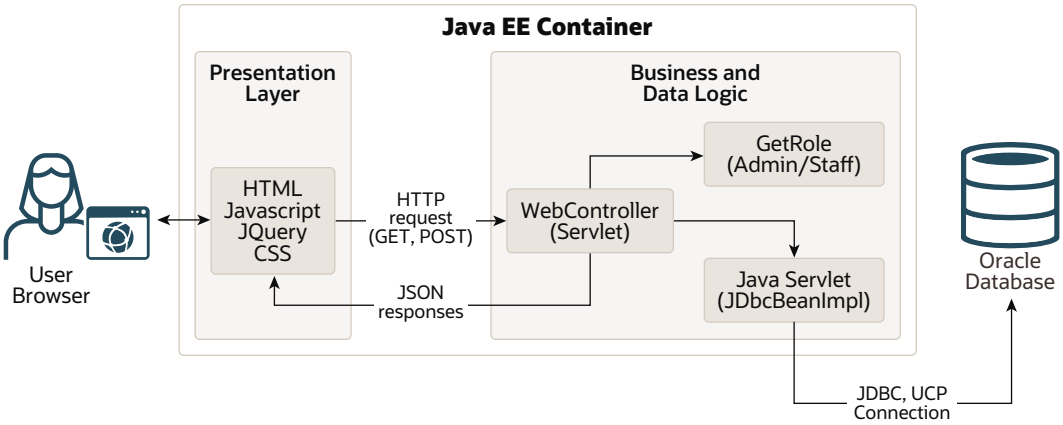
The **View** contains all elements of the user interface (UI). This includes buttons, display box, links, input box etc.

The **Controller** connects **Model** and **View**.

As a user, you will see the interface (View) that could be a JSP page to an HTML page after you log into the application. The Controller (a Java Servlet) renders the correct View to the user during logging in, or any other flow. When you request for data or an update to the data, the Controller invokes the Model that represents the data in terms of tables or views, and renders the data. The Model represents the user data usually stored in an Oracle Database or any other database.

The Controller then passes on this data to the **View** to show it to the user in a presentable format.

Figure 1-1 Pictorial Depiction of the Web Application



The following table describes the various components of the application:

Table 1-1 Architecture of the Web Application

Name	Technologies Used	Description
Model	Oracle Database, Java Beans	Represents the information or the data on which the application operates.
View	HTML, JavaScript, JQuery, CSS	User interface that renders the model to the end user. It includes all elements visible to the user such as buttons, links, input, etc.
Controller	Java Servlet	The controller processes and responds to user actions. It orchestrates the flow based on user input. It also connects Model and View and renders an output to the user.

You will use HR schema and the Employees to understand the flows in the Web application.

1.2 Required Components for the HR Web Application

The following table lists and describes all the components required for the application.

Table 1-2 Components Required for the Application

Package Name	Description
src	Contains source files
target	Contains class files
src/main/java/com/oracle/jdbc/samples	-
/bean/JdbcBean.java	Defines the employee details as attributes
/bean/JdbcBeanImpl.java	Implementation class of the EmployeeBean

Table 1-2 (Cont.) Components Required for the Application

Package Name	Description
src/main/java/com/oracle/jdbc/samples	-
entity/Employee.java	Consists of all employee attributes and their defined data types
/web/WebController.java	Servlet that controls the application flow
/web/GetRole.java	Creates HRStaff and HRAdmin roles for the application
src/main/resources	-
SalaryHikeSP.java	Java class to be invoked from Java in the database to process an increment in salary
SalaryHikeSP.sql	SQL file with a procedure to increase the salary of the employees based on their salary range
src/main/webapp	-
about.html	Contains the details about the HR Web application
login.html	Contains the login page for the HR Web application
login-failed.html	Page to show when the login is unsuccessful
index.html	Landing page of the HR Web application
listAll.html	HTML page to display all employee records
listByName.html	HTML page to display the result when employees are searched by name
listById.html	HTML page to display the result when employees are searched by employee id
incrementSalary.html	HTML page to display the result after an increment is made to the salary
src/main/webapp	-
css/app.cs	Contains all style and font details used in the HR Web application
src/main/webapp	-
WEB-INF/web.xml	Controller for the HR Web application

1.3 Objectives and Tasks

By the end of this book, you will be able to:

a. Understand the JDBC, UCP, Java in the database and run a simple Java program to get familiar with these products.

c. You will learn how to use Universal Connection Pool (UCP) and Java in the Database (OJVM).

b. Implement the functionality to list all employees, search and retrieve an employee and update an employee record.

An overview of each chapter is described as follows:

1. **Introduction to JDBC, UCP and Java in the Database:** This chapter familiarizes you with the products, associated binaries and packages through a sample code.

2. **Overview of the HR Web Application:** This chapter discusses the HR Web application in depth and familiarize you with the flows of the Web application, packages and files that you create as a part of the web application.
3. **Getting Started with the Application:** In this chapter, you understand the prerequisites for building the application and how to get the environment ready. It starts with signing up for the Oracle Cloud Free Tier or installing the Oracle Database on premise. Later, you install IntelliJ, an IDE to build the application. You use Tomcat Java EE container to deploy and run the application.
   The chapter also helps you download any other tools, such as Maven, that helps you to build the application.
4. **List All Employees:** This chapter helps you to put all the components together and build an initial functionality to connect to the Oracle Database, and retrieve employee details from the database.
5. **Search By Employee ID:** This chapter provides details on how to implement the 'Search by Employee ID' functionality.
6. **Update an Employee Record:** In this chapter, you learn how to update employee records. This is a two step process. Firstly, you search the employee's records, based on first name. Once you retrieve the required results, you can update the salary, job ID, firstname, lastname and other details.
7. **Delete an Employee Record:** In this chapter, you learn how to delete an employee record, in a two-step process.
8. **Increase Salary to All Employees:** In this chapter, you understand how to provide an increment to the salary of the employees listed in the table, using 'Java in the database'.
9. **Creating Application Users:** This chapter shows how to create 'hradmin' and 'hrstaff' users in Tomcat and IntelliJ.
10. **Summary:** This chapter summarizes all that you have learnt so far. It also provides appropriate references and links for enhancing your use of the web application.

2

Brief Introduction to JDBC, UCP, and Java in the Database

The Oracle Database is a relational database that you can use to store, modify and use data.

The Java Database Connectivity (JDBC) standard is used by Java applications to access and manipulate data in relational databases.

JDBC is an industry-standard application programming interface (API) that lets you access a RDBMS using SQL from Java. Each vendor implements the JDBC Specification with its own extensions.

Maven is a build automation tool which is used to build and manage the Java project.

Universal Connection Pool (UCP) is a cache of database connection objects that promote reuse of the connections, thus improving the performance.

Java in the Database (OJVM) helps group SQL operations with Java data logic and load them into the database for in-place processing.

This chapter introduces you to the JDBC driver, Universal Connection Pool (UCP), Java in the Database (OJVM) and Maven with the Oracle Database.

- Java Database Connectivity Driver (JDBC)
- Universal Connection Pool (UCP)
- Java in the Database (Oracle JVM)
- Maven

2.1 Java Database Connectivity Driver (JDBC)

JDBC is a database access protocol that enables you to connect to a database and run SQL statements and queries on the database.

JDBC drivers implement and comply with the latest JDBC specifications. Java applications need to have `ojdbc8.jar` compatible with JDK8 in their classpath.

This guide uses the following JDBC standard:

- Oracle JDBC Thin Driver
- Oracle JDBC Packages

Oracle JDBC Thin Driver

Oracle recommends using the JDBC Thin Driver for most requirements. The JDBC Thin Driver will work on any system with a suitable Java Virtual Machine. (JVM). Some other client drivers that Oracle provides are JDBC thin driver, Oracle Call Interface (OCI) driver, server side thin driver, and server side internal driver.

The JDBC Thin Driver is a pure Java, Type IV driver. The JDBC driver version (`ojdbc8.jar`) includes support for JDK 8.

JDBC Thin Driver communicates with the server using SQL*Net to access the database.

> **See Also:**
>
> *Oracle Database JDBC Developer's Guide*

Oracle JDBC Packages

The following core Java class libraries provide the JDBC APIs:

- `java.sql`
- `javax.sql`

Include the `import` statements at the beginning of your program to import the classes which your application needs.

Using Maven Central

All supported releases of the Oracle JDBC drivers are available on Maven Central. So, you can consider Maven Central as a distribution center for the Oracle JDBC drivers and companion JAR files.

> **See Also:**
>
> Maven Central Guide

2.2 Universal Connection Pool

Connection pools help improve performance by reusing connection objects and reducing the number of times that connection objects are created.

Oracle Universal Connection Pool (UCP) is a feature rich Java connection pool that provides connection pool functionalities, along with high availability, scalability and load balancing with the help of tighter integration with Oracle Database configurations.

A Java application or container must have `ucp.jar` in their classpath, along with the `ojdbc8.jar` (JDK8), to be able to use UCP.

> **See Also:**
>
> *Oracle Universal Connection Pool Developer's Guide*

2.3 Java in the Database (OJVM)

Oracle Database has a Java Virtual Machine (JVM) that resides in the server. It helps Java applications running in the Oracle JVM on the server to access data present on the same system and same process.

Java in the Database is recommended for applications that are data-intensive. JVM has the ability to use the underlying Oracle RDBMS libraries directly, without the use of a network connection between the Java code and SQL data. This helps improve performance and execution. For data access, Oracle Database uses server-side internal driver when Java code runs on the server.

3

Overview of the HR Web Application

The HR Web Application is intended to give you access to information related to all employees of AnyCo Corporation.

The two types of users that will be able to access this application are:

- HRStaff
- HRAdmin

The HRStaff and HRAdmin accounts have different privileges.

HRStaff has read only access to the application and does not have privileges to update/delete an employee record. HRStaff can only List the employees and Search by Employee ID.

The HRAdmin, has complete control on the application and has read and write privileges. HRAdmin is the only user who has access to all functionalities of the application such as *update/delete an employee record*, or *provide salary increment for all employees*.

This chapter has the following sections:

3.1 Functionalities of the HR Web Application

Following is a list of functionalities to access information related to AnyCo Corporation:

Through the **hrstaff**, you can perform the following functions:

- **List All Employees**

  Use the `List All Employees` option to retrieve employee information. This function lists information such as `Employee_ID`, `First_Name`, `Last_Name`, `Email`, `Phone_Number`, `Job_Id`, and `Salary`.

- **Search By Employee ID**

  Use the `Employee ID` which is the primary key to search for a particular employee.

  Through the **hradmin** user, you can perform the following functions:

  The **hradmin** user has full control of the application and has both read and update privileges.

- **Update Employee Record**

  You can update employee records, using the `Update Employee Record` function. First, search for employees, based on the name of the employee. You can then update employee details in the record, such as `first_name`, `last_name`, `email`, `phone_number`, `job_id` and `salary` using the `UPDATE` function.

  Use the `DELETE` function to the delete the entire employee record from the database.

- **Increment Salary**

  Through the increment salary tab, you can alter (increase or decrease) the percentage of salary for hike.

- **About**

  This page provides an overview of the HR Application and explains the various functionalities it offers.

4
Getting Started with the Application

To develop a Java application that connects to the Oracle Database, you must ensure that certain components are installed.

This chapter covers the following topics:

- What You Need to Install
- Verifying the Oracle Database 21c Installation

4.1 What You Need to Install

To develop the sample application, you need to install the following products and components:

- Oracle Database
- J2SE or JDK
- Apache Maven
- IntelliJ
- Apache Tomcat

4.1.1 Oracle Database

To develop the Java web application, you need a working installation of Oracle Database along with the HR schema. There are two ways you can install the Oracle Database.

Option 1. Oracle Autonomous Database

The Oracle Database Cloud Services offer access to Oracle Cloud Free Tier which provides services of two Autonomous Databases for an unlimited time. Oracle Autonomous Database is an all-in-one cloud database solution for data marts, data lakes, operational reporting, and batch data processing. Oracle uses machine learning to completely automate all routine database tasks—ensuring higher performance, reliability, security, and operational efficiency. You can get access to a working Autonomous Database in just a few minutes. After signing up, you will have access to your choice of **Autonomous Transaction Processing** or **Autonomous Data Warehouse** databases.

This guide uses Autonomous Transaction Processing database to run the HR Application.

> **See Also:**
>
> Provisioning an ATP Database instance video for instructions

Option 2: Oracle Database Free Available on OTN

As an alternate option, you can install Oracle Database Free on your system.

> **See Also:**
>
> *Oracle Database Free Installation Guide*

4.1.2 Install the HR schema

The HR web application uses the tables and data from the HR sample schema provided by Oracle. You need to install the HR schema in your database.

Once you provision an Autonomous Database or install Oracle Database Free, see Installing HR Schema for detailed instructions to install the sample schema.

4.1.3 J2SE or JDK

To create and compile Java applications, you need the full Java 2 Platform, Standard Edition, Software Development Kit (J2SE SDK), formerly known as the Java Development Kit (JDK).

1. Download and install the Java SE. Refer http://www.oracle.com/technetwork/java/javase/downloads/index.html.
2. Set the `PATH` environment variable for JDK. Refer JDK Installation Instructions for Windows for detailed instructions.

4.1.4 JDBC Drivers

You need to download certain JDBC drivers for running the HR application.

This guide uses Maven Central to download the necessary JDBC drivers required for the application. Later in this guide, you learn to add the following dependency to your project's `pom.xml` file:

```
<dependencies>
  <dependency>
    <groupId>com.oracle.database.jdbc<groupId>
    <artifactId>ojdbc8-production</artifactId>
    <version>19.7.0.0</version>
    <type>pom</type>
  </dependency>
</dependencies>
```

The `ojdbc8-production` pulls all the required JDBC jars from the Maven Central Repository. For more information, see the Maven Central Guide.

4.1.5 Integrated Development Environment

For ease in developing the application, you can choose to develop your application in an integrated development environment (IDE). This guide uses IntelliJ Idea community version to create and update the files for this application.

> **Note:**
>
> To download and install IntelliJ Idea, see https://www.jetbrains.com/idea/.

4.1.6 Web Server

The sample application developed in this guide uses JavaServer Pages (JSP) technology to display information and accept input from users. To deploy these pages, you need a Web server with a servlet and JSP container.

This guide uses the Apache Tomcat server for deploying the JSP pages.

For more information on how to download and install Apache Tomcat, refer to http://tomcat.apache.org/.

4.2 Verifying the Oracle Database Installation

Oracle Database installation is platform-specific. If you are using Oracle Database Free, then verify that the installation was successful before you proceed to create the sample application.

Verify the Oracle Database Free Installation

Use the following commands to connect to the Oracle Database Free installation and verify whether you have successfully installed it or not:

```
$ cd $ORACLE_HOME/bin
$ ./sqlplus / as sysdba
```

An output similar to the following confirms that you are now connected to the root container `CDB$ROOT` of the multitenant database (CDB), as the `SYS` user:

```
SQL*Plus: Release 23.0.0.0.0 - Developer-Release on Wed Mar 15 22:37:14 2023
Version 23.2.0.0.0
Copyright (c) 1982, 2023, Oracle. All rights reserved.
Connected to:
Oracle Database 23c Free, Release 23.0.0.0.0 - Developer-Release
Version 23.2.0.0.0
```

> **See Also:**
>
> *Oracle Database Free Installation Guide*

Verify connecting to Oracle ATP Instance

Once you provision an ATP instance, verify the connection as follows:

1. Download the Client Credentials. Login to your database instance and click on the **DB Connection** tab. Select **Instance Wallet** for Wallet type and click **Download Wallet**. Unzip the file to a secure directory on your local machine.

2. Follow instructions provided in the Java Connectivity with Autonomous Database to connect using JDBC and UCP code samples.

4.3 Github Repository Details

The source files for the HR Web Application is available on GitHub.

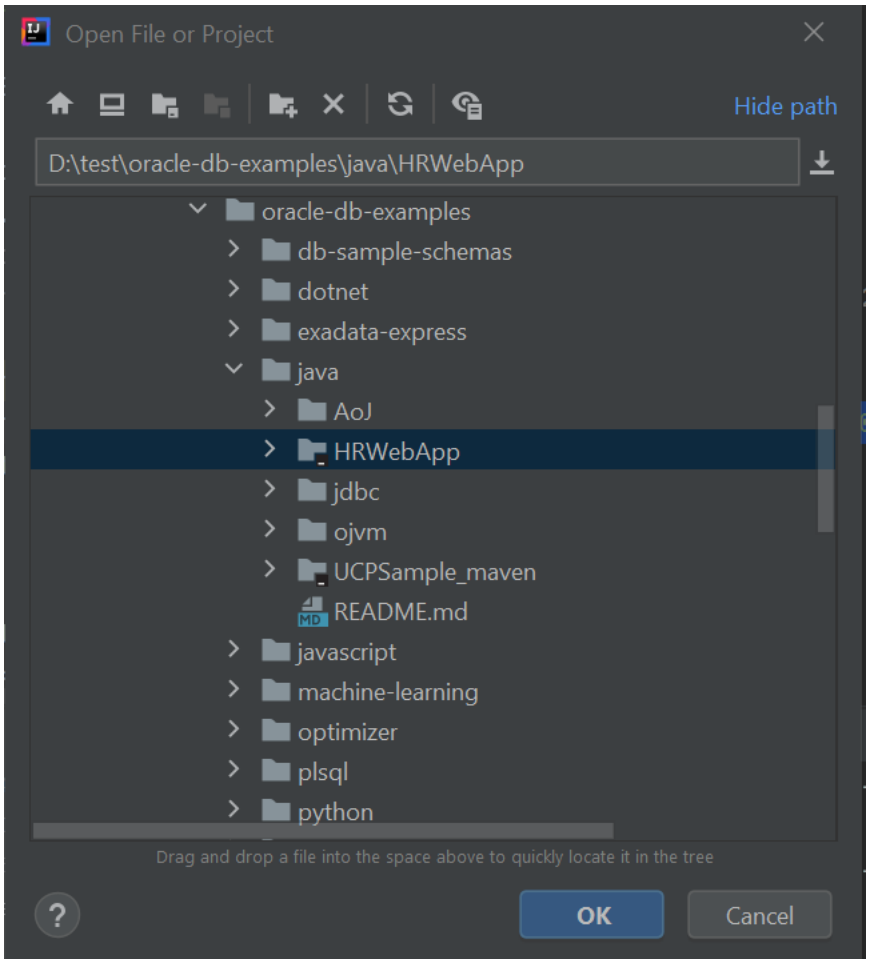
Table 4-1 Github Repository Details

Name and Location	Details
HRWebApp	This repository contains the complete code samples of the application.

4.4 Import the Application in IntelliJ

Import the application in IntelliJ as follows:

1. Open IntelliJ and select **Open or Import**.
2. Navigate to the location where the HRWebApp is downloaded. Select HRWebApp and click **OK.**



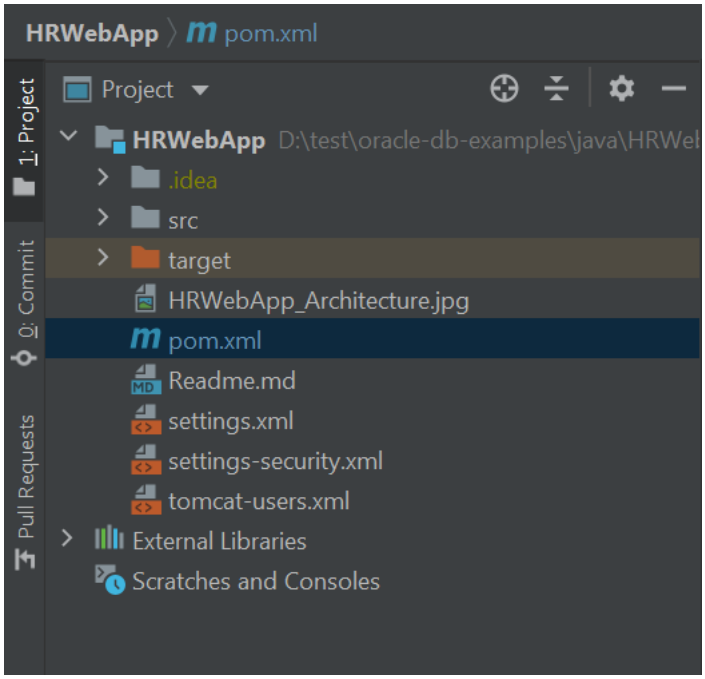
3. A project with all the files required to build the HR Web application is displayed.

4.5 Compile the Application in IntelliJ

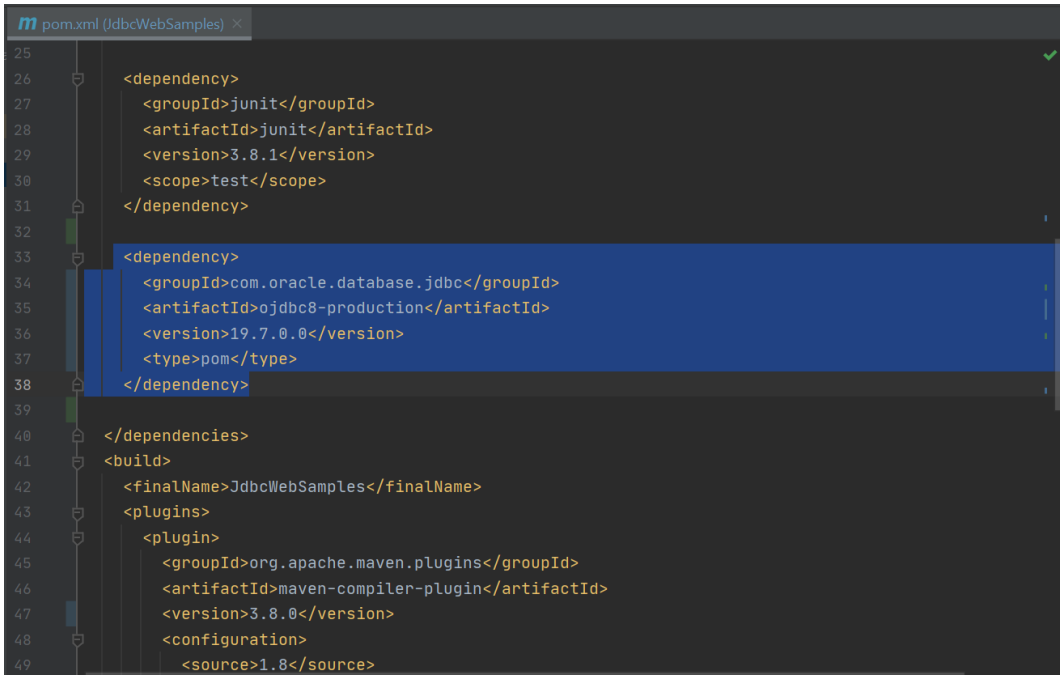
The code requires a few updates before you compile the application.

The HRWebApp must be downloaded and opened in IntelliJ.

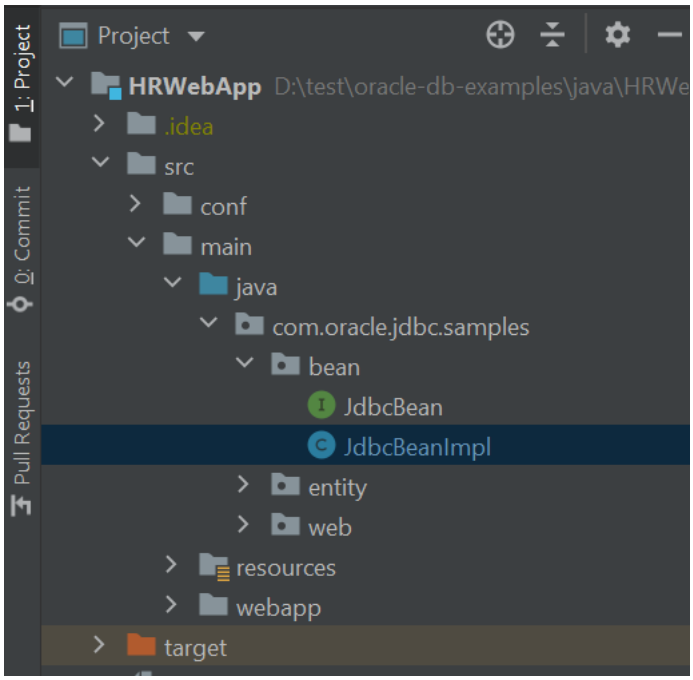
1. In the Project window on the left, double-click on **pom.xml** file.



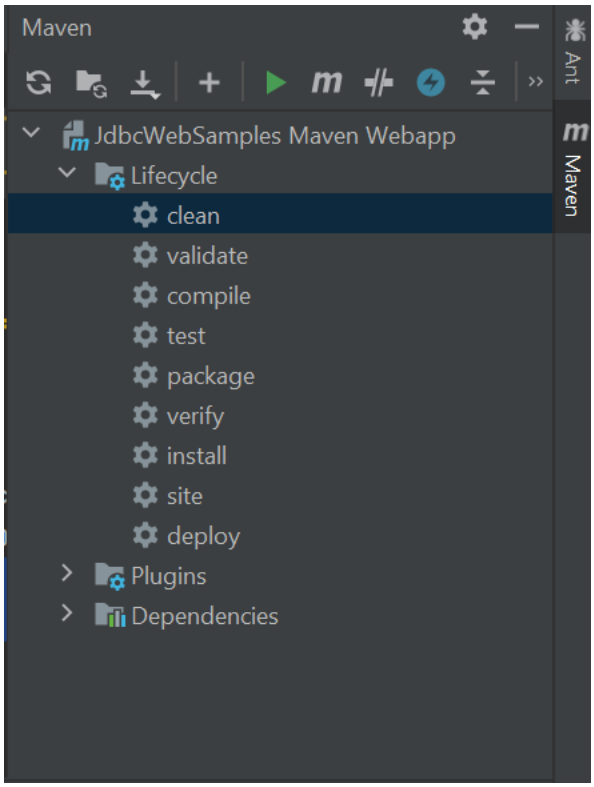
2. Add the following dependency under the `<dependencies>` tag in `pom.xml` file. See JDBC Drivers for more information.



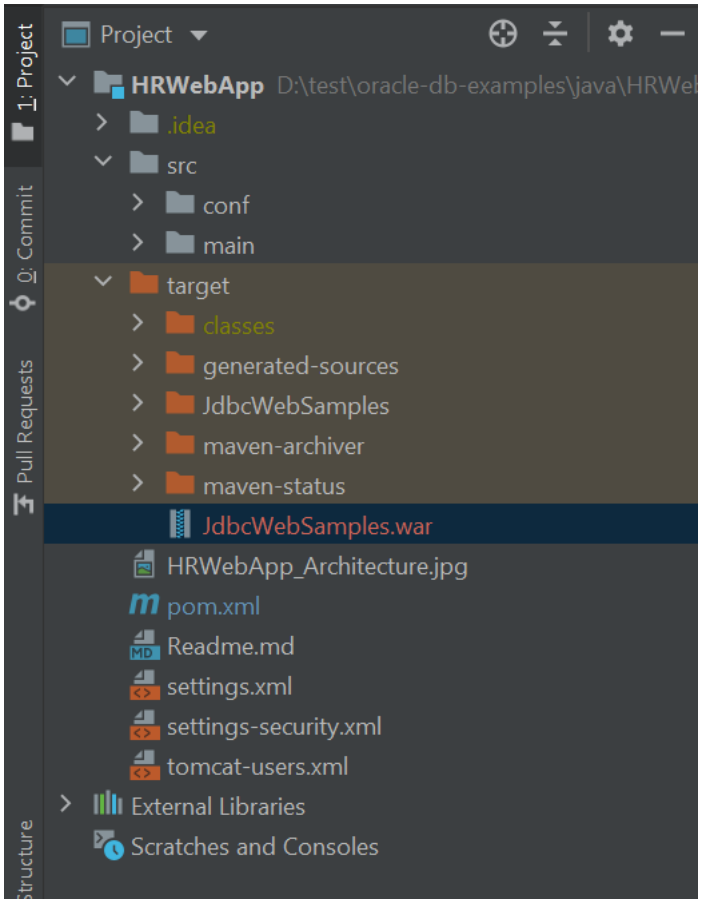
3. In the Project window, expand the **src** folder, expand **main** and then **Java** folder. Under **Java**, expand **com.oracle.jdbc.samples**. In the **bean** folder, double-click **JdbcBeanImpl.**



4. Update the *connection* variable with the details of your database connection.
5. In the **File** menu, select **Save All** to save the changes.
6. In the Maven window on the right, double-click on **clean** to clean the source code. The build progress is shown in the Run window at the bottom.



7. Similarly, double-click on **compile** and **package** in the same order.
8. Once the source code is packaged successfully, a war file is generated. Locate the `JdbcWebSamples.war` under the **target** folder.



4.6 Run the HR Web Application

The HR Web application is run using the Tomcat server.

Deploy the .war file on the Apache server

1. Navigate to the HRWebApp folder on your local machine. Under the target folder, locate `JdbcWebSamples.war` file.
2. Place `JdbcWebSamples.war` file under `TOMCAT_HOME/webapps/`.
3. Navigate to the HRWebApp folder on your local machine. Locate `tomcat-users.xml` file.
4. Place `tomcat-users.xml` file under `TOMCAT_HOME/conf/`.
5. Start the tomcat server.
6. Once the tomcat is started, access the HR web application from a browser using the URL `http://localhost:8080/JdbcWebSamples/`.

Verify the HR Web Application

1. Login to the application with either **hradmin** or **hrstaff** user.

> **Note:**
>
> - For more information about the **hradmin** and **hrstaff** users, see Overview of the HR Web Application and **Create Login and Logout Functionality**.
> - Refer tomcat-users.xml file for the username and password information required to login to the HR Web Application.

2. HR Web Application has several functionalities. **List All** displays the employees' details such as Employee_id, First_name, Last_Name, Email, Phone_number, Job_id, Salary etc. The details are retrieved from the *Employees* table and displayed on a web page. See the screenshot below that shows **List All** functionality.
3. Similarly, verify the remaining functionalities such as Search by ID, Update Employee Record and Increment Salary by giving appropriate inputs.

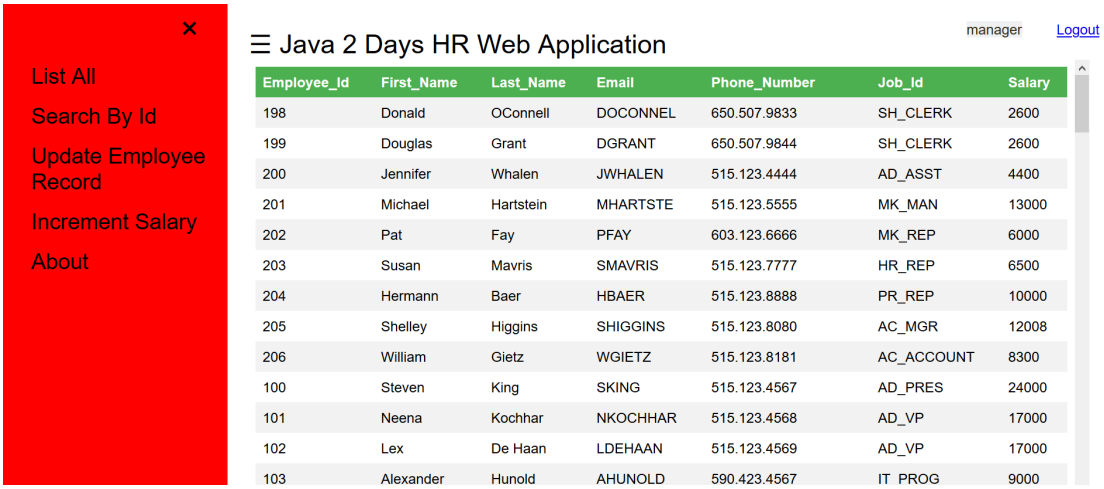
5

List All Employees

The HR web Application has several functionalities. The **List All** tab in the application displays the details of employees on the web page.

The employee details such as *Employee_id, First_name, Last_Name, Email, Phone_number, Job_id, Salary* and so on, are retrieved from the *Employees* table in the database.

In this chapter, you learn to create the basic structure of all the Java classes required to run the HR Web application. You learn to add the code required to build the **List All** functionality. You will learn how to:



1. Create `JavaBean.java` and declare a new method `getEmployees` in `JavaBean.java`.
2. Create `JavaBeanImpl.java` and implement a new method `getEmployees` in `JavaBeanImpl.java`.
3. Create `WebController.java` to process the request and response.
4. Create a HTML page `listAll.html` to display the results.
5. Create a CSS page `app.css` to be used by the HR Web Application.

> **Note:**
>
> The upcoming chapters of the guide explain the step-by-step instructions to create the Java classes and methods that you require for each functionality of the HR Web Application. Use these instructions as a reference. You can download the complete source code from GitHub. See the Github Repository Details.

5.1 Creating a Java Bean Entity for an Employee

The `Employee` class contains the getter and setter methods for all attributes of an employee. For example, the `First_name` has a getter and a setter method like `getFirst_Name` and `setFirst_Name` respectively.

Class Name: `src/main/java/com/oracle/jdbc/samples/entity/Employee.java`

Github Location: Employee.java

Steps to create Employee.java:

1. Declare the package for the class `Employee.java`.

```
package com.oracle.jdbc.samples.entity;
```

2. Import the following packages required for the `Employee` class.

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
```

3. Declare an `Employee` class. Add a pair of parenthesis ({ }). Place the cursor in between the parenthesis:

```
public class Employee {}
```

4. Declare the following variables for each one of the attributes of an employee.

```
private int Employee_Id;
private String First_Name;
private String Last_Name;
private String Email;
private String Phone_Number;
private String Job_Id;
private int Salary;
```

5. Create a constructor for the `Employee` class which takes ResultSet as the input and throws a SQLException. In this constructor, set all the values for the attributes of the `Employee` class.

```
public Employee(ResultSet resultSet) throws SQLException {
  this.Employee_Id = resultSet.getInt(1);
  this.First_Name = resultSet.getString(2);
  this.Last_Name = resultSet.getString(3);
  this.Email = resultSet.getString(4);
  this.Phone_Number = resultSet.getString(5);
  this.Job_Id = resultSet.getString(6);
  this.Salary = resultSet.getInt(7);
}
```

6. Create the Getter and Setter methods, that is, getX and setX methods to get and set the values for all attributes of the `Employee` such as `Employee_id`, `first_name`,

`last_name`, `salary`, and so on. For example, the getter and setter methods for the `Employee_Id` is as follows:

```
public int getEmployee_Id() {
   return Employee_Id;
}
  public void setEmployee_Id(int Employee_Id) {
     this.Employee_Id = Employee_Id;
  }
```

5.2 Creating a Java Bean Interface for a JDBC Connection

The `JdbcBean.java` class fetches a list of Employee objects.

Class Name: `src/main/java/com/oracle/jdbc/samples/bean/JdbcBean.java`

Github Location: JdbcBean.java

Steps to create JdbcBean.java:

1. Declare the package for the class `JdbcBean.java`.

   ```
   package com.oracle.jdbc.samples.bean;
   ```

2. Import `Employee` entity class as it contains the employee details.

   ```
   import com.oracle.jdbc.samples.entity.Employee;
   ```

3. Declare an interface `EmployeeBean` class. Inside the `EmployeeBean` class, declare a method `getEmployees()` that returns a list of `Employee` objects. Similarly, you learn to declare the methods `getEmployee(int)`, `updateEmployee(int)`, `getEmployeeByFn(String)` and `incrementSalary(int)` for other functionalities in the next few chapters.

   ```
   public interface EmployeeBean {
       public List<Employee> getEmployees();
   }
   ```

5.3 Creating a Java Bean Implementation for a JDBC Connection

The `JdbcBeanImpl.java` is an implementation class that implements all the methods declared in the `JdbcBean.java`.

Class Name: `src/main/java/com/oracle/jdbc/samples/bean/JdbcBeanImpl.java`

Github Location: JdbcBeanImpl.java

Steps to create JdbcBeanImpl.java:

1. Create a method `getConnection()` to establish a connection to the database. Ensure that you update the connection string, the database username and the database password to point to your database.

   a. Declare the package for the `JavaBean.java` class. Import the `Employee` class as it contains the employee details.

   ```
   package com.oracle.jdbc.samples.bean;
   import com.oracle.jdbc.samples.entity.Employee;
   ```

   b. Import the other dependent classes as shown in the following code snippet. If a particular class is not imported, then IntelliJ displays a message reminding you to import the required package.

   ```
   import java.sql.*;
   import java.util.ArrayList;
   import java.util.List;
   import java.util.logging.Level;
   import java.util.logging.Logger;

   import java.sql.PreparedStatement;
   import oracle.jdbc.OracleStatement;
   import oracle.jdbc.OracleConnection;
   import oracle.jdbc.driver.OracleDriver;
   import oracle.jdbc.OracleTypes;
   import java.sql.PreparedStatement;
   import oracle.jdbc.OracleStatement;
   import oracle.jdbc.OracleConnection;

   import oracle.ucp.jdbc.PoolDataSourceFactory;
   import oracle.ucp.jdbc.PoolDataSource;
   ```

   c. Declare the `JavaBeanImpl` class that implements `JavaBean`.

   ```
   public class JavaBeanImpln implements JavaBean {  }
   ```

   d. Inside the `JavaBeanImpl` class, create a logger to log exceptions.

   ```
   static final Logger logger =
   Logger.getLogger("com.oracle.jdbc.samples.bean.JdbcBeanImpl");
   ```

   e. Inside the `JavaBeanImpl` class, declare a static method `getConnection()`. The `getConection()` method registers the driver and establishes the database connection by passing the connection string, the database username and the database password as shown in the following code snippet.

   ```
   public static Connection getConnection() throws SQLException {
     DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
     Connection connection =
   DriverManager.getConnection("jdbc:oracle:thin:@//myorclhost:5521/
   myorcldbservice", "hr", "hr");
   ```

```
    return connection;
  }
```

2. Create a method `getEmployees()` to retrieve a list of employees from the `EMPLOYEES` table. Update the `SELECT` query to be used by choosing the columns that you want from the `EMPLOYEES` table.
   a. Inside the `JavaBeanImpl` class, declare the method `getEmployees()`.
   b. Use `try` and `catch` blocks to establish a database connection and fetch the employee details using a SQL `SELECT` statement.
   c. Store the employee details in an array `returnValue`.
   d. Catch the `SQLException` and log the message in logger.

```
public List<Employee> getEmployees() {
  List<Employee> returnValue = new ArrayList<>();
  try (Connection connection = getConnection()) {
     try (Statement statement = connection.createStatement()) {
         try (ResultSet resultSet = statement.executeQuery("
              SELECT Employee_Id, First_Name, Last_Name, Email,
Phone_Number, Job_Id, Salary
               FROM EMPLOYEES")) {
            while(resultSet.next()) {
              returnValue.add(new Employee(resultSet));
            }
        }
      }
  } catch (SQLException ex) {
logger.log(Level.SEVERE, null, ex);
    ex.printStackTrace();
  }

return returnValue;
}
```

> **Note:**
>
> This topic describes how to add the implementation method of **List All** functionality. Similarly, you learn to add `getEmployee`, `updateEmployee`, `getEmployeeByFn` and `incrementSalary` implementation methods for other functionalities of the HR Web Application in the upcoming chapters.

5.4 Creating a Servlet to Process the Request

In this section, you learn to create a Servlet to process a request.

Class Name: `src/main/java/com/oracle/jdbc/samples/web/WebController.java`

Github Location: WebController.java

Description: This is the main servlet that controls all the flows of the application. For every new functionality of the application, we will be adding the code to handle the new requests and responses in `doPost()` and `processResponse()` respectively.

Steps to create a Servlet:

1. Declare the package for the `WebController.java`. Import `Employee`, `EmployeeBeanImpl` and Google GSON for displaying the Employee results and other dependent classes as shown below. If the particular class is not imported, then IntelliJ will display a message reminding you to import the required package.

```
package com.oracle.jdbc.samples.web;
import com.oracle.jdbc.samples.entity.Employee;
import com.oracle.jdbc.samples.bean.EmployeeBean;
import com.oracle.jdbc.samples.bean.EmployeeBeanImpl;
import com.google.gson.Gson; import
com.google.gson.reflect.TypeToken;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.logging.Logger;
```

2. Add the annotation to the Servlet.

```
@WebServlet(name = "WebController", urlPatterns = {"/
WebController"})
```

3. Declare the `WebController` class that extends `HttpServlet`. Initialize `jdbcBean` of the type `JdbcBeanImpl`. This will be a global variable and available for all the methods such as `reportError()`, `processRequest()`, and `doGet()` to use.

```
public class WebController extends HttpServlet {
    JdbcBean jdbcBean = new JdbcBeanImpl();
}
```

4. Create the `reportError()` method to capture and display the error on the web page.

```
private void reportError(HttpServletResponse response, String
message)
throws ServletException, IOException {
  response.setContentType("text/html;charset=UTF-8"); /*Set the
response content type to be “text/html” and charset=UTF-8*/

 /*Print the error message*/
 try (PrintWriter out = response.getWriter()) {
```

```
    out.println("<!DOCTYPE html>");
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet WebController</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>" + message + "</h1>");
    out.println("</body>");
    out.println("</html>");
  }
}
```

5. Create the `processRequest` method to create processes that requests for HTTP `GET` and `POST` methods.

```
protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
  Gson gson = new Gson();
  List<Employee> employeeList = null;

   if ((value = request.getParameter(LOGOUT)) != null) {

/* Get session and then invalidate it */

    HttpSession session = request.getSession(false);
    if (request.isRequestedSessionIdValid() && session != null) {
      session.invalidate();
    }
    handleLogOutResponse(request,response);
    response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
  }
else {
   /*Instantiate the employeeList object by invoking getEmployees method
of JavaBean*/
    employeeList = jdbcBean.getEmployees();
  }

if(employeeList != null) {
    response.setContentType("application/json");  /*Set the content type
to 'application/json' */

 /* Invoke the toJson(…) method and convert the employeeList to JSON*/
    gson.toJson(employeeList,
        new TypeToken<ArrayList<Employee>>() {
        }.getType(),
        response.getWriter());
  }
/*Add an else condition to cover the error scenario when the employeeList
is empty*/
else {
    response.setStatus(HttpServletResponse.SC_NOT_FOUND);
  }
```

```
}
```

6. Create the `handleLogOutResponse(request, response)` method to edit the cookie information when a user is logging out of the application.

```
private void handleLogOutResponse(HttpServletRequest request,
HttpServletResponse response) {
  Cookie[] cookies = request.getCookies();
  for (Cookie cookie : cookies) {
    cookie.setMaxAge(0);
    cookie.setValue(null);
    cookie.setPath("/");
    response.addCookie(cookie);
  }
}
```

7. Create the `doGet` method to get the employee details from the database and show the results in JSON. JSON will be the output format of the results that is shown on the HTML.

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
  processRequest(request, response);
}
```

8. Create the `getServletInfo` method to display generic information about the servlet.

```
public String getServletInfo() {
  return "JdbcWebServlet: Reading Employees table using JDBC and
transforming it as a JSON.";
}
```

5.5 Create an HTML Page to Display Results

This section describes the steps to create a HTML page that displays the list of employees retrieved from the database.

Class Name:`src/main/webapp/listAll.html`

Github Location: listAll.html

Steps to create the HTML page:

1. Create the title, stylesheet, and body for the HTML page.

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<meta charset="UTF-8">
<title>List all Employees</title>
<link rel="stylesheet" type="text/css" href="css/app.css" >
</head>
```

2. Inside the <script> tag, declare few variables for the URL and HTTPRequest.

```
<script>
var xmlhttp = new XMLHttpRequest();
var url = "WebController";
```

3. Define the action to be performed when the requests are sent, that is, when the links for each one of the functionalities is selected.

```
xmlhttp.onreadystatechange=function() {
     if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
       processResponse(xmlhttp.responseText);
     }
   }
xmlhttp.open("GET", url, true);
xmlhttp.send();
```

4. Create the function `processResponse()` to display JSON results on HTML page.

```
function processResponse(response) {
// Process the JSON response into an array.
var arr = JSON.parse(response);
var i;
var out - "<table>";
keys = Object.keys(arr[0]);
//Print headers
out += "<tr>"
for(i = 0; i < keys.length; ++i) {
  out += "<th>"+keys [i]+"</th>"
}
out += "</tr>";
// Print values
for(j = 0; j < arr.length; j++) {
  out += "<tr>"
  for(i = 0; i < keys.length; ++i) {
    out += "<td>"+arr[j][keys[i]]+"</td>"
  }
out += "</tr>"
}
out += "</table>";
document.getElementById("id-emp").innerHTML = out;
}
```

5.6 Create a CSS File

The stylesheet has the color, font, and style specifications for all the UI elements such as buttons, side navigation, main page, links, etc., on the web page.

Class Name: `src/main/webapp/css/app.css`

Github Location : app.css

Steps to use the CSS file:

1. Download the `app.css` file.
2. Include the stylesheet in all the HTML pages of the HR Web application by adding the following line of code in the `<head>` section of the HTML page.

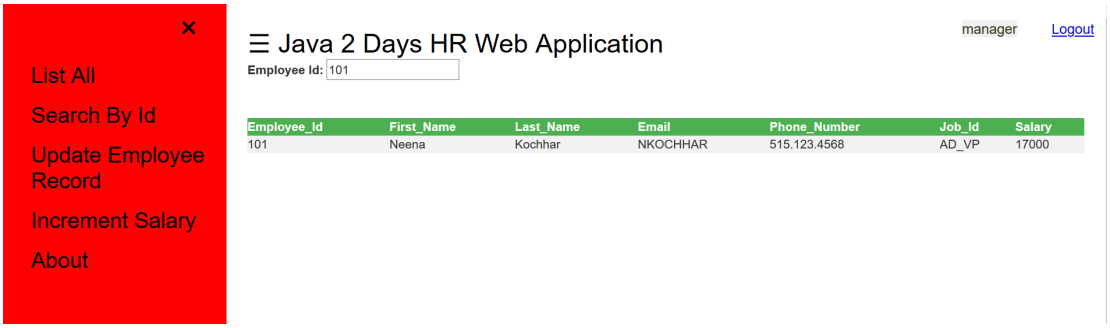
```
<link rel="stylesheet" type="text/css" href="css/app.css">
```

6

Search by Employee ID

Search by Employee Id searches for a particular employee based on the given employee Id.

The employee ID is the primary key in the `EMPLOYEES` table. You enter a valid employee ID in the HR Web application and submit to fetch the corresponding employee details from the database.



In this chapter, you learn to add code required to build the **Search by Employee ID** functionality. You will learn how to:

1. Declare a new method `getEmployee(int)` in `JavaBean.java`.
2. Implement a new method `getEmployee(int)` in `JavaBeanImpl.java`.
3. Add new code to `WebController.java` to process the request and response.
4. Create a HTML page `listById.html` to display the results.

6.1 Jdbc Java Bean

Declare and implement a new method `getEmployee(int)` in `JBedbcan.java` and `JdbcBeanImpl.java` respectively. This method takes the employee Id as an input parameter and returns an object of type Employee.

Class Name: `src/main/java/com/oracle/jdbc/samples/bean/JdbcBean.java`

Github Location: JdbcBean.java

Steps to declare the new method:

1. Open the `JdbcBean.java` file in IntelliJ. To create the JdbcBean.java class, refer to Creating a Java Bean Interface for a JDBC Connection. Use the same class and declare new methods for each one of the functionalities.
2. Add the following code snippet to declare the `getEmployee(int)` method which takes the employee Id as input:

```
public List<Employee> getEmployee(int empId);
```

Class Name: `src/main/java/com/oracle/jdbc/samples/bean/JdbcBeanImpl.java`

Github Location: JdbcBeanImpl.java

Steps to implement the new method:

1. Open the `JdbcBeanImpl.java` file in IntelliJ. To create the `JdbcBeanImpl.java` class, refer to Creating a Java Bean Implementation for a JDBC Connection. Use the same class and add new implementation methods for each one of the functionalities.
2. Add the following code snippet to implement the `getEmployee(int)` method:

```
public List<Employee> getEmployee(int empId) {
  List<Employee> returnValue = new ArrayList<>();

/*Get the database connection*/
  try (Connection connection = getConnection()) {
/* Insert the SQL statement to fetch an employee using the employee
Id */
    try (PreparedStatement preparedStatement =
connection.prepareStatement(
        "SELECT Employee_Id, First_Name, Last_Name, Email,
Phone_Number, Job_Id, Salary FROM EMPLOYEES WHERE Employee_Id
= ?")) {
        preparedStatement.setInt(1, empId);       /* Set the input
parameter */
        try (ResultSet resultSet =
preparedStatement.executeQuery()) {
        if(resultSet.next()) {    /* Check if the resultSet has any
value */
          returnValue.add(new Employee(resultSet));
        }
      }
    }
  } catch (SQLException ex) {    /* Catch the SQLException and log
the message in logger*/
logger.log(Level.SEVERE, null, ex);
    ex.printStackTrace();
  }

return returnValue;   /* Return the List of Employees */
}
```

6.2 Add the code to a Servlet to process the request

Add the relevant code to `WebController.java` to search by employee Id.

Class Name: `src/main/java/com/oracle/jdbc/samples/web/WebController.java`

Github Location: WebController.java

Steps to add the code:

1. Open the `WebController.java` class. To create the `WebController.java`, refer to Creating a Servlet to Process the Request. Use the same class and add the required code.

2. Declare a variable `ID_KEY` to capture the employee id. This is a global variable, hence, declare it outside the method `processRequest()` but within the `WebController` class.

```
private static final String ID_KEY = "id";
```

3. The method `processRequest()` is already created in the **ListAll** feature. Now, we add the code to implement **Search by employee id** functionality. Declare a variable value of the String type to capture the input from the user.

```
String value = null;
```

4. Add an IF condition to handle the new functionality. Get the employee id entered by the user and invoke the method `getEmployee(int)` to verify if the employee record exists.

```
if ((value = request.getParameter(ID_KEY)) != null) {
      int empId = Integer.valueOf(value).intValue();
      employeeList = employeeBean.getEmployee(empId);
   } else {
      // Previously used getEmployees() method for Listall feature
      employeeList = employeeBean.getEmployees();
   }
```

6.3 Create a New HTML for Search by Employee Id

A HTML page that shows an input placeholder for the user to enter the employee id. If the employee record is found, then the details of the employee is displayed on the page, otherwise, an error message will be displayed.

Class Name:`src/main/webapp/listById.html`

Github Location: listById.html

Steps to create the HTML page:

1. Create the title, stylesheet, and body for the HTML page.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>List Employee by Id</title>
<!-- Specify the stylesheet here -->
<link rel="stylesheet" type="text/css" href="css/app.css" >
<!-- Bootstrap JS for the UI -->
<link rel="stylesheet"
href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/
bootstrap.min.css">
</head>
```

2. Start the `<body>` tag and a `<input>` tag for capturing the employee id.

```
<body>
<div><label>Employee Id: </label>
<input id="empId" type="textfield"
```

```
onkeypress="return waitForEnter(event)"\>
</div>
<br/>
<br/>
<script>
function waitForEnter(e) {
  if (e.keyCode == 13) {
    var tb = document.getElementById("empId");
    fetchElementById(tb.value)
    return false;
  }
}
<script>
var xmlhttp = new XMLHttpRequest();
var url = "WebController";
```

3. Define the action when a request is sent, that is, when a link for any one of the functionality is selected.

```
xmlhttp.onreadystatechange=function() {
     if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
       processResponse(xmlhttp.responseText);
     }
   }
xmlhttp.open("GET", url, true);
xmlhttp.send();
```

4. Create the `processResponse()` function to display the JSON results on HTML page.

```
function processResponse(response) {
//Process the JSON respnse into an array.
var arr = JSON.parse(response);
     var i;
var out = "<table>";
keys = Object.keys(arr[0]);

// Print Headers
out += "<tr>"
for(i = 0; i < keys.length; ++i) {
out += "<th>"+keys[i]+"</th>"
}
out += "</tr>";
// Print values
for(j = 0; j < arr.length; j++) {
out += "<tr>"
for(i = 0; i < keys.length; ++i) {
out += "<td>"+arr[j][keys[i]]+"</td>"

}
out += "</tr>"
}
out += "</table>";
```

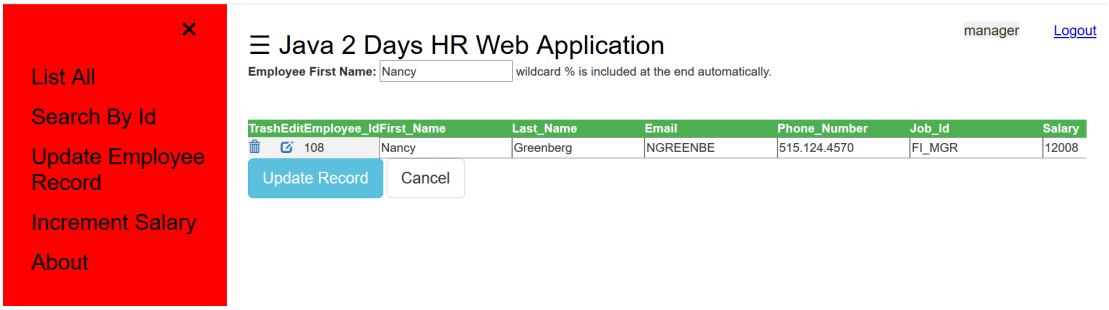
```
document.getElementById("id-emp").innerHTML = out;
}
```

7

Update an Employee Record

The **Update** functionality modifies an employee record in the database according to the user edits on the web page.

First, you must search for an employee in the records. Once you retrieve the information related to the employee, you will find the **Edit** button to modify details related to the employee.



In this chapter, you learn to add code required to build the **Update** functionality. You will learn how to:

1. Declare a new method `getEmployeeByFn(String)` in `JavaBean.java`.
2. Declare a new method `updateEmployee(int)` in `JavaBean.java`.
3. Implement a new method `getEmployeeByFn(String)` in `JavaBeanImpl.java`.
4. Implement a new method `updateEmployee(int)` in `JavaBeanImpl.java`.
5. Add new code to `WebController.java` to process the request and response.
6. Create a HTML page `listByName.html` to display the results.

> **Note:**
>
> The **hradmin** user has the privilege to update an employee record. The **hrstaff** user does not have the privilege to update an employee record.

7.1 Declare a new method *getEmployeeByFn(String)* in EmployeeBean.java

To modify the details of an employee, the **hradmin** must first search for the employee based on his/her first name. The `getEmployeeByFn(String)` method searches employees based on their first name.

Class Name: `src/main/java/com/oracle/jdbc/samples/bean/EmployeeBean.java`

Github Location: EmployeeBean.java

Steps to declare the new method:

1. Open the `JdbcBean.java` file in IntelliJ. To create the JdbcBean.java class, refer to Creating a Java Bean Interface for a JDBC Connection. Use the same class and declare new methods for each one of the functionalities.
2. Declare a method `getEmployeeByFn(String)` that takes first name as an input parameter.

```
public List<Employee> getEmployeeByFn(String fn);
```

7.2 Declare a new method *updateEmployee(Employee)*

The `updateEmployee(Employee)` method updates the attributes of an employee such as first name, last name, salary, job_id and so on.

Class Name: `src/main/java/com/oracle/jdbc/samples/bean/JavaBean.java`.

Github Location: EmployeeBean.java

Steps to declare a new method:

1. Open the `JdbcBean.java` file in IntelliJ. To create the JdbcBean.java class, refer to Creating a Java Bean Interface for a JDBC Connection. Use the same class and declare new methods for each one of the functionalities.
2. Declare a method `updateEmployee(Employee)` that takes Employee object as an input parameter.

```
public Employee updateEmployee(int empId);
```

7.3 Implement a New Method `getEmployeebyFn()` for Search by Employee name

The `getEmployeeByFn(String)` method takes the employee id as the input parameter and returns an object of type Employee.

Class Name: `src/main/java/com/oracle/jdbc/samples/bean/JdbcBeanImpl.java`

Github Location: EmployeeBeanImpl.java

Steps to implement the method:

1. Open the `JdbcBeanImpl.java` file in IntelliJ. To create the `JdbcBeanImpl.java` class, refer to Creating a Java Bean Implementation for a JDBC Connection. Use the same class and add new implementation methods for each one of the functionalities.
2. Add the following code snippet to implement the `getEmployeeByFn(String)` method:

```
public List<Employee> getEmployeeByFn(String fn) {
/* Declare an array to store the returned employee list */
  List<Employee> returnValue = new ArrayList<>();
```

```
/* Get the database connection */
  try (Connection connection = getConnection()) {
/* Insert the SQL statement to fetch an employee using the employee first
name */
try (PreparedStatement preparedStatement = connection.prepareStatement(
"SELECT Employee_Id, First_Name, Last_Name, Email, Phone_Number, Job_Id,
Salary FROM EMPLOYEES WHERE First_Name LIKE ?")) {
      /* Set the input parameter as the first name */
      preparedStatement.setString(1, fn + '%');
      try (ResultSet resultSet = preparedStatement.executeQuery()) {
while(resultSet.next()) {                  /* Check if the resultSet has
any value */
          returnValue.add(new Employee(resultSet));
        }
      }
    }
  } catch (SQLException ex) {  /* Catch the SQLException and log the
message in logger*/
logger.log(Level.SEVERE, null, ex);
    ex.printStackTrace();
  }

/* Return the list of employees from the method */
return returnValue;
}
```

7.4 Implement a new method *updateEmployee(Employee)*

The `updateEmployee(Employee)` method enables you to update the employee details such as first_name, last_name, and so on in the employee record.

Class Name: `src/main/java/com/oracle/jdbc/samples/bean/EmployeeBeanImpl.java`

Github Location: EmployeeBeanImpl.java

Steps to Implement a new method:

1. Open the `JdbcBeanImpl.java` file in IntelliJ. To create the `JdbcBeanImpl.java` class, refer to Creating a Java Bean Implementation for a JDBC Connection. Use the same class and add new implementation methods for each one of the functionalities.
2. Add the following code snippet to implement the `updateEmployee(Employee)` method:

```
public String updateEmployee(Employee employee) throws SQLException {
  /*Declare and initialize a variable to capture the number of records
updated*/
  int updateCount = 0;

/* Get the database connection*/
  try (Connection connection = getConnection()) {
    try (PreparedStatement preparedStatement =
connection.prepareStatement(
    /* Insert the SQL statement to select an employee based on the
```

```
employee id */
      "UPDATE employees SET FIRST_NAME = ?, LAST_NAME = ?, EMAIL
= ?, PHONE_NUMBER = ?,
         SALARY = ? WHERE EMPLOYEE_ID = ?")) {
        /*Set the new values entered by the user for each attribute
          and execute the prepapredStatement */
             preparedStatement.setString(1,
employee.getFirst_Name());
             preparedStatement.setString(2,
employee.getLast_Name());
             preparedStatement.setString(3, employee.getEmail());
             preparedStatement.setString(4,
employee.getPhone_Number());
             preparedStatement.setInt(5, employee.getSalary());
             preparedStatement.setInt(6,
employee.getEmployee_Id());
             updateCount = preparedStatement.executeUpdate();
      }
   }catch (SQLException ex) {   /* Catch the SQLException and log
the message in the logger*/
      logger.log(Level.SEVERE, "Unable to update record", ex);
      throw new SQLException("Alert! Record could not be updated,
"+ex.getMessage(), ex);
    }

/* Log the message with the number of records updated to the logger
*/
   logger.fine("Update count: " +updateCount);
/* If none of the records were updated, enter an alert message */
   if (updateCount != 1) {
     logger.severe("Unable to update record");
     throw new SQLException("Alert! Record could not be updated");
  }
/* Return the success message if the record was updated */
  return "Success: Record updated";
}
```

7.5 Add the code to a Servlet to process the request

Add the relevant code to `WebController.java` to update an employee.

Class Name: `src/main/java/com/oracle/jdbc/samples/web/WebController.java`

Github Location: WebController.java

Steps to add the code:

1. Open the `WebController.java` class. To create the `WebController.java`, refer to Creating a Servlet to Process the Request. Use the same class and add the required code.

2. Declare a variable `FN_KEY` to capture first name of the employee. This is a global variable, hence, declare it outside the method `processRequest()` but within the `WebController` class.

```
private static final String FN_KEY = "firstName";
```

3. The method `processRequest()` is already created in the **ListAll** feature. Now, we add the code to implement **Update an Employee** functionality. Add an ELSEIF condition to handle the new functionality. Get the employee id entered by the user and invoke the method `getEmployee(int)` to verify if the employee record exists.

```
if ((value = request.getParameter(ID_KEY)) != null) {
      int empId = Integer.valueOf(value).intValue();
      employeeList = employeeBean.getEmployee(empId);
   }
/* New code added below */
else if ((value = request.getParameter(FN_KEY)) != null) {
  employeeList = jdbcBean.getEmployeeByFn(value);
}
else {
      /* Previously used getEmployees() method for Listall feature */
      employeeList = employeeBean.getEmployees();
   }
```

7.6 Create a New HTML for Search by Employee Id

A HTML page that shows an input placeholder for the user to enter the employee first name. If the employee record is found, then the details of the employee is displayed on the page, otherwise, an error message will be displayed.

Class Name:`src/main/webapp/listById.html`

Github Location: listByName.html

Steps to create the HTML page:

1. Create the title, stylesheet, and body for the HTML page.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>List Employee by Id</title>
<!-- Specify the stylesheet here -->
<link rel="stylesheet" type="text/css" href="css/app.css" >
<!-- Bootstrap JS for the UI -->
<link rel="stylesheet"
href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/
bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/
jquery.min.js"></script>
</head>
```

2. Start the `<body>` tag and a `<input>` tag for capturing the employee id.

```
<body>
<div><label>Employee First Name: </label>
  <input id="firstName" type="textfield"
         onkeypress="return waitForEnter(event)"\> wildcard % is
included at the end automatically.</div>
<br/>
<br/>
<div id="id-emp"></div>
<div id="UpdateButton">  <button type="button" class="btn btn-info
btn-lg"
  onclick='javascipt:confirmUpdate()'>Update Record</button>
<button type="button"
  class="btn btn-default btn-lg"
onclick='javascipt:cancelUpdate()'>Cancel</button>
</div>
```

3. Define the action when a request is sent, that is, when a link for any one of the functionality is selected.

```
$('#UpdateButton').hide();
// keys;

function waitForEnter(e) {
  if (e.keyCode == 13) {
    fetchElement($("#firstName").val());
    return false;
  }
}

function fetchElement(firstName) {
  var xmlhttp = new XMLHttpRequest();
  var url = "WebController?firstName=" +firstName;

  xmlhttp.onreadystatechange=function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      processResponse(xmlhttp.responseText);
    }
  }

  xmlhttp.open("GET", url, true);
  xmlhttp.send();
}
```

4. Create the `processResponse()` function to display the JSON results on HTML page.

```
function processResponse(response) {
  var arr = JSON.parse(response);
  if (arr == null || arr.length == 0) {
    out = '<div class="alert alert-warning"><strong>Alert!</strong>'
```

```
     +' No records found for the given Fist_Name</div>'
  }
  else {
    var i;
    var out = "<table>";
    // keys is global so that it can be used later as well
    keys = Object.keys(arr[0]);

    // Print headers
    out += "<tr><th>Trash</th><th>Edit</th>"

    for(i = 0; i < keys.length; ++i) {
      out += "<th>"+keys[i]+"</th>"
    }
    out += "</tr>";

    // Print values
    for(j = 0; j < arr.length; j++) {
      pk = arr[j][keys[0]];
          out += '<tr><td><a href="javascript:confirmDelete(\'' +pk
+'\')">'
          +'<span  class="glyphicon glyphicon-trash"></span>'
          +'</a></td>'
          +'<td><a href="javascript:allowEditSalary(\'' +pk +'\')">'
          +'<span class="glyphicon glyphicon-edit"></span>'
          +'</a></td>';
      // 0 is the primary key
      for(i = 0; i < keys.length; ++i) {
        // creating an id to each column
        out += "<td id='" +pk +'_' +keys[i] +"'> "+arr[j][keys[i]]+"</
td>";
      }
      out += "</tr>"
    }
    out += "</table>";
  }
  $('#id-emp').html(out);

}
```

5. Add the allowEditSalary(pk) function to make the field names editable once the employee record is displayed.

```
function allowEditSalary(pk) {
  // If the edit button is pressed already
  if(typeof currentPK != 'undefined' && currentPK == pk) {
    console.log('Make column readonly');
    for(i = 1; i < keys.length; ++i) {
      var x = '#' +pk +"_" +keys[i];
      var value = $(x).text().trim();
      console.log(value);
      $(x).val(value);
    }
    $('#UpdateButton').hide();
```

```
    currentPK = '';
  }
  else{
    currentPK = pk;
    for(i = 1; i < keys.length; ++i) {
      var x = '#' +pk +"_" +keys[i];
      var value = $(x).text().trim();
      $(x).html("<input type='text' value='" +value +"' \>");
    }
    $('#UpdateButton').show();
  }
}
```

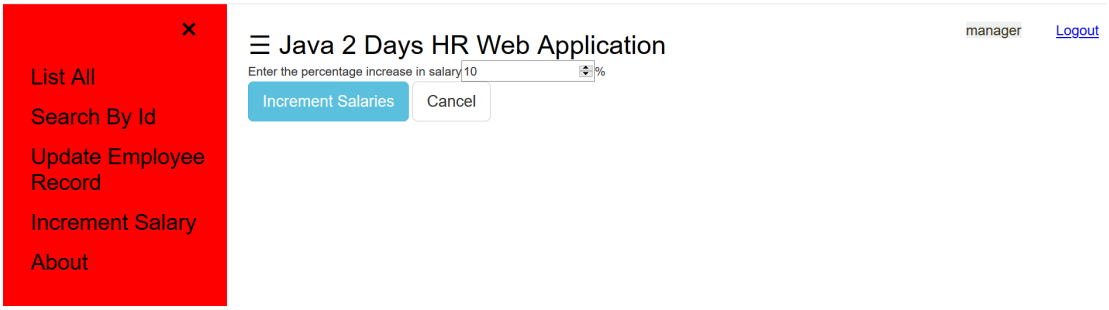
6. Add the `confirmUpdate()` and `cancelUpdate()` functions to define the confirm and cancel actions respectively.

```
function confirmUpdate() {
  var res = confirm("Do you really want to Update");
  if(res == true) {
    console.log("Udating record");
      $('#UpdateButton').hide();
  }
  else {
    console.log("Record not updated");
  }
}

function cancelUpdate() {
   if(typeof currentPK != 'undefined') {
    console.log('Make column readonly');
    for(i = 1; i < keys.length; ++i) {
      var x = '#' +pk +"_" +keys[i];
      var value = $(x).text().trim();
      console.log("cancelUpdate: " +value);
      $(x).text(value);
    }
    $('#UpdateButton').hide();
    currentPK = '';
  }
}
```

8
Increment Salary

The Increment Salary functionality modifies the salaries of all employees by incrementing the values according to the input percentage.

Enter a percentage for salary hike in the placeholder on the web page. Click confirm to modify the salaries of all employees in the database table. You can verify the changes by clicking on the **List All** tab.



In this chapter, you learn how to add code required to build the **Increment Salary** functionality. You will learn how to:

1. Declare a new method `incrementSalary(int)` in `JavaBean.java`.
2. Implement a new method `incrementSalary(int)` in `JavaBeanImpl.java`.
3. Add new code to `WebController.java` to process the request and response.
4. Create a HTML page `incrementSalary.html` to display the results.

8.1 Declare a new method *incrementSalary(int)*

The `incrementSalary(int)` method updates the salary value of all employees by incrementing the value according to a given percentage.

Class Name: `src/main/java/com/oracle/jdbc/samples/bean/JavaBean.java`.

Github Location: JavaBean.java

Steps to declare a new method:

1. Open the `JdbcBean.java` file in IntelliJ. To create the JdbcBean.java class, refer to Creating a Java Bean Interface for a JDBC Connection. Use the same class and declare new methods for each one of the functionalities.
2. Declare a method `incrementSalary(int)` that takes an integer for percentage as an input parameter.

```
public List<Employee> incrementSalary(int incrementPct);
```

8.2 Implement a new method *incrementSalary(int)*

The `incrementSalary(int)` method enables you to increment the salary of all employees according to a given percentage.

Class Name: `src/main/java/com/oracle/jdbc/samples/bean/JavaBeanImpl.java`

Github Location: JavaBeanImpl.java

Steps to Implement a new method:

1. Open the `JdbcBeanImpl.java` file in IntelliJ. To create the `JdbcBeanImpl.java` class, refer to Creating a Java Bean Implementation for a JDBC Connection. Use the same class and add new implementation methods for each one of the functionalities.
2. Add the following code snippet to implement the `incrementSalary(int)` method:

```
public List<Employee> incrementSalary (int incrementPct) {
  List<Employee> returnValue = new ArrayList<>();


/* Get the database connection*/
  try (Connection connection = getConnection()) {
    try (CallableStatement callableStatement =
        connection.prepareCall("begin ? :=
refcur_pkg.incrementsalary(?); end;")) {
          callableStatement.registerOutParameter(1,
OracleTypes.CURSOR);
          callableStatement.setInt(2, incrementPct);
          callableStatement.execute();
          try (ResultSet resultSet = (ResultSet)
callableStatement.getObject(1)) {
            while (resultSet.next()) {
              returnValue.add(new Employee(resultSet));
            }
          }
    }
  } catch (SQLException ex) {
/* Catch the SQLException and log the message in the logger*/
    logger.log(Level.SEVERE, null, ex);
    ex.printStackTrace();
  }

return returnValue;
}
```

8.3 Add the Code to a Servlet

Add the relevant code to `WebController.java` to give a salary raise to all employees.

Class Name: `src/main/java/com/oracle/jdbc/samples/web/WebController.java`

Github Location: WebController.java

Steps to add the code:

1. Open the `WebController.java` class. To create the `WebController.java`, refer to Creating a Servlet to Process the Request. Use the same class and add the required code.
2. Declare the variables `INCREMENT_PCT` and to capture the salary increment percentage. This is a global variable, hence, declare it outside the method `processRequest()` but within the `WebController` class.

```
private static final String INCREMENT_PCT = "incrementPct";
```

3. Add the `doPost(req, res)` method as follows:

```
protected void doPost(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException {
  Map<String,String[]> x = request.getParameterMap();
  String value = null;
  if ((value = request.getParameter(INCREMENT_PCT)) != null) {
    Gson gson = new Gson();
    response.setContentType("application/json");
    List<Employee>  employeeList =
jdbcBean.incrementSalary(Integer.valueOf(value));
    gson.toJson(employeeList,
        new TypeToken<ArrayList<Employee>>() {
        }.getType(),
        response.getWriter());
  }
else {
    response.setStatus(HttpServletResponse.SC_NOT_FOUND);
  }
}
```

8.4 Create a new HTML for Increment Salary

The incrementSalary.html page displays an input box to enter the percentage for calculating the salary increase.

Class Name: `src/main/webapp/incrementSalary.html.`

Github Location: incrementSalary.html

Steps to create the HTML page:

1. Create the title, stylesheet, and body of the HTML page.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Increment Salary</title>
```

```
<link rel="stylesheet" type="text/css" href="css/app.css" >
<link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/
bootstrap/3.3.6/css/bootstrap.min.css">
<script    src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/
jquery.min.js">script    src="https://ajax.googleapis.com/ajax/libs/
jquery/1.12.2/jquery.min.js"></script>
</head>
```

2. Start the `<body>` tag and a `<input>` tag for capturing the percentage for salary raise.

```
<body>
<div> Enter the percentage increase in
salary<inputid='incrementField' type="number" max="100" min="3">%
</div>
<div id="UpdateButton"> <button type="button" class="btn btn-info
btn-lg" onclick='javascipt:confirmUpdate()'> Increment Salaries</
button> <button type="button" class="btn btn-default btn-lg"
onclick='javascipt:cancelUpdate()'>Cancel</button></div>
<div id="status" class="none"></div>
<div id="id-emp"></div>
<script>
function showStatus(c, message) {
     $('#status').text(message);
     $('#status').attr('class', c);
       }

function confirmUpdate() {
  var increment = $('#incrementField').val();
  var res = confirm("Do you really want to Increment Salary by "
+increment +"%?");
  if(res == true) {
    console.log("Salary record");
    $('#UpdateButton').hide();
    showStatus("alert alert-info", "Updating records, processing
request");
    var xmlhttp = new XMLHttpRequest();
    var url = "WebController?op=incrementSalary&incrementPct="
+increment;
    xmlhttp.onreadystatechange = function() {
      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        processResponse(xmlhttp.responseText);
        showStatus("alert alert-success", "Updating records,
successfully updated");
      }
      else {
        showStatus("alert alert-danger", "Updating records,
failure, could not update records");
      }
    }
    xmlhttp.open("POST", url, true);
    xmlhttp.send();
    showStatus("alert alert-info", "Updating records, request
sent");
```

```
  }
  else {
    console.log("Salary not updated");
    showStatus("alert alert-warning", "Updating records, attempt
cancelled");

  }
}

</script>
```

3. Create the function `processRequest()` to display the JSON results on HTML page.

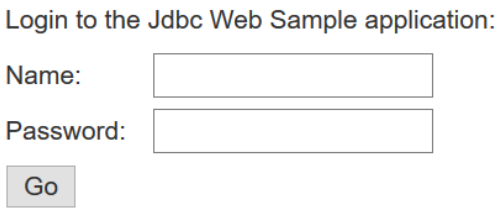
```
unction processResponse(response) {
  var arr = JSON.parse(response);
  var i;
  var out = "<table>";
keys = Object.keys(arr[0]);

 /* Print headers */
  out += "<tr>"
for(i = 0; i < keys.length; ++i) {
    out += "<th>"+keys[i]+"</th>"
}
out += "</tr>";

/* Print values */
for(j = 0; j < arr.length; j++) {
out += "<tr>"
for(i = 0; i < keys.length; ++i) {
out += "<td>"+arr[j][keys[i]]+"</td>"
}
out += "</tr>"
}
out += "</table>";
document.getElementById("id-emp").innerHTML = out;
}
```

9

Create Login and Logout Functionality

The HR Web Application has two users, namely `hradmin` and `hrstaff`.

Once you login to the HR Application, you see the landing page, with details of the web application. The `hradmin` and `hrstaff` have different privileges and access to different features.



This chapter shows the required classes and additional code required to build the **Login** and **Logout** functionality in the application.

- Create a XML file `tomcat-users.xml` for login functionality.
- Create a HTML page `login.html` to login the user.
- Create a HTML page `login-failed.html` to display the error message.
- Create a `web.xml` to authenticate the users during login.
- Create a HTML page `about.html` to show more details about the application.
- Create a landing page `index.html` and define the html pages for redirection.
- Add code to the servlet `WebController.java` to process logout.

9.1 Create tomcat-users.xml

Create an XML file `tomcat-users.xml` and list down the users you want to allow access to. Specify both the username and password for each one of the users.

Class Name: `/java/HRWebApp/tomcat-users.java`

Github Location: tomcat-users.xml

Steps to create the xml file:

1. Create the file `tomcat-users.xml`.

```
<?xml version='1.0' encoding='utf-8'?> z
<tomcat-users xmlns="http://tomcat.apache.org/xml"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
version="1.0">
<role rolename="manager"/>
<role rolename="staff"/>
<user username="hradmin" password="welcome" roles="manager,staff"/>
<user username="hrstaff" password="welcome" roles="staff"/>
</tomcat-users>
```

2. Place this file under `TOMCAT_HOME/conf/tomcat-users.xml` on your machine.

9.2 Create login.html

The login page is displayed when you invoke the main page of the web application. The login page displays the fields to capture the username and password.

Class Name: `src/main/webapp/login.html`

Github Location: login.html

Steps to create the HTML page:

1. Create the title, head, and stylesheet for `login.html`.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Login to Jdbc Web Sample application</title>
<link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/
bootstrap/3.3.6/css/bootstrap.min.css">
<style>
#cent {
    position:absolute;
    top:50%;
    left:50%;
    margin-top:-50px; /* this is half the height of your div*/
    margin-left:-100px; /*this is half of width of your div*/
  }
```

```
    td {
      height: 30px;
    }

</style>
</head>
```

2. Create the `<body>` and `<form>` to submit the login credentials entered by the user.

```
<body>
<div id="cent">
<form method="POST" action="j_security_check">
<table>
<tr>
<td colspan="2">Login to the Jdbc Web Sample application:</td>
</tr>
<td>Name:</td>
<td><input type="text" name="j_username" /></td>
</tr>
<tr>
<td>Password:</td>
<td><input type="password" name="j_password"/></td>
</tr>
<tr>
<td colspan="2"><input type="submit" value="Go" /></td>
</tr>
</table>
</form>
</div>
</body>
```

9.3 Create `login-failed.html`

A html page to display the error messgae if the login is unsuccessful.

Class Name: `src/main/webapp/login-failed.html`

Github Location: login-failed.html

Steps to create the HTML page:

1. Create the `login-failed.html` as shown below.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Login Failed</title>
</head>
<body>
<p>
Sorry, login failed!
</p>
```

```
</body>
</html>
```

9.4 Create `web.xml`

The `web.xml` file consists of descriptors to authenticate the users when the login page is displayed to the user.

Class Name: `src/main/webapp/WEB-INF/web.xml`

Github Location: web.xml

Steps to create the xml file:

1. Use the following code to create the `web.xml` file.

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
<display-name>Jdbc Web Sample</display-name>
<security-role>
<role-name>manager</role-name>
</security-role>
<security-role>
<role-name>staff</role-name>
</security-role>
<security-constraint>
<web-resource-collection>
<web-resource-name>Wildcard means whole app requires
authentication</web-resource-name>
<url-pattern>/*</url-pattern>
<http-method>GET</http-method>
<http-method>POST</http-method>
</web-resource-collection>
<auth-constraint>
<role-name>manager</role-name>
</auth-constraint>
<user-data-constraint>
<transport-guarantee>NONE</transport-guarantee>
</user-data-constraint>
</security-constraint>
<security-constraint>
<web-resource-collection>
<web-resource-name>Wildcard means whole app requires
authentication</web-resource-name>
<url-pattern>/*</url-pattern>
<http-method>GET</http-method>
</web-resource-collection>
<auth-constraint>
<role-name>staff</role-name>
</auth-constraint>
<user-data-constraint>
<transport-guarantee>NONE</transport-guarantee>
</user-data-constraint>
```

```
  </security-constraint>
<login-config>
<auth-method>FORM</auth-method>
<form-login-config>
<form-login-page>/login.html</form-login-page>
<form-error-page>/login-failed.html</form-error-page>
</form-login-config>
</login-config>
</web-app>
```

9.5 Create about.html

The about.html file displays information about the HR Application, users and functionalities.

Class Name: src/main/webapp/about.html

Github Location: about.html

Steps to use the HTML page: Download the about.html and use it in your application.

9.6 Create index.html

The index.html file consists of all details about the HR Web Application. It describes in detail its users and functionalities.

Class Name: src/main/webapp/index.html

Github Location: index.html

Steps to create the HTML page:

1. Create the title, head, and stylesheet for index.html.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Employee table listing</title>
<link rel="stylesheet" type="text/css" href="css/app.css" >
<style>
iframe:focus {
outline: none;
}
iframe[seamless] {
display: block;
}
</style>
</head>
<body>
```

2. Create <body> and actions for the features through navigation links and logout.

```
<body>
<div id="sideNav" class="sidenav">
```

```
<a href="javascript:void(0)" class="closebtn" onclick="closeNav()"
class="staff">×</a>
<a href="javascript:switchSrc('listAll.html')" class="staff">List
All</a>
<a href="javascript:switchSrc('listById.html')"
class="staff">Search By Id</a>

<a href="javascript:switchSrc('listByName.html')"
class="manager">Update Employee Record</a>
<a href="javascript:switchSrc('incrementSalary.html')"
class="manager">Increment Salary</a>
<a href="javascript:switchSrc('about.html')">About</a>
</div>
<div id="main">
<div align="right">
<div
id="myrole"
        style="display:inline; color:#393318; display: block;
background-color:#eff0f1;position: absolute; top: 20px; right: 8%;"
    >myrole</div>
<a href="javascript:void(0)"
       onclick="logout()"
       class="staff"
       style="display: block; position: absolute; top: 20px; right:
1%">Logout</a>
</div>
<div>
<span style="font-size:30px;cursor:pointer" onclick="openNav()">
Java Get Started HR Web Application        </span>
</div>
<div>
<iframe id="content"
src="about.html"
frameborder="0"
style="overflow:hidden; height:100%; width:100%"
height="100%"
width="100%"></iframe>
</div>
</div>
<script>
function openNav() {
  document.getElementById("sideNav").style.width = "256px";
  document.getElementById("main").style.marginLeft = "256px";
}

function closeNav() {
  document.getElementById("sideNav").style.width = "0";
  document.getElementById("main").style.marginLeft= "0";
}

function switchSrc(src) {
  document.getElementById('content').src = src;
}

function logout() {
```

```
  var xmllogout = new XMLHttpRequest();
  xmllogout.open("GET", "WebController?logout=true", true, "_", "_");
  xmllogout.withCredentials = true;
  // Invlalid credentials to fake logout
  xmllogout.setRequestHeader("Authorization", "Basic 00001");
  xmllogout.send();

  xmllogout.onreadystatechange = function() {
    window.location.replace("index.html");
  }

 return true;
}

var xmlhttp = new XMLHttpRequest();
var url = "getrole";

xmlhttp.onreadystatechange = function() {
  if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
    role = xmlhttp.responseText;
    console.log("role: " +role);
    if (role == "staff") {
      console.log ("disabling manager");
      var x = document.getElementsByClassName('manager');
      for(i = 0; i < x.length; ++i) {
        x[i].style.display = 'none';
      }
    }
    document.getElementById('myrole').innerHTML = ' '+role+' ';
  }
}
xmlhttp.open("GET", url, true);
xmlhttp.send();
</script>
</body>
```

9.7 Add the code to a Servlet to process the request

Add the relevant code to `WebController.java` to login and logout of the application.

Class Name: `src/main/java/com/oracle/jdbc/samples/web/WebController.java`

Github Location: WebController.java

Steps to add the code:

1. Open the `WebController.java` class. To create the `WebController.java`, refer to Creating a Servlet to Process the Request. Use the same class and add the required code.

2. Declare a variable `LOGOUT` to capture the status of the user. This is a global variable, hence, declare it outside the method `processRequest()` but within the `WebController` class.

```
private static final String LOGOUT = "logout";
```

3. The method `processRequest()` is already created in the **ListAll** feature. Now, we add the code to implement **Logout** functionality. Create an `if` block to verify the functionality you will invoke based on input. Check if the input value is `LOGOUT`.

```
if ((value = request.getParameter(LOGOUT)) != null) {
/* Getting session and then invalidating it */

    HttpSession session = request.getSession(false);
   if (request.isRequestedSessionIdValid() && session != null)    {
        session.invalidate();
   }
handleLogOutResponse(request,response);
   response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
return;
}
```

4. If the input value is `LOGOUT`, invoke the method to handle the logout of user.

```
private void handleLogOutResponse(HttpServletRequest request,
HttpServletResponse response) {
    Cookie[] cookies = request.getCookies();
    for (Cookie cookie : cookies) {
      cookie.setMaxAge(0);
      cookie.setValue(null);
      cookie.setPath("/");
      response.addCookie(cookie);
    }
  }
```

10
Best Practices

1. Use Database Service on Cloud:

Use the Oracle Database Service on Cloud (DBCS) to create a database on cloud. DBCS comes with an in-built HR schema and tables that you can use when you build the HR web application.

> **Note:**
>
> Use Oracle Database 21c to use features and functionalities of the latest Oracle Database.

2. JDBC Driver, UCP:

It is recommend that you use the latest 12.2.0.1 versions of JDBC drivers and UCP.

> **Note:**
>
> Download the latest JDBC drivers and UCP from 12.2.0.1 JDBC driver and UCP

3. JDK 8

It is recommended that you use the latest version of Oracle JDBC driver 12.2.0.1, that is compliant with JDK 8.

4. Auto-closeable statements

Starting JDK7, 'Auto-closeable statements' has been introduced, that close by default without an explicit `catch` statement.

5. Use PreparedStatement instead of Statement objects:

Statement in JDBC must be localized to being used for `DDL` (`ALTER, CREATE, GRANT` etc) since these commands cannot accept bind variables.

Use `PreparedStatement` or `CallableStatement` for any other type of statement. These statements can accept bind variables.

11
Troubleshooting and Debugging

1. Tomcat log file:

Check `TOMCAT_HOME/logs/catalina.out` for any errors after deploying the application.

2. Additional Logging:

Enable logging in Tomcat to find logging messages in the code.

> **Note:**
>
> Refer https://tomcat.apache.org/tomcat-8.0-doc/logging.html for more information

Debugging UI Related Issues

1. Browser Version:

This application has been tested on Firefox (version 52) and Chrome (version 58) successfully.

2. Browser Console:

Look for errors in the browser console to find and debug issues.

3. Monitor Network Traffic:

Track network traffic to find out the requests being made, and the responses to the requests. A return status higher than 400 indicates errors. If you find errors in the range of 400 t 500 inspect the Tomcat log files for debugging.

Inspect the `JSON` responses from various calls to the backend server.

4. Additional Logging:

Edit the packaged HTML files, and add `console.log(...)` to add extra logging.

Index