Forte™ for Java™ 4, Enterprise Edition チュートリアル

Forte for Java 4

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650-960-1300

Part No. 816-7448-10
2002 年 6 月 Revision A

原典： *Forte for Java 4, Enterprise Edition Tutorial*
Part No: 816-4057-10
Revision A

目次

図目次

表目次

はじめに

Forte™ for Java™, Enterprise Edition チュートリアルにようこそ。このチュートリアルでは、次のような Enterprise Edition の機能の使用方法を説明しています。

- EJB™ 2.0 ビルダー -『Enterprise JavaBeans™ Specification, Version 2.0』に基づく、チュートリアルアプリケーションの Enterprise JavaBeans™ コンポーネントを開発、作成する
- EJB モジュールアセンブリ - EJB コンポーネントを EJB モジュールにアセンブルし、EJB Java Archive (JAR) ファイルにエクスポートする
- テストアプリケーション機能 - J2EE™ リファレンス実装 (Reference Implementation: RI)、 バージョン 1.3.1 をアプリケーションサーバーとして使用し、クライアントを手動で作成せずに エンタープライズ Bean をテストする
- Web サービス機能 - 既存の EJB コンポーネントから SOAP RPC に基づくWeb サービスを構築し、Web ブラウザで表示可能な JSP™ ページを生成する
- J2EE リファレンス実装 - J2EE RI のカスタマイズされたバージョン1.3.1 を使用し、チュートリアルアプリケーションを配備およびテストする

このマニュアルで説明しているプログラム例は、実際に作成することができます。作業環境については、以下の Web サイトにあるリリースノートを参照してください。

`http://sun.co.jp/forte/ffj/documentation/index.html`

使用するプラットフォームによっては、このマニュアルに掲載している画面イメージと異なることがあります。その場合でも表示上の違いはわずかであるため、内容を理解するのには問題ありません。ほとんどの手順で Forte for Java のユーザーインタ

フェースを使用しますが、場合によっては、コマンド行にコマンドを入力する必要があります。その場合は、次のように、Microsoft Windows の「コマンドプロンプトウィンドウ」でのプロンプトと構文が例として示されています。

```
c:\>cd MyWorkSpace\MyPackage
```

UNIX® や Linux 環境では、次のようなプロンプトとなり、¥ マーク (またはバックスラッシュ) ではなくスラッシュを使用します。

```
% cd MyWorkSpace/MyPackage
```

お読みになる前に

このチュートリアルでは、Java 2 Platform, Enterprise Edition (J2EE™) Blueprints リソースに記載されているアーキテクチャに準拠したアプリケーションを作成します。J2EE 準拠のアプリケーションを作成、開発、配備するために Forte for Java 4, Enterprise Edition の機能を使用するには、このチュートリアルが役に立ちます。

チュートリアルを開始する前に、次の内容をよく理解しておく必要があります。

- Java™ プログラミング言語
- Enterprise JavaBeans の概念
- Java サーブレット構文
- JDBC™ 対応のドライバ構文
- JavaServer Pages™ 構文
- HTML 構文
- リレーショナルデータベースの概念 (表やキーなど)
- データベースの使用方法
- J2EE アプリケーションのアセンブリと配備の概念

また、次に示すような J2EE の概念に関する一般的な知識が必要です。

- *Java 2 Platform, Enterprise Edition Blueprints*
  `http://java.sun.com/j2ee/blueprints`
- *Java 2 Platform, Enterprise Edition Specification*
  `http://java.sun.com/j2ee/download.html#platformspec`

- *The J2EE Tutorial* (for J2EE SDK version 1.3)
  `http://java.sun.com/j2ee/tutorial/1_3-fcs/index.html`
- *Java Servlet Specification Version 2.3*
  `http://java.sun.com/products/servlet/download.html#specs`
- *JavaServer Pages Specification Version 1.2*
  `http://java.sun.com/products/jsp/download.html#specs`

さらに、SOAP (Simple Object Access Protocol) バージョン 1.1 の Apache implementation の詳細に精通していると役立ちます。Apache SOAP は Apache XML プロジェクトのサブプロジェクトです。詳細は、以下を参照してください。

`http://xml.apache.org/soap/index.html`

注 - Sun では、本マニュアルに掲載した第三者の Web サイトのご利用に関しましては責任はなく、保証するものでもありません。また、これらのサイトあるいはリソースに関する、あるいはこれらのサイト、リソースから利用可能であるコンテンツ、広告、製品、あるいは資料に関して一切の責任を負いません。Sun は、これらのサイトあるいはリソースに関する、あるいはこれらのサイトから利用可能であるコンテンツ、製品、サービスのご利用あるいは信頼によって、あるいはそれに関連して発生するいかなる損害、損失、申し立てに対する一切の責任を負いません。

内容の紹介

このマニュアルは、初めから順を追って読むことを前提に作成されています。チュートリアルの各章は、前の章で作成したコードに基づいて構成されています。

第 1 章では、このチュートリアルを学ぶにあたって準備しておく必要がある、Forte for Java 4 IDE（integrated development environment) の起動方法、IDE が適切な Web サーバーとアプリケーションサーバーを使用しているかどうかの確認方法、およびチュートリアルのデータベース表の作成方法を説明します。また、インストールされる Forte for Java 4 のディレクトリの一覧と説明も記載されています。

第 2 章 では、チュートリアルアプリケーションの機能と構造について説明します。

第 3 章では、チュートリアルアプリケーションの EJB 層を作成する手順と、それぞれの Bean をテストするテストアプリケーション機能の使用方法を説明します。

第 4 章では、EJB 層からチュートリアルの Web サービスを作成するためのForte for Java 4 IDE の使用方法と、Web サービスのテスト方法を説明します。

第 5 章では、提供されている Swing クライアントが、第 4 章でモジュールから生成された出力にどのようにアクセスするか、またチュートリアルアプリケーションがどのように実行されるかを説明します。

付録 A は、このチュートリアルアプリケーションのソースファイルの全内容のまとめです。

付録 B は、このチュートリアルアプリケーションのデータベーススクリプトの全内容のまとめです。

書体と記号について

次の表と記述は、このマニュアルで使用している書体と記号について説明しています。

書体または記号	意味	例
`AaBbCc123`	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コーディング例。	`.login` ファイルを編集します。 `ls -a` を使用してすべてのファイルを表示します。 `machine_name% You have mail.`
`AaBbCc123`	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して表わします。	`machine_name%` **`su`** `Password:`
AaBbCc123 または ゴシック	コマンド行の可変部分。実際の名前または実際の値と置き換えてください。	`rm` *filename* と入力します。 `rm` **ファイル名** と入力します。
『 』	参照する書名を示します。	『Solaris ユーザーマニュアル』

書体または記号	意味	例
「」	参照する章、節、または、強調する語を示します。	第 6 章「データの管理」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合、バックスラッシュは、継続を示します。	`machinename% grep '^#define \` `  XV_VERSION_STRING'`
➤	階層メニューのサブメニューを選択することを示します。	作成:「返信」➤「送信者へ」

シェルプロンプトについて

シェル	プロンプト
UNIX の C シェル	`machine_name%`
UNIX の Bourne シェルと Korn シェル	`machine_name$`
スーパーユーザー (シェルの種類を問わない)	#

関連マニュアル

Forte for Java のマニュアルは、Acrobat Reader (PDF) ファイル、オンラインヘルプ、サンプルアプリケーションの `Readme` ファイル、Javadoc™ 文書の形式で提供しています。

オンラインで入手可能なマニュアル

次のマニュアルは、Forte for Java のポータルサイトおよび `docs.sun.com` の Web サイトから入手できます。

Forte for Java ポータルサイトでのマニュアルの入手先は、
`http://sun.co.jp/forte/ffj/documentation/index.html` です。
`docs.sun.com` の URL は、`http://docs.sun.com` です。

- リリースノート (HTML 形式)

  Forte for Java の Edition ごとに用意されています。このリリースでの変更情報と技術上の注意事項を説明しています。

- インストールガイド (PDF 形式)

  Forte for Java の Edition ごとに用意されています。対応プラットフォームへの Forte for Java のインストール手順を説明しています。さらに、システム要件、アップグレード方法、Web サーバーやアプリケーションサーバーのインストール、コマンド行での操作、インストールされるサブディレクトリ、Javadoc の設定、データベースの統合、アップデートセンターの使用方法などが含まれます。

- Forte for Java プログラミングシリーズ (PDF 形式)

  Forte for Java の各機能を使用して優れた J2EE アプリケーションを開発するための方法を詳細に説明しています。

  - 『Web コンポーネントのプログラミング』

    JSP ページ、サーブレット、タグライブラリを使用し、クラスやファイルをサポートする Web アプリケーションを J2EE Web モジュールとして構築する方法を説明しています。

  - 『J2EE アプリケーションのプログラミング』

    EJB モジュールや Web モジュールを J2EE にアセンブルする方法を説明しています。また、J2EE アプリケーションの配備や実行についても説明しています。

  - 『Enterprise JavaBeans コンポーネントのプログラミング』

    Forte for Java の EJB ビルダーウィザードや、他の IDE コンポーネントを使用し、EJB コンポーネント (コンテナ管理や Bean 管理の持続性の機能を持つセッション Bean やエンティティ Bean) を作成する方法を説明しています。

  - 『Web サービスのプログラミング』

    Web サービスモジュールが提供するツールを使用して Web サービスを構築する方法を説明しています。Web サービスは、XML (Extensible Markup Language) 文書の形式で提供されるアプリケーションビジネスサービスであり、HTTP を介して配信されます。

- 『Java DataBase Connectivity の使用』

  Forte for Java の JDBC 生産性向上ツールを使用し、JDBC アプリケーションを作成する方法について説明しています。

- Forte for Java チュートリアル (PDF 形式)

  Forte for Java の Edition ごとに用意されています。Forte for Javaのツールを使用してアプリケーションを作成する方法を、順を追って説明しています。

  チュートリアルアプリケーションは、以下のサイトからもアクセスできます。

  `http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`

`docs.sun.com` (`http://docs.sun.com`) の Web サイトでは、他のサンのマニュアルの参照、印刷、購入をすることもできます。

オンラインヘルプ

オンラインヘルプは、Forte for Java 開発環境内から参照できます。ヘルプキー (Solaris オペレーティング環境では Help キー、Windows および Linux 環境では F1 キー) を押すか、「ヘルプ」 >「内容」を選択します。ヘルプの項目と検索機能が表示されます。

プログラム例

Forte for Java の機能を紹介したプログラム例とチュートリアルアプリケーション (各 Edition のチュートリアルで説明されているアプリケーションを含む) を、以下の Forte for Java のポータルサイトからダウンロードすることができます。

`http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`

Javadoc

Javadoc 形式のマニュアルは、Forte for Java の多くのモジュールに用意されており、IDE の中で参照できます。このマニュアルの使用方法については、リリースノートを参照してください。IDE を起動すると、エクスプローラの Javadoc タブで Javadoc マニュアルを参照できます。

ご意見の送付先

Sun のマニュアルについてのご意見やご要望をお寄せください。今後のマニュアル作成の参考にさせていただきます。次のアドレスまで電子メールをお送りください。

`docfeedback@sun.com`

電子メールのタイトルに、対象マニュアルの Part No. を明記してください。

第1章

チュートリアルを開始する前に

この章では、このチュートリアルを開始する前に準備しておく必要がある事柄をまとめています。便宜を考えて、このチュートリアルには、『インストールガイド』と重複するインストール情報が含まれています。

この章の内容は次のとおりです。

注 - このマニュアルには、「DiningGuide アプリケーションファイル」の名前を参照している箇所が出てきます。それらのファイルは、完成したチュートリアルアプリケーション、そのアプリケーションの実行方法を説明した readme ファイル、必要なデータベース表を作成するための SQL スクリプトファイルなどのことです。これらのファイルを zip 形式で圧縮したものを `http://forte.sun.com/ffj/documentation/tutorialsandexamples.html` から入手することができます。

チュートリアルアプリケーションのソフトウェア要件

この節では、このチュートリアルを学ぶにあたって用意しておく必要があるソフトウェアをまとめています。Forte for Java 4 統合開発環境 (IDE) の実行に必要なコンポーネントだけでなく、チュートリアルアプリケーションの作成と実行に必要なコンポーネントをすべて揃えておく必要があります。

全般的なシステム要件については、リリースノートまたは Forte for Java Developer Resources ポータルサイトの Documentation ページをご覧ください。
`http://forte.sun.com/ffj/documentation/index.html`

Forte for Java 4 IDE の実行に必要なコンポーネント

Forte for Java 4 IDE には、Java 2 Platform, Standard Edition (Java 2 SDK) が必要です。IDE をインストールすると、インストーラがユーザーのシステム内で SDK ソフトウェアを検索し、システム上に正しいバージョンがインストールされていない場合は、その旨をユーザーに通知して IDE のインストールを停止します。Java 2 SDK の適切なバージョンは、以下の Java Developer のポータルサイト からダウンロードできます。
`http://java.sun.com/j2se/`

チュートリアルアプリケーションの作成と実行に必要なコンポーネント

チュートリアルアプリケーションの作成および実行には、この後に示すソフトウェアが必要です。その一部は、Forte for Java 4, Enterprise Edition をインストールすると、デフォルトの設定で 一緒にインストールされます。これらソフトウェアのサポートされているバージョンについては、以下のサイトにあるリリースノートをご覧ください。
`http://forte.sun.com/ffj/documentation/index.html`

- Web ブラウザ

  チュートリアルアプリケーションのページを表示するには Web ブラウザが必要です。

- Web サーバー

  Forte for Java 4 テストクライアントは、Web サーバーを必要とする Web アプリケーションです。このチュートリアルでは、IDE 上でのテストに必要な Web サーバー機能を提供する、埋め込み版 Tomcat (バージョン 4.0) を使用します。

- データベースソフトウェア - PointBase Server 4.2 Restricted Edition

  このチュートリアルアプリケーションはデータベースにアクセスします。このため、このチュートリアルでは、Forte for Java 4 IDE とともにインストールできる PointBase Network Server ソフトウェアの使用方法を説明します。Forte for Java 4 を自分でインストールしなかった場合は、Forte for Java 4 がインストールされているディレクトリに `pointbase` ディレクトリが存在するかどうかを調べることによって、PointBase がインストールされているかどうかを確認することができます。PointBase がインストールされていない場合は、再度 IDE のインストーラを起動してインストールを行うことができます。

- アプリケーションサーバー

  このチュートリアルの Java 2 Platform, Enterprise Edition (J2EE™) アプリケーションを配備するにはアプリケーションサーバーが必要です。このため、このチュートリアルでは、J2EE RI (J2EE リファレンス実装) サーバーの使用方法を説明します。必ず、Forte for Java 4, Enterprise Edition に付属しているバージョン 1.3.1 の J2EE RI を使用してください。このバージョンの J2EE RI サーバーは、PointBase Network Server データベースを必要とします。

  IDE は、デフォルトで J2EE RI を使用するように設定されます。この確認方法については、8 ページの「デフォルトのアプリケーションサーバーと Web サーバーの確認」を参照してください。

Forte for Java 4 IDE の起動

次の節で説明している手順で、Forte for Java IDE を起動します。詳細は『Forte for Java4, Enterprise Edition インストールガイド』に記載されています。

Solaris、Linux などの UNIX 環境における起動

インストールが完了すると、*forte4j-home*¥bin ディレクトリに runide.sh スクリプトが配備されます。端末ウィンドウで以下を入力して、このスクリプトを起動してください。

```
$ sh runide.sh
```

このスクリプトをカスタマイズする方法については、5 ページの「コマンド行スイッチを使用したセッションの変更」を参照してください。

Microsoft Windows における起動

IDE を起動する方法は次の 3 通りあります。

- デスクトップ上の「Forte for Java 4.0 EE」アイコンをダブルクリックする

  runidew.exe 実行可能ファイルが実行され、コンソールウィンドウなしで IDE が起動します。なお、この実行可能ファイルは *forte4j-home*¥bin ディレクトリにあり、このディレクトリにはまた runide.exe という実行可能ファイルも含まれています。runide.exe アイコンをダブルクリックすると、コンソールウィンドウを使用して IDE を起動し、ウィンドウにIDE の標準エラーと標準出力を表示します。コンソール上で Ctrl+Break キーを押すと実行スレッドのリストを表示することができます。また、Ctrl+C キーを押すとプログラムがすぐに終了します。

- 「スタート」->「プログラム」->「Forte for Java 4.0 EE」->「Forte for Java」を選択する

- コマンド行から実行可能ファイルを実行する

```
C:¥> runide.exe [switch]
```

コマンド行オプションについては、次節を参照してください。

コマンド行スイッチを使用したセッションの変更

表 1-1 で IDE の起動方法を変更するためのオプションについて説明します。詳細は『Forte for Java, Enterprise Edition インストールガイド』を参照してください。

- Microsoft Windows システムの場合

  コマンド行から IDE を実行する際にオプションを設定できます。

- Solaris、Linux などの UNIX 環境の場合

  インストールディレクトリの `bin` サブディレクトリにある `ide.cfg` ファイルを変更するか、オプションを指定して `runide.sh` を呼び出す独自のシェルスクリプトを作成できます。

表 1-1　`runide` コマンド行スイッチ

オプション	意味
`-classic`	クラシック JVM を使用する
`-cp:p` *addl-classpath*	Forte for Java 4 のクラスパスの先頭に、指定されたクラスパスを追加する
`-cp:a` *addl-classpath*	Forte for Java 4 のクラスパスの末尾に、指定されたクラスパスを追加する
`-fontsize` *size*	IDE のグラフィカルユーザーインタフェース (GUI) で使用するフォントのサイズを、指定されたサイズに設定する
`-locale` *language* [:*country*[:*variant*]]	デフォルトのロケールの代わりに、このセッションで使用するロケールを指定する
`-J`*jvm-flags*	指定されたフラグを JVM に直接渡す。 (`-J` と引数の間に空白を挿入しない)
`-jdkhome` *jdk-home-dir*	デフォルトの SDK の代わりに、指定された Java 2 SDK を使用する
`-h` 又は `-help`	GUI ダイアログを開いて、コマンド行オプション一覧表示する
`-hotspot`、`-client`、`-server`、`-classic`、`-native`、`-green` のいずれか	指定されたタイプの JVM を使用する

表 1-1　runide コマンド行スイッチ (続き)

オプション	意味
`-single`	シングルユーザーモードで IDE を実行する。このモードでは、ユーザー設定ディレクトリではなく *forte4j-home* から IDE を起動できる
`-ui` *UI-class-name*	指定されたクラスを使って IDE を実行する。IDE のルック & フィールがそのクラスのルック & フィールになる
`-userdir` *user-directory*	このセッションのユーザー設定ディレクトリとして、指定されたディレクトリを使用する。詳細は次節を参照してください

ユーザー設定ディレクトリの指定

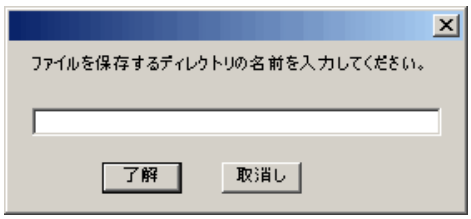
Forte for Java 4 IDE は、各ユーザー専用のディレクトリにユーザーのプロジェクトやサンプル、IDE 設定を保存します。このため、各開発者は開発作業の同期をとりながら、個別に作業および設定内容を管理することができます。

- Solaris、Linux などの UNIX 環境の場合

  `-userdir` コマンド行スイッチを使用してユーザー設定ディレクトリを明示的に指定しなかった場合、ユーザー設定はデフォルトで *user-home*`/ffjuser40ee` ディレクトリに保存されます。

- Microsoft Windows システムの場合

  Forte for Java 4 IDE を初めて起動すると、ユーザー設定ディレクトリを指定するよう求められます。`C:¥MyWork` のようにフルパスで指定します。



  この値は、後で使用できるようレジストリに記録されます。IDE の起動時に `-userdir` コマンド行スイッチを使用することによって、別のユーザー設定ディレクトリを指定することもできます。

Forte for Java 4 のディレクトリ構成

表 1-2 は、Forte for Java 4 ソフトウェアをインストールしたときにインストールディレクトリに作成されるサブディレクトリをまとめています。

表 1-2　BeanForte for Java 4 のディレクトリ構成

ディレクトリ	用途
`beans`	IDE にインストールされた JavaBeans™ コンポーネントが含まれる
`bin`	Forte for Java 4 の起動ツール (Microsoft Windows の場合は `ide.cfg` ファイルも) が含まれる
`docs`	Forte for Java 4 のヘルプファイルとその他の各種マニュアルが含まれる (リリースノートは *forte4j-home* にある)
`examples`	Enterprise Edition のサンプルコードのソースファイルと readme ファイルが含まれる
`j2sdkee1.3.1`	J2EE リファレンス実装が含まれる(インストールされている場合)
`jwsdp`	Java Web Services Developer's Pack ソフトウェアが含まれる (UDDI 内部レジストリ)
`lib`	IDE 実装の中核となる JAR ファイルとオープン API が含まれる
`modules`	Forte for Java 4 モジュールの JAR ファイルが含まれる
`pointbase`	PointBase Server 4.2 Restricted Edition データベース用の実行可能ファイルとクラス、データベース、マニュアルが含まれる (インストールされている場合)
`sources`	ライブラリのソースが含まれる。これらのソースは、ユーザーアプリケーションと一緒に再配布できる
`system`	IDE が特別な目的に使用するファイルとディレクトリが含まれる。たとえば技術サポートを依頼するときに有用な情報が含まれている `ide.log` などのファイルが含まれる
`tomcat401`	Tomcat 4.0 Web サーバーのソースが含まれる

表 1-3 は、Forte for Java 4 ソフトウェアを起動したときにユーザー設定ディレクトリに作成されるサブディレクトリをまとめています。これらサブディレクトリのほとんどは、Forte for Java 4 のホームディレクトリのサブディレクトリに対応しており、設定を保持するために使用されます。

表 1-3　ユーザー設定ディレクトリのディレクトリ構成

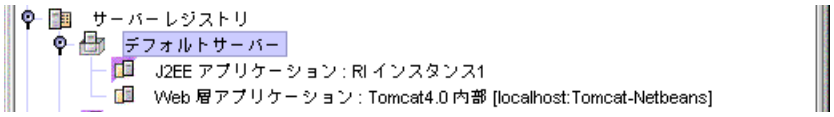
ディレクトリ	用途
`beans`	IDE にインストールされた JavaBeans コンポーネントのユーザー設定が含まれる
`javadoc`	IDE にインストールされた Javadoc ファイルのユーザー設定が含まれる
`lib`	システム `lib` ファイルのユーザー設定が含まれる
`modules`	Update Center からダウンロードされたモジュールが含まれる
`sampledir`	エクスプローラの「ファイルシステム」タブにデフォルトでマウントされるディレクトリ。このディレクトリには、IDE で作成したオブジェクトが保存される。 ただし、別のディレクトリをマウントして代わりに使用した場合は、そのマウントされたディレクトリに保存される
`sampledir/examples`	複数の NetBeans サンプルアプリケーションが含まれる
`system`	システムファイルおよびディレクトリのユーザー設定が含まれる
`tomcat401_base`	JSP ページの操作のユーザー設定が含まれる

デフォルトのアプリケーションサーバーと Web サーバーの確認

このチュートリアルの DiningGuide アプリケーションは、J2EE RI アプリケーションサーバーを使用します。また、DiningGuide の Web サービスをテストする際に生成されるテストアプリケーションは Tomcat Web サーバーを使用します。IDE のインストーラは、これらのサーバーをともにデフォルトのサーバーとして設定します。他のサーバーを使用する場合は、DiningGuide アプリケーションをテストまたは実行する前にそれらのサーバーをデフォルトサーバーに設定してください。

J2EE RI アプリケーションサーバーと Tomcat Web サーバーがデフォルトサーバーになっていることを確認するには、次の操作を行います。

1. Forte for Java 4 IDE で、エクスプローラの「実行時」タブをクリックします。
2. 「サーバーレジストリ」ノードを開き、そのサブノードの「デフォルトサーバー」を開きます。
   - IDE が正しいサーバーを使用している場合、「デフォルトサーバー」ノードは以下のように表示されます。



   - RI インスタンス 1 と Tomcal 4.0 以外のサーバーが示されている場合は、以下の操作を行います。
     i. 間違っているデフォルトサーバーのノードを右クリックして、「デフォルトサーバーを設定」を選択します。

        「デフォルト Web (またはアプリケーション) サーバーを選択」ダイアログが表示されます。

     ii. 適切なサーバーを選択して「了解」をクリックします。

チュートリアルデータベース表の作成

Forte for Java, Enterprise Edition チュートリアルを開始するには、前もって 2 つのデータベース表を作成して、PointBase Network Server データベースにインストールしておく必要があります。付録 B の SQL スクリプトを使用して、その 2 つのデータベース表を作成してください。スクリプトファイル `rest_pb.sql` は DiningGuide チュートリアルの `diningguide.zip` にも含まれており、以下の URL から 入手できます。

`http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`

付録 B のスクリプトは、表 1-4 に示すデータベーススキーマを作成します。

表 1-4　DiningGuide データベース表

表名	列	主キー	その他
Restaurant	restaurantName	○	
	cuisine		
	neighborhood		
	address		
	phone		
	description		
	rating		
CustomerReview	restaurantName	○	CustomerName との複合主キー (Restaurant 表の restaurantName を参照している)
	customerName	○	
	review		

Restaurant 表には、表 1-5 に示すレコードが含まれます。

表 1-5　Restaurant 表のレコード

restaurant -Name	cuisine	neighborhood	address	phone	description	rating
French Lemon	Mediterranean	Rockridge	1200 College Avenue	510 888 8888	Very nice spot.	5
Bay Fox	Mediterranean	Piedmont	1200 Piedmont Avenue	510 888 8888	Excellent.	5

CustomerReview 表には、表 1-6 に示すレコードが含まれます。

表 1-6　CustomerReview 表のレコード

restaurantName	customerName	comment
French Lemon	Fred	Nice flowers.
French Lemon	Fred	Excellent Service

これらのチュートリアルのデータベース表は、以下の手順に従ってデフォルトの PointBase データベースにインストールします。

注 - データベース表の作成中、Forte for Java IDE は実行したままでも、終了してもかまいません。

---

1. PointBase サーバーを起動します。

   - Solaris または Linux 環境の場合
     forte4j-home/pointbase/server ディレクトリにある Server ファイルを実行します。

   - Microsoft Windows の場合
     「スタート」->「プログラム」->「Forte for Java 4.0 EE」->「PointBase」->「Network Server」->「Server」を選択するか、*forte4j-home*¥pointbase¥server ディレクトリにある server.bat ファイルをダブルクリックします。

2. PointBase コンソールを起動します。

   - Solaris または Linux 環境の場合
     forte4j-home/pointbase/client ディレクトリにある Console ファイルを実行します。

   - Microsoft Windows の場合
     「スタート」->「プログラム」->「Forte for Java 4.0 EE」->「PointBase」->「Client Tools」->「コンソール」を選択するか、
     forte4j-home¥pointbase¥client ディレクトリにある console.bat ファイルをダブルクリックします。

     「Connect To Database」ダイアログが表示され、PointBase ドライバとデフォルトのサンプルデータベースの値が示されます。

3. 「OK」をクリックします。

   PointBase コンソールが表示されます。

4. 付録 B の PointBase スクリプトをコンソールの SQL 入力ウィンドウにコピーします。

   `diningguide.zip` ファイルをダウンロードした場合は、「File」->「Open」を選択して、`rest_pb.sql` スクリプトを開いてもかまいまん。

5. 「SQL」->「Execute All」を選択します。

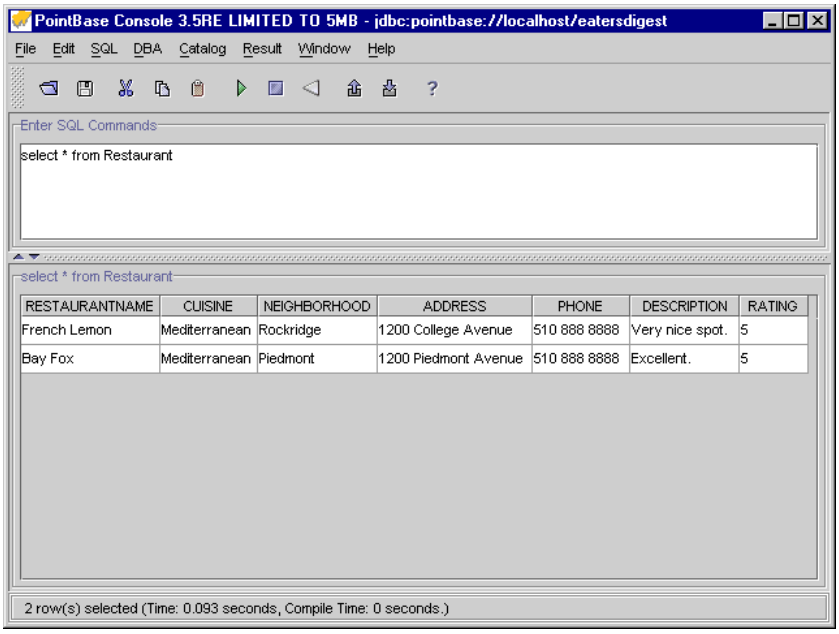
   スクリプトを実行したことを示すメッセージが表示されます。「Cannot find the table...」で始まる初期メッセージは無視してください。このメッセージが表示されるのは表に対する DROP 文があるためで、表はまだ作成されていないからです。これらの DROP 文は、後でスクリプトを再実行して表を初期化する場合に役立ちます。

6. 「Window」->「Clear Input」を選択して SQL 入力ウィンドウの内容を消去してから、次を入力することによって、表が作成されていることを確認します。

```
select * from Restaurant;
```

7. 「SQL」->「Execute」を選択します。

   コンソールに Restaurant 表が表示されます。



注 - このような表が表示されない場合は、「Window」->「Windows」を選択して、表示の種類を変更してください。

---

8. PointBase のコンソールウィンドウを閉じます。

これで、チュートリアルを開始する準備ができました。

第2章

チュートリアルアプリケーションの概要

ここでは、チュートリアルのサンプルアプリケーションを作成することで、Forte for Java 4, Enterprise Edition の機能を使用して簡単な J2EE アプリケーションを作成する方法を学習します。

この章では、作成するアプリケーションを紹介し、アプリケーションの要件、およびその要件を満たすアーキテクチャについて説明します。また、最後の節ではForte for Java 4, Enterprise Edition の機能 (EJB ビルダー、テストアプリケーション機能、新規 Web サービスウィザードなど) の使用方法について説明します。

この章の構成は次のとおりです。

チュートリアルアプリケーションの機能

チュートリアルアプリケーションである「DiningGuide」は、利用できるレストランとその特徴をまとめたリストを表示する、単純なお食事ガイドアプリケーションです。このアプリケーションのユーザーは、特定のレストランに関する利用者のコメントを表示したり、レストランのレコードに自分のコメントを追加したりできます。表示されるレストランの特徴としては、店名、料理の種類、近隣情報、所在地、電話番号、簡単な説明、評価 (1 ～ 5) などがあります。

ユーザーは、このアプリケーションのインタフェースと以下のように対話します。

- レストランの一覧を表示する
- 特定のレストランに関する利用者のコメントのリスト表示を要求する
- レストランに対するユーザー自身のコメントをコメントリストに追加する

アプリケーションとユーザーの対話シナリオ

DiningGuide アプリケーションとそのユーザーの対話は、ユーザーがデータベース内のすべてのレストランレコードを一覧表示するクライアントページを実行することで始まり、ユーザーがアプリケーションのクライアントを終了したときに終了します。このチュートリアルには、アプリケーションの機能との対話方法を具体的に示すための単純な Swing クライアントが用意されています。しかし、Web クライアントや他のアプリケーションなど、これ以外の種類のクライアントも DiningGuide アプリケーションのビジネスメソッドにアクセスすることができます。

次のシナリオは、アプリケーションでどのような対話が行われ、アプリケーションにどのような条件が求められるのかを明らかにします。

1. ユーザーがアプリケーションの `RestaurantTable` クラスを実行します。

   アプリケーションは「DiningGuide Restaurant Listing」ウィンドウを表示します。このウィンドウには、すべてのレストランの店名と料理の種類、所在地、電話番号、短いコメント、および 1 から 5 の評価が表示されます。またこのページには、「View Customer Comments」というボタンがあります。

2. ユーザーがリストからレストランのレコードを選択し、「View Customer Comments」ボタンをクリックします。

   アプリケーションは、選択されたレストランに関する利用者のコメントがすべて入った「All Customer Reviews By Restaurant Name」ウィンドウを表示します。

3. ユーザーコメントウィンドウで、ユーザーが「Customer Name」および「Review」フィールドにテキストを入力して、「Submit Customer Review」ボタンをクリックします。

   アプリケーションは入力されたユーザー名とコメントを CustomerReview データベース表に追加し、新規追加されたレコードの入った「All Customer Reviews By Restaurant Name」ウィンドウを再表示します。

4. ユーザーが「Restaurant Listing」ウィンドウに戻り、別のレストランを選択して、「View Customer Comments」ボタンをクリックします。

   アプリケーションは、選択されたレストランに対するコメントがすべて入った新しい「All Customer Reviews By Restaurant Name」ウィンドウを表示します。

アプリケーションの機能仕様

上記のような対話シナリオをサポートする DiningGuide アプリケーションのユーザーインタフェースの主な機能としては、以下が挙げられます。

- すべてのレストランデータのマスタービュー (リスト表示)
- 特定のレストランに関するすべての利用者のコメントを読み込むためのボタン (マスターレストランリストのウィンドウに表示)
- 特定のレストランに対するすべての利用者のコメントデータのマスタービュー
- 新しいコメントを追加するためのボタン (ユーザーコメントリストウィンドウに表示)
- 現在のレストランの新規ユーザー名とコメントを入力するためのテキスト入力フィールド (ユーザー批評リストウィンドウに表示)
- 入力されたコメントデータをデータベースに送信するためのボタン (ユーザー批評リストウィンドウに表示)

ユーザーから見たチュートリアルアプリケーション

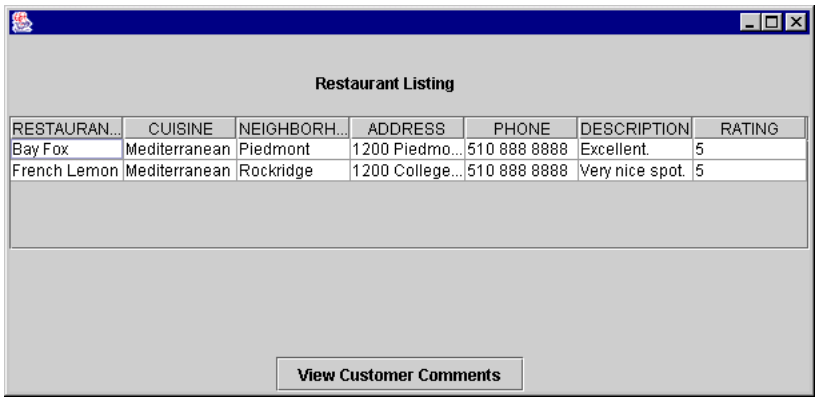
15 ページの「チュートリアルアプリケーションの機能」で説明したシナリオと機能仕様が、ユーザーから見た場合にどのように実現されるかを示します。

注 - このプログラムを実際に動かすことはできません。 以降はチュートリアルアプリケーションの動作を把握するためのイメージとして参考にしてください。

---

1. Forte for Java 4 エクスプローラで「RestaurantTable」ノードを右クリックして「実行」を選択します。

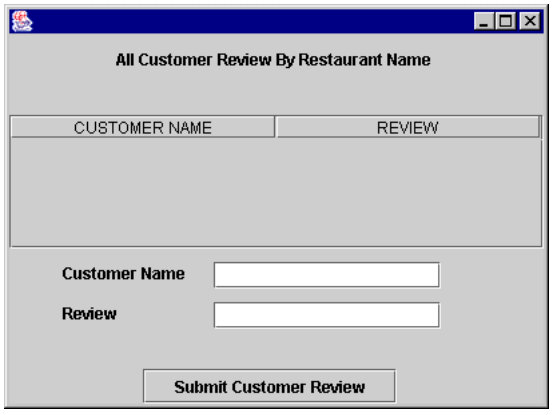
   IDE が実行時モードに切り替わります。実行ウィンドウに「Restaurant」ノードが現れ、以下のような「RestaurantTable」ウィンドウが表示されます。



   このウィンドウに表示されているのは、9 ページの「チュートリアルデータベース表の作成」で作成した Restaurant 表のデータです。

2. 特定のレストランに関する利用者のコメントを表示するには、店名を選択して、「View Customer Comments」ボタンをクリックします。

   たとえば Bay Fox レストランを選択してください。「CustomerReviewTable」ウィンドウが表示されます。



   この場合は、データベースに何もコメントがないため、レコードは表示されません。表 1-6 を参照してください。

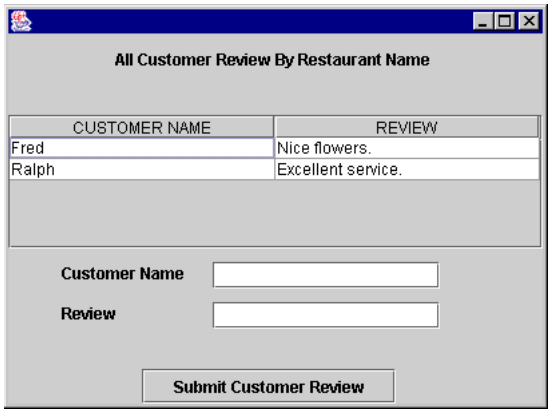
3. コメントを追加するには、利用者名とコメントを入力して、「Submit Customer Review」ボタンをクリックします。

   たとえば利用者名として `New User`、コメントとして `I'm speechless!` と入力してください。利用者コメントウィンドウが再表示されます。

   他のレストランのコメントを表示してみます。

4. 「Restaurant List」ウィンドウで「French Lemon」を選択して、「View Customer Comments」ボタンをクリックします。

   新しい「利用者コメントリスト」ウィンドウに French Lemon レストランに対するコメントが表示されます。



   利用者コメントレコードが 2 つ表示されています。表 1-6 で確認してください。

5. 続けて利用者コメントレコードを追加して、表示してみます。

6. 練習を終えたら、アプリケーションのウィンドウのいずれかを閉じることによってアプリケーションを終了します。

7. 新しい利用者コメントレコードがデータベースに書き込まれたことを確認するには、PointBase データベースコンソールを起動します。

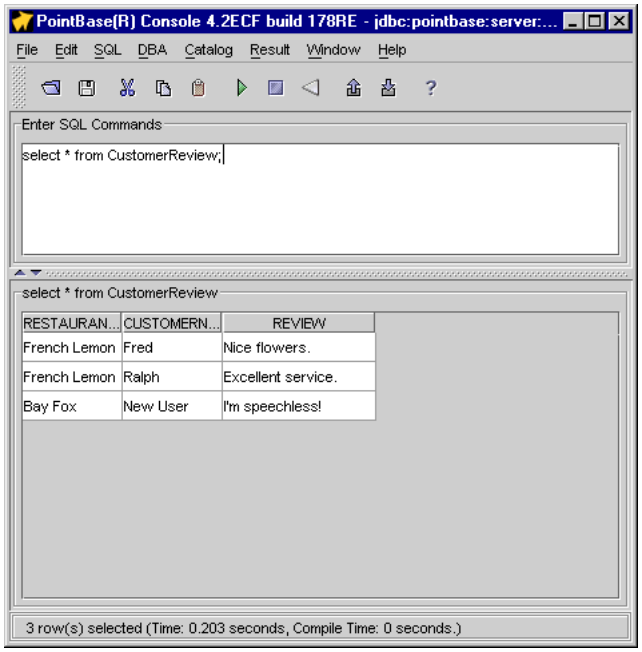
   PointBase サーバーが動作している必要があります。9 ページの「チュートリアルデータベース表の作成」の手順を参照してください。

8. 「PointBase」コンソールで次の文を入力します。

   ```
   select * from CustomerReview;
   ```

9. 「SQL」->「Execute」を選択します。

   入力したコメントの入った CustomerReview 表が、以下のようにコンソールに表示されます。



チュートリアルアプリケーションの構造

チュートリアルアプリケーションの中枢は、エンティティタイプの 2 つのエンタープライズ Bean と 2 つの詳細クラス、および1 つのセッション Bean からなる EJB 層です。2 つのエンティティ Bean は 2 つの DiningGuide データベース表 (Restaurant 表および CustomerReview 表) を示します。2 つの詳細クラスはエンティティ Bean フィールドを反映するクラスで、それぞれのフィールドの取得メソッドおよび設定メソッドが含まれます。詳細クラスを使用する目的は、データベースデータを読み込む際のエンティティ Bean に対するメソッド呼び出し回数を減らすことにあります。セッション Bean はクライアント (Web サービス経由) とエンティティ Bean 間の対話を管理します。

図 2-1 は DiningGuide アプリケーションのアーキテクチャ図です。

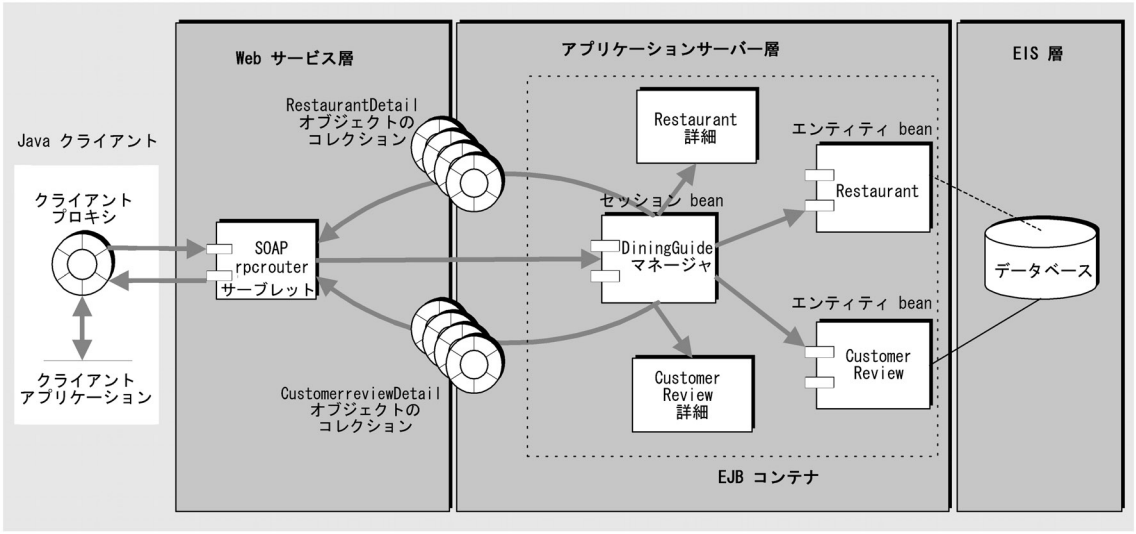


図 2-1　DiningGuide アプリケーションのアーキテクチャ

図 2-1 のクライアントには クライアントプロキシが含まれています。このプロキシは SOAP 実行環境システムを使用して、Web サーバー上の SOAP 実行システムと通信します。要求は SOAP メッセージとして渡されます。Web サービスは、この SOAP メッセージを EJB 層のセッション Bean のメソッドの呼び出しに変換します。その応答はセッション Bean が Web サービスに返し、SOAP メッセージに戻されてクライアントプロキシに渡され、最終的には表示データまたはアクションに変換されます。

SOAP 要求は XML ラッパーであり、Web サービスのメソッド呼び出しとシリアル化された形式の入力データを含みます。

アプリケーションの構成要素

ここでは、図 2-1 に示すアプリケーションの構成要素をまとめています。

- アプリケーションサービス層 (EJB 層)

  このチュートリアルでは、最初に EJB 層を作成してテストします。EJB 層は、次の要素で構成されています。

  - 2 つのエンティティエンタープライズ Bean - コンテナ管理の持続性 (Container-Managed Persistence: CMP) を使用し、アプリケーションの 2 つのデータベース表を表します。
  - 2 つの詳細クラス - 返されたデータベースレコードを保持します。
  - ステートレスセッションエンタープライズ Bean - クライアントの要求を管理し、クライアントに返されるオブジェクトをフォーマットします。

- Web サービス層
  - Web モジュール - セッション Bean のメソッド実行用のサーブレットと JSP ページが含まれます。

    このモジュールは、セッション Bean に対するテストアプリケーションを作成したときに自動的に作成されます。
  - Web サービス論理ノード - Web サービス全体を表し、Web サービスの変更や構成を可能にします。
  - クライアントプロキシ - Web サービスを配備したときに生成されます。
  - WSDL (Web Services Descriptive Language) ファイル - クライアント用の Web サービスの記述です。

- クライアント

  クライアントコンポーネントは、アプリケーションのページを表示する Swing クライアントです。第 5 章では、第 4 章で Web サービスに作成するクライアントプロキシをインスタンス化するコードを、提供されるクライアントページからコピーします。

EJB 層の詳細

DiningGuide アプリケーションの EJB 層には、エンティティタイプの 2 つの エンタープライズ Bean、2 つの詳細クラス、およびその他のコンポーネント (セッション Bean) が含まれており、このうちのセッション Bean を使用して、クライアントとエンタープライズ Bean のやりとりを管理します。

- `Restaurant` CMP EJB コンポーネント
  `Restaurant` Bean はエンティティ Bean であり、コンテナ管理の持続性 (CMP) を使用し、Restaurant データベース表のデータを表します。

- `Customerreview` CMP EJB コンポーネント
  CMP タイプのエンティティ Bean である `Customerreview` エンティティ Bean も、`CustomerReview` データベース表のデータを表します。

- `RestaurantDetail` クラス
  このコンポーネントには、`Restaurant` エンティティ Bean と同じフィールド、および各フィールドに対する getter/setter メソッド (エンティティ Bean のリモート参照からこのデータを取得するためのもの) が定義されています。このコンストラクタは、レストランデータを表すオブジェクトをインスタンス化します。また、このオブジェクトはクライアントで JSP ページ、HTML ページ、または Swing コンポーネントにフォーマットされ、表示できるようになります。

- `CustomerreviewDetail` クラス
  このコンポーネントは、`RestaurantDetail` クラスが `Restaurant` エンティティ Bean に対して機能するのと同様に、`Customerreview` エンティティ Bean に対して機能します。

- `DiningGuideManager` セッション EJB コンポーネント
  このコンポーネントはステートレスセッション Bean で、クライアントとエンティティ Bean のやりとりを管理します。

チュートリアルアプリケーションの作成に必要な作業

チュートリアルアプリケーションを作成する作業は、3 つの章に分けて説明します。まず、第 3 章で EJB 層を作成し、作業を進めながら IDE のテストアプリケーション機能を使用して各エンタープライズ Bean をテストします。次に、トラフィックを管理するセッション Bean を作成します。この作業は、Web サービスとクライアントを手動で作成する際の一般的なモデルです。

第 4 章では、Web サービスを作成し、どの EJB 層のビジネスメソッドを参照するか指定します。また、Web サービスを配備して、そのときに生成されるクライアントプロキシをテストします。

第 5 章では、提供されている 2 つの Swing クラスをアプリケーションにインストールおよび実行し、アプリケーションをテストします。

EJB コンポーネントの作成

第 3 章では、Forte for Java 4 のさまざまな機能を使用し、以下の処理を行う方法について学習します。

- EJB ビルダーを使用してエンティティ Bean とセッション Bean を簡単に作成する
- データベーススキーマから、取得メソッドや設定メソッドを使用してクラスを生成する
- テストアプリケーション機能を使用し、エンタープライズ Bean から J2EE テストアプリケーションをアセンブルする
- J2EE アプリケーションに EJB 参照を追加する
- J2EE リファレンス実装アプリケーションサーバーにテストアプリケーションを配備する
- テストアプリケーション機能によって作成されたテストクライアントページから、エンタープライズ Bean メソッドを実行する

EJB ビルダーの使用方法

EJB ビルダーウィザードは、エンタープライズ Bean を構成するさまざまなコンポーネントを自動的に作成します。ここで自動生成されるセッション Bean は、ステートレス Bean でもステートフル Bean でもよく、また、エンティティ Bean はコンテナ管理

の持続性 (CMP) をもつ Bean でも Bean 管理の持続性 (BMP) の特徴を持つ Bean でもかまいません。第 3 章 では、既存のデータベース表に基づいて 2 つの CMP エンティティ Bean を作成します。

エンティティ Bean を作成しながら、作成プロセス中にデータベースにどのように接続するか、また、フィールドが表の列を表すエンティティ Bean をどのように生成するかについて学習します。Bean の基本的な部分は、Java コードを使用して Forte for Java 4 のエクスプローラに生成されます。この Java コードは、ホームインタフェース、リモートインタフェース、Bean クラス、(適切な場合は) 主キークラスに対し、すでに生成されているものです。また、Bean を全体的に表す論理的な Bean ノードを使用して Bean プロパティを編集、変更する方法を学習します。EJB ビルダーの GUI 機能を使用し、作成メソッド、検索メソッド、ビジネスメソッドを追加する方法についても学習します。

詳細クラスの作成

詳細クラスには、エンティティ Bean と同じフィールドが存在する必要があります。2 つのクラスを作成し、それらクラスに適切な Bean プロパティを追加します。各プロパティの追加では、そのプロパティに対する補助メソッドを自動的に生成するオプションを有効にします。これは、アプリケーションが要求する取得メソッドと設定メソッドを取得するためです。続いて、プロパティをインスタンス化する各クラスのコンストラクタをコーディングします。最後に、2 つのエンティティ Bean のそれぞれに、対応する詳細クラスのインスタンスを返すためのコードを追加します。

テストアプリケーション機能

Forte for Java 4 IDE には、テスト用のクライアントを作成せずに、エンタープライズ JavaBean コンポーネントをテストする機能が用意されています。この機能では、J2EE リファレンス実装をアプリケーションサーバーとして使用し、エンタープライズ Bean を、Web モジュールとクライアント JSP ページを含んだ J2EE アプリケーションの一部として配備します。これらの JSP ページは 1 枚の HTML ページにまとめられているため、Web ブラウザから Bean の 1 つのインスタンスを作成し、そのビジネスメソッドを実行することができます。

3 つの エンタープライズ Bean に対しては、それぞれ個別のテストアプリケーションを作成します。エンティティ Bean に対しては、テストアプリケーションは J2EE アプリケーションを生成します。この J2EE アプリケーションには Web モジュールが含まれており、クライアントが Web ブラウザから使用するために自動的に生成された JSP ページと、エンティティ Bean に対する EJB モジュールで構成されています。セッ

ション Bean の EJB モジュールには、エンティティ Bean のメソッドをコールするために、エンティティ Bean の EJB モジュールも含まれています。IDE でコマンドを使用し、セッション Bean の EJB モジュールに対してエンティティ Bean の参照を追加します。テストアプリケーションの作成中に作成された EJB モジュールは、後で Web サービスによって参照されます。

Web ブラウザでセッション Bean をテストすると、アプリケーションのすべてのビジネスメソッドを実行することができます。第 3 章 の最後には、独自の Web サービスとクライアントを手動で作成する場合に、テストクライアントアプリケーションを使用するためのガイドラインが記載されています。

チュートリアルアプリケーションの Web サービスの作成

第 4 章では、Forte for Java 4 の機能を使用して以下のことを行う方法を学びます。

- 論理 Web サービスを作成する
- Web サービスで参照するセッションビジネスメソッドを指定する
- Web サービスを入れる J2EE アプリケーションを作成する
- Web サービスの実行時クラスとクライアントページを生成する
- Web サービスのクライアントプロキシを生成する

Web サービスの作成

Web サービスは、そのサービスに含まれるオブジェクトセット全体を表す論理的なエンティティで、Web サービスの変更と構成を容易に行えるようにします。Web サービスは、新規ウィザードを使用して名前とパッケージの場所を定義することによってエクスプローラに作成します。このウィザードの実行中に、Web サービスに参照させるビジネスメソッドの指定が求められます。

Web サービスのプロパティの 1 つとして Apache SOAP 実行環境の場所を示す URL を指定し、Web サービスの実行時クラス、つまり Web サービスを実装する EJB コンポーネントを生成します。

テストクライアントの作成

フロントエンドクライアントとバックエンド J2EE アプリケーションからなるテストクライアントを作成して、セッション Bean の EJB モジュールと Web サービスへの参照を追加します。この参照の追加によって、Web サービスの WAR および EJB JAR ファイルが利用可能になり、それらのプロパティをカスタマイズすることができます。カスタマイズするプロパティとしては、Web コンテキストプロパティがあります。これで DiningGuide の J2EE アプリケーションが完成し、配備準備完了です。

Web サービスの配備とテストクライアントの作成

Web サービスを含む J2EE アプリケーションを配備すると、IDE によってクライアントプロキシとサポートファイルが自動的に生成されます。サポートファイルとしては、参照される各メソッドの JSP ページ、 JSP エラーページ、開始ページなどです。

Web サービスのテスト

IDE のコマンドを使用して、DiningGuide アプリケーションを配備します。アプリケーションサーバーが起動し、1 つのページにすべての操作をまとめた、テストクライアントの開始ページを表示します。生成される JSP ページには、入力パラメータが必要なときの入力フィールドと、操作を実行するための「Invoke」ボタンが含まれます。これらの手段を利用して、Web サービスがどのようにしてセッション Bean の各メソッドを呼び出すのかを確認します。

他の開発者が Web サービスを利用できるようにする

このチュートリアルでは、UDDI レジストリに Web サービスを公開する方法は説明しませんが、テスト目的で他の開発者が Web サービスを利用できるようにする非公式な方法について説明します。この方法を使うと、WSDL ファイルを生成してサーバーに置くか、電子メールなどの他の方法で配布することによって、他の開発者が利用できるようにします。他の開発者はこのファイルからクライアントプロキシを生成し、Web サービスで利用できるメソッドを特定し、それに従ってクライアントを作成できます。また、配備した Web サービスの URL を教えることによって、他の開発者が Web サービスに対してクライアントをテストすることもできます。

Forte for Java 4 IDE にはまた、テストに利用できるシングルユーザー用内部 UDDI レジストリも用意されています。 StockApp のサンプルはこのデバイスを使用して Web サービスを公開する具体例で、Forte for Java Developer のポータルサイトの Examples and Tutorials ページから入手できます。Examples and Tutorials ページの URL は以下のとおりです。

```
http://forte.sun.com/ffj/documentation/tutorialsandexamples.html
```

UDDI レジストリへの Web サービスの公開についての詳細は、Forte for Java プログラミングシリーズの『Web サービスのプログラミング』をご覧ください。

提供クライアントのインストールと利用

このチュートリアルの付録 A には、DiningGuide アプリケーションの働きを具体的に理解するための、単純な Swing クライアントのコードが記載されています。このクライアントは、データベース表ごとに 1 つの Swing クラスで構成されます。2 つのクラスを作成して、デフォルトのコードを提供されるコードに置き換え、主クラスを実行するだけです。

提供コードを調べることによって、クライアントがアプリケーションのメソッドにアクセスする方法が分かります。最初に、クライアントはクライアントプロキシをインスタンス化する必要があります。このインスタンス化によって、クライアントがクライアントプロキシのメソッドを利用できるようになります。それらのメソッド (図 2-1 を参照) は、アプリケーションの EJB 層のメソッドにアクセスする際に SOAP 実行環境が使用します。

最後に

このチュートリアルアプリケーションは、Forte for Java 4, Enterprise Edition の主要機能を具体的に紹介するための実行アプリケーションとして、簡潔で比較的短期間 (1 日程度) で作成できるように設計されています。このため、次のような制約があります。

- エラー処理がない
- デバッグプロシージャがない
- Web サービスの公開の説明がない

すぐに完成できるように単純なアプリケーションとして設計されているとはいえ、このチュートリアルアプリケーション全体のインポートやそのソースファイルの表示、作成するメソッドへのメソッドコードのコピーが行えると便利です。DiningGuide アプリケーションは、以下の URL の Forte for Java 4 Developer のポータルサイトの Examples and Tutorials ページから入手できます。

```
http://forte.sun.com/ffj/documentation/tutorialsandexamples.html
```

第3章

DiningGuide アプリケーションの EJB 層の作成

この章では、DiningGuide チュートリアルアプリケーションの EJB 層の作成方法を手順に従って説明します。その過程で EJB ビルダーを使用してエンティティ Bean とセッション Bean の両方を作成し、IDE のテスト機能を使用してそれら Bean をテストします。この章の内容は次のとおりです。

この章の作業を完了すると、1 つのテストアプリケーションとして配備した DiningGuide アプリケーションの EJB 層全体を実行できるようになります。

EJB 層を完成すると、独自の Web サービスやクライアントページを自由に作成できるようになります。また、そのまま第 4 章に進み、Forte for Java 4 の Web サービス機能を使用してアプリケーションの Web サービスを作成する方法を学ぶこともできます。

EJB 層の概要

この章では、チュートリアルアプリケーションの中枢モジュールである EJB 層を作成します。各コンポーネントを作成するたびに、IDE のテストアプリケーション機能を使用してテストを行います。このテストアプリケーションでは、テスト用の Web サービスとクライアントを自動的に作成します。

作成する EJB 層には、次の Bean が定義されます。

- `Restaurant` エンティティ Bean
- `Customerreview` エンティティ Bean
- `DiningGuideManager` セッション Bean
- `RestaurantDetail` Bean
- `CustomerreviewDetail` Bean

J2EE アーキテクチャにおける EJB 層の役割についての詳細は、Forte for Java 4 プログラミングシリーズの『Enterprise JavaBeans コンポーネントのプログラミング』を参照してください。このマニュアルでは、あらゆる Bean 要素を完全解説しているとともに、エンタープライズ Bean においてトランザクション、持続性、セキュリティがどのようにサポートされているかについても説明しています。

このチュートリアルの例と同様に EJB 層とそこから生成される Web サービスを使用するアプリケーションがどのようなものであるかを学びたい場合は、Forte for Java 4 Example ページ (`http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`) にある PartSupplier というサンプルアプリケーションをご覧ください。

エンティティ Bean

エンティティ Bean には、概念を定義する共有データセットに対し、一貫したインタフェースが用意されています。このチュートリアルでは、2 つの概念 (レストラン、利用者のコメント) を使用します。ここで作成する `Restaurant` および `Customerreview` のエンティティ Bean は、第 1 章で作成したデータベース表を表します。

これらのエンティティ Bean には、コンテナ管理の持続性 (CMP) または Bean 管理の持続性 (BMP) のいずれかを定義することができます。BMP エンティティ Bean では、開発者は、Bean フィールドをデータベース表の列にマップするためのコードを記述す

る必要があります。CMP エンティティ Bean では、EJB の実行環境で持続性の操作を管理します。このチュートリアルでは、CMP エンティティ Bean を使用します。IDE の EJB ビルダーウィザードを使用し、データベースに接続してマップする列を指定します。ウィザードでは、データベースにマップされるエンティティ Bean を作成します。

EJB ビルダーでは、CMP エンティティ Bean のフレームワークを作成します。具体的には、エンティティ Bean を体系付けてカスタマイズを容易にするための論理ノード、必要なホームインタフェース、リモートインタフェース、Bean クラスを作成します。

このエンティティ Bean の作成、検索、ビジネスメソッドは手動で定義します。メソッドを定義すると、IDE によって適切な Bean コンポーネントにそのメソッドが自動的に伝達されます。たとえば **生成**メソッドは Bean のホームインタフェースに、また、対応する `ejbCreate` メソッドは Bean のクラスに伝達されます。メソッドを編集すると、その変更もまた伝達されます。

検索メソッドでは、目的のオブジェクトを探すための適切なデータベース文を定義する必要があります。EJB 2.0 アーキテクチャには、EJB QL という、データベースに依存しないバージョンの SQL が定義されており、データベースの定義文には、この SQL を使用します。配備するとき、J2EE RI プラグインは EJB QL を実際のデータベースに合った SQL に変換して配備記述子に書き込みます。

セッション Bean

エンティティ Bean が共有データを表すのに対し、セッション Bean は、共有されずに複数の概念にまたがっているデータにアクセスします。この種の Bean は、特定の処理を行うために必要なステップを管理することもできます。セッション Bean には、ステートフルとステートレスの 2 つのタイプがあります。「ステートフル」セッション Bean は、クライアントとの会話状態を維持しながら、そのクライアントのためだけに仕事をします。これに対して「ステートレス」セッション Bean は、会話状態を維持することはなく1 つのクライアント専用にはなりません。スレートレス Bean は、あるクライアントのためのメソッド呼び出しを終了すると、別のクライアントからの要求にサービスを提供することができます。

DiningGuide アプリケーションにおけるクライアント要求としては、データベース内のすべてのレストランのデータの取得、特定のレストランに関するすべての利用者コメントの検出などが考えられます。また、特定のレストランに関するコメントを送信するのもクライアント要求です。これらの要求の間に関連性はなく、会話状態を維持

する必要はありません。このため、DiningGuide アプリケーションはステートレスセッション Bean を使用して、それぞれの要求に必要なさまざまなステップを管理します。

セッション Bean は、クライアントの要求を満たすためにレストランと利用者コメントレコードのコレクションを繰り返し作成します。この処理は、フィールドごとの取得メソッドと設定メソッドをエンティティ Bean に追加することによっても実現できますが、この方法では、セッション Bean が表の行を取り出す必要があるたびに個々のフィールドについてメソッド呼び出しが必要になります。メソッドの呼び出し回数を減らすため、チュートリアルアプリケーションでは、詳細クラスと呼ばれる特殊なヘルパークラスを使用して、行のデータを保持します。

詳細クラス

詳細クラスは対応するエンティティ Bean と同じフィールドを持つとともに、各フィールド用の取得メソッドと設定メソッドを持ちます。エンティティ Bean をルックアップする際、セッション Bean は対応する詳細クラスを使用することによって、エンティティ Bean が返す各リモート参照のインスタンスを作成します。セッション Bean は、表示する行データをインスタンス化する際、単に詳細クラスのコンストラクタを呼び出すだけです。こうしてセッション Bean が行インスタンスのコレクションを作成すると、次の段階としてクライアントが表示する HTML ページの形式に変換することが可能なります。

図 3-1 は、詳細クラスの仕組みの概念図です。

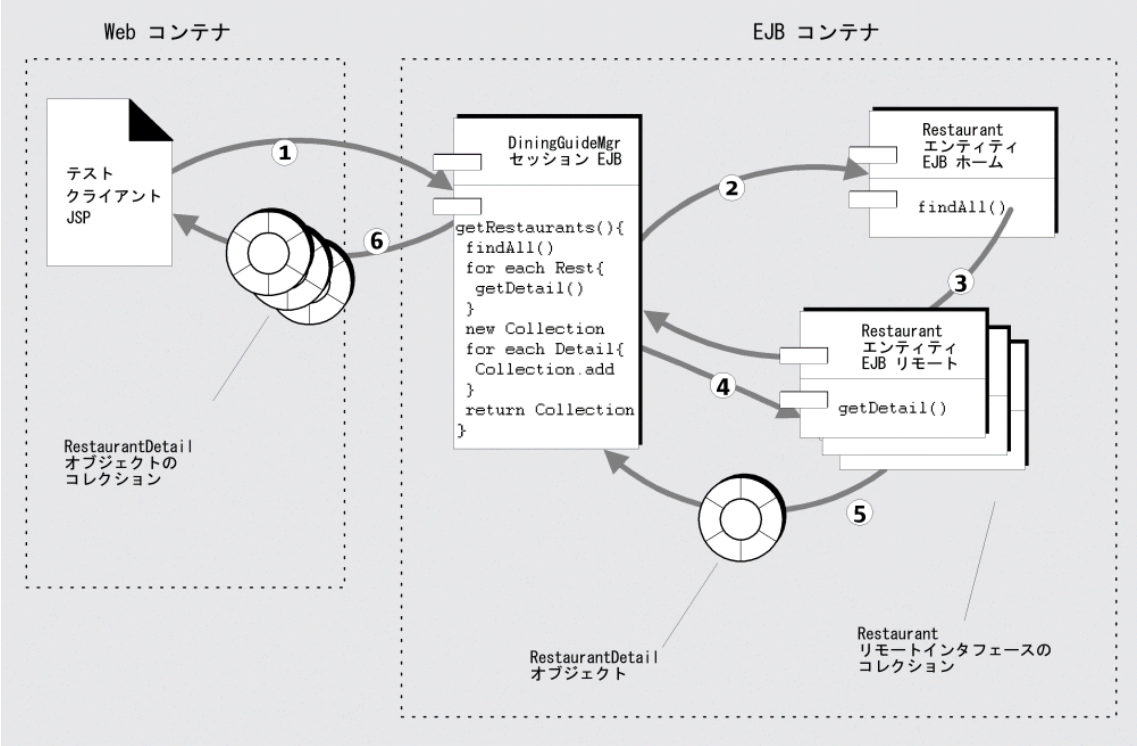


図 3-1　詳細クラスの働き

図 3-1 中の番号が振られた矢印はそれぞれ、以下のアクションを意味します。

1. Web コンテナが、クライアントからの全レストランデータの要求を `DiningGuideManager` セッション Bean に渡します。
2. セッション Bean が、`Restaurant` エンティティ Bean の `findAll` メソッドを呼び出して、`Restaurant` エンティティ Bean をルックアップします。
3. `findAll` メソッドが、エンティティ Bean に対する使用可能なすべてのリモート参照を取得します。
4. 返されたリモート参照ごとに、セッション Bean が `Restaurant` Bean の `getRestaurantDetail` メソッドを呼び出して、`RestaurantDetail` クラスをフェッチします。

5. `getRestaurantDetail` が `RestaurantDetail` オブジェクトを返し、それがコレクションに追加されます。
6. セッション Bean がすべての `RestaurantDetail` オブジェクトのコレクションを Web コンテナに返し、これがクライアント表示に適切な形式にデータを変換します。

手順の概要

EJB 層を作成するには、次の 6 つの作業を行う必要があります。

1. エンティティ Bean の作成

   まず、EJB ビルダーウィザードを使用して CMP エンティティ Bean のスケルトンを作成し、続いてテスト用の作成・検索メソッド、単純なビジネスメソッドを追加します。

2. エンティティ Bean と同じフィールドを持つ詳細クラスの作成

   通常の JavaBeans である `Restaurant` および `Customerreview` クラス、および各フィールドの取得メソッドと設定メソッドを作成します。

3. エンティティ Bean に対する詳細クラスをフェッチするビジネスメソッドの作成
4. IDE のテストアプリケーション機能によるエンティティ Bean のメソッドのテスト

   自動的に生成されたテストクライアントを Web ブラウザで表示し、生成メソッドを実行することでBean のインスタンスを作成して、そのビジネスメソッドを使用可能にします。ビジネスメソッドを作成したら、そのビジネスメソッドを実行します。

5. セッション Bean の作成

   EJB ビルダーを使用してステートレスセッション Bean のスケルトンを作成し、その Bean の生成メソッドを変更して、エンティティ Bean ルックアップを行えるようにします。また、詳細クラスから詳細オブジェクトのコレクションを作成する取得メソッドをエンティティ Bean ごとに作成し、データベースに利用者コメントレコードを作成するメソッドを作成します。SOAP 実行環境が必要とするダミーのビジネスメソッドも 2 つ作成します。

6. テストアプリケーション機能によるセッション Bean のテスト

   EJB モジュールのプロパティシートで CMP エンティティ Bean に対する参照を追加します。テストアプリケーションを作成して、EJB モジュールをテストアプリケーションの EJB モジュールに追加します。最後にテストクライアントを使用することによって、セッション Bean のインスタンスを作成し、そのメソッドを実行します。

注 - チュートリアル用アプリケーションの作成を開始するには、第 1 章で説明した設定手順をすべて完了しておく必要があります。

EJB ビルダーによる エンティティ Bean の作成

ここでは、第 1 章で作成した 2 つのデータベース表を表す 2 つのエンティティ Bean、`Restaurant` と `Customerreview` を作成します。

EJB アーキテクチャのバージョン 2.0 では、エンティティ Bean はローカルインタフェースかリモートインタフェース、あるいはその両方を持つことができます。どれを使用するかの基準は、Bean のメソッドを呼び出すクライアントがエンティティ Bean に対して、リモートにあるかローカルにあるかに依存します。このチュートリアルでは、Web サービスから エンティティ Bean のメソッドへのアクセス方法に柔軟性を持たせるため、リモートとローカル両方のインタフェースを持つ Bean を作成します。セッション Bean がエンティティ Bean のメソッドにアクセスするケース (ローカルインタフェースを使用) と、 Web サービスがそれらメソッドに直接アクセスするケース (リモートインタフェースを使用) の 2 つケースが考えられます。

参照 - EJB ビルダーの操作についての詳細は、Forte for Java 4 ヘルプのEJB コンポーネントに関する項目を参照してください。

注 - 付録 A に、完成したエンティティ Bean のソースコードがあります。

Restaurant エンティティ Bean の作成

最初に、アプリケーションの保存先のディレクトリをファイルシステムとしてマウントします。その後で EJB 層のパッケージを作成し、パッケージ内にエンティティ Bean を作成します。

注 - この手順は、Forte for Java 4 IDE と PointBase サーバーの両方が動作していることを前提としています (9 ページの「チュートリアルデータベース表の作成」の手順 1 を参照)。

Restaurant エンティティ Bean を作成するには、次の操作を行います。

1. ファイルシステムの適当な場所に、DiningGuide という名前でディレクトリを作成します。
2. Forte for Java 4 IDE から「ファイル」->「ファイルシステムをマウント」を選択します。

   新規ウィザードが表示されます。
3. 「ローカルディレクトリ」を選択して「次へ」をクリックします。

   新規ウィザードに「ディレクトリを選択」区画が表示されます。
4. ファイル検索機能を使用して DiningGuide ディレクトリを見つけ、そのディレクトリを選択して、「完了」をクリックします。

   ディレクトリ （たとえば c:¥DiningGuide) がマウントされて、エクスプローラに表示されます。
5. マウントしたファイルシステムを右クリックして、「新規」->「Java パッケージ」を選択します。

   このパッケージに、アプリケーションの EJB 層を格納します。
6. 新規パッケージに Data という名前を付けて、「完了」をクリックします。

   DiningGuide ディレクトリに Data パッケージが表示されます。

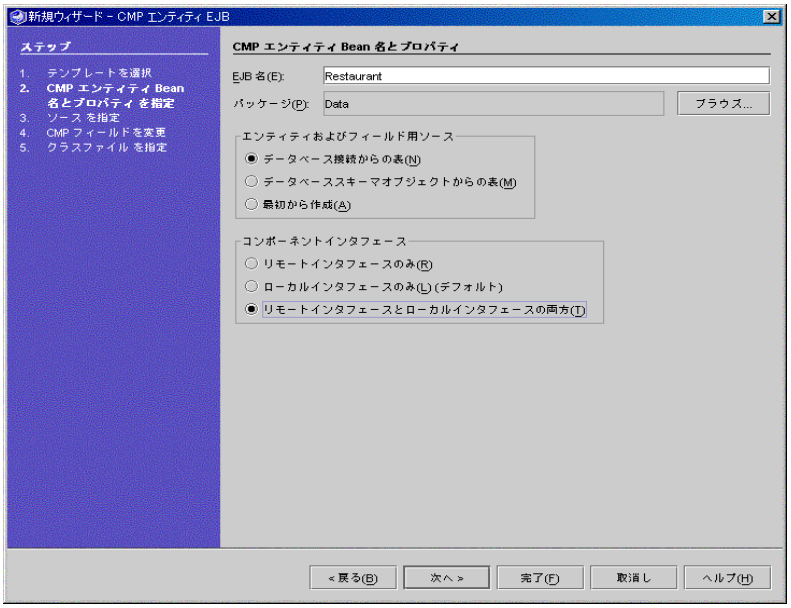
7. 「Data」パッケージを右クリックして、「新規」->「J2EE」->「CMP エンティティ EJB」選択します。

   新規ウィザードの「CMP エンティティ Bean 名とプロパティ」区画 が表示されます (この設定は EJB ビルダーモジュールによって使用される)。ウィザードの区画で「ヘルプ」ボタンをクリックすると、CMP エンティティ Bean の作成に関する、状況に応じたヘルプを見ることができます。

8. CMP Bean に `Restaurant` という名前を付けて、次のオプションを選択します。

   - エンティティおよびフィールド用のソース：「データベース接続からの表」
   - コンポーネントインタフェース：「リモートインタフェースとローカルインタフェースの両方」

   新規ウィザードが次のような表示になります。



9. 「次へ」をクリックします。

   「データベース接続からの表」区画が表示されます。9 ページの「チュートリアル データベース表の作成」で作成したデータベースとの接続が `jdbc:pointbase:server://localhost:9092/sample [pbpublic on PBPUBLIC]` というラベルで、四角形を 2 つに割ったようなアイコン ( ) で表示されます。

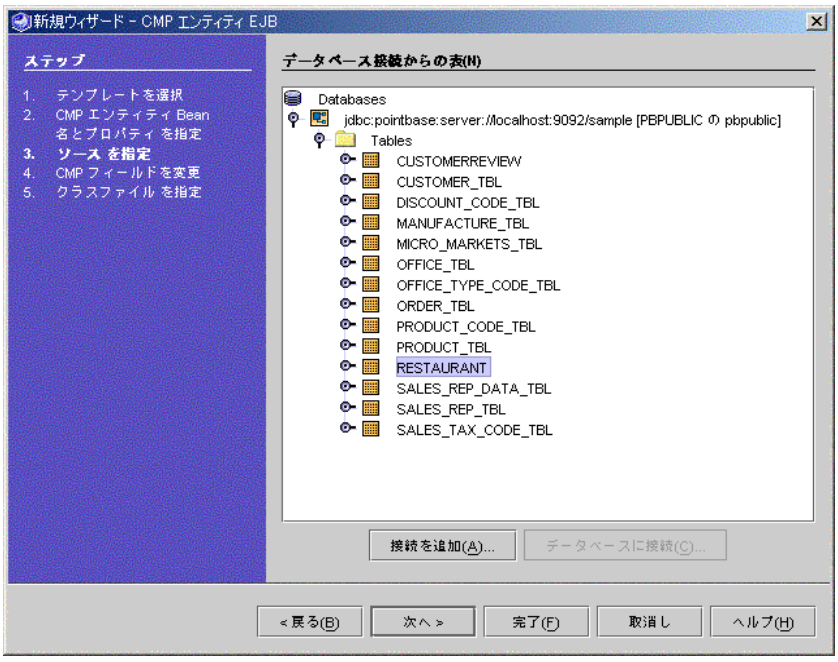
10. そのアイコンを選択して、「データベースに接続」ボタンをクリックします。

    2 つに割れていた四角形が完全な四角形 ( ) になり、接続が存在することを示します。

注 - このようにならない場合は、たいていPointBase Network Server が起動していないことが原因です。その場合は、「ツール」->「PointBase ネットワークサーバー」->「サーバーを起動」を選択した後「接続を追加」ボタンをクリックして、接続を追加します。

---

11. 「データベース接続」ノードを開いて「Tables」ノードを開き、「RESTAURANT」表を選択します。



12. 「次へ」をクリックします。

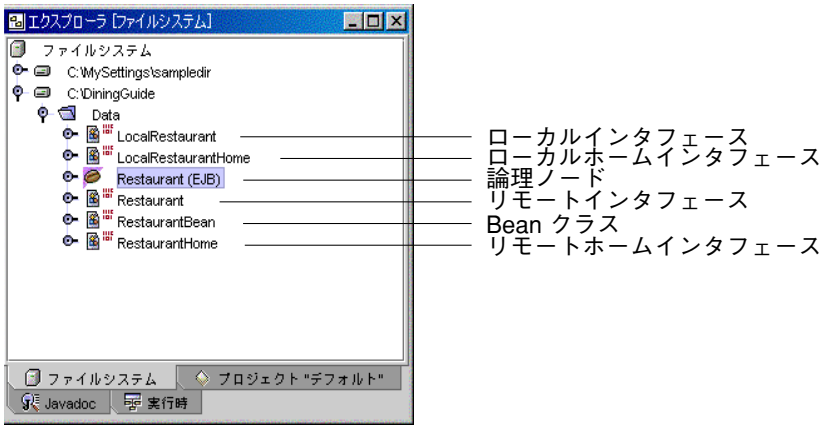
    「CMP フィールド」区画が表示されます。Restaurant データベース表の列の横に、ウィザードによって Restaurant エンティティ Bean が作成されたときに対応づけられる Java フィールドが表示されます。

13. デフォルトのすべてのラベルをそのまま受け入れて、「次へ」をクリックします。

    「CMP エンティティ Bean クラスファイル」区画が表示され、作成する `Retaurant` Bean のすべての部品が示されます。EJB ビルダーウィザードによって、自動的にデータベース表と同じ名前がこのエンティティ Bean に付けられていることに注目してください。

14. 「完了」をクリックします。

    `Restaurant` エンティティ Bean とその全部品が作成され、「エクスプローラ」ウィンドウに表示されます。



    部品のうちの 5 つはインタフェース、1 つは Bean クラスです。6 番目の部品の論理ノードは、エンタープライズ Bean のすべての要素を 1 つのグループにまとめて、簡単に扱えるようにします。

15. 「ファイル」->「保存」を選択して、作業内容を保存します。

`Customerreview` エンティティ Bean の作成

ここでは、`Restaurant` Bean を作成したのと同様の手順で `Customerreview` エンティティ Bean を作成します。

1. 「`Data`」パッケージを右クリックして、「新規」->「J2EE」->「CMP エンティティ EJB」選択します。

2. CMP Bean に `Customerreview` という名前を付けて、次のオプションを選択します。

    - エンティティおよびフィールド用のソース：「データベース接続からの表」

- コンポーネントインタフェース：「リモートインタフェースとローカルインタフェースの両方」

3. 「次へ」をクリックします。

   「データベース接続からの表」区画が表示されます。

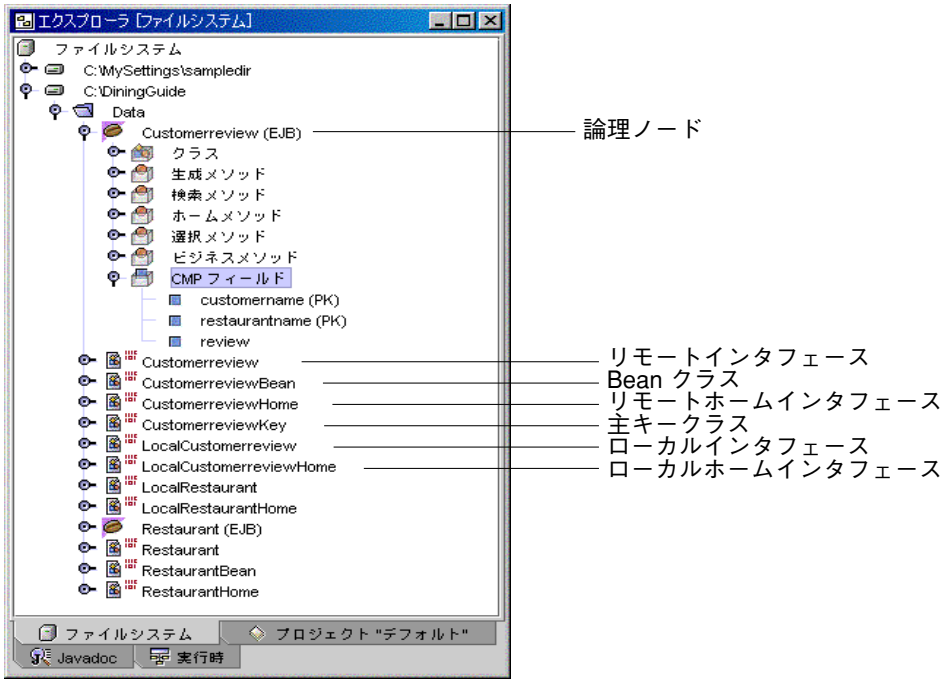
   サンプルデータベースが接続されたままの場合は、手順 4 に進みます。接続されているかどうかは、四角形のアイコン ( ) になっているかどうかで判ります。

   四角形を 2 つに割ったようなアイコン ( ) が表示されている場合は、そのアイコンを選択して、「データベースに接続」ボタンを選択します。

4. 「データベース」ノードを開いて、「Tables」フォルダを開き、CUSTOMERREVIEW 表を選択して、「次へ」をクリックします。

5. 「CMP フィールド」区画で「次へ」をクリックし、最後の区画 (「CMP エンティティ Bean クラスファイル」区画) で「完了」をクリックします。

   エクスプローラの Data パッケージに Customerreview エンティティ Bean が表示されます。CustomerreviewKey Bean という追加のコンポーネントがあることに注目してください。これは、エンティティ Bean に複合主キーがある場合に自動的に作成される Bean です (第 1 章の表 1-4 で複合主キーを確認してください)。



6. 「ファイル」->「すべてを保存」を選択して、作業内容を保存します。

CMP エンティティ Bean の生成メソッドの作成

ここでは、両方の エンティティ Bean の生成メソッドを作成して、パラメータと、それら Bean のインスタンスのフィールドを初期化するコードを追加します。

Restaurant Bean の生成メソッドの作成

以下の手順で、Restaurant エンティティ Bean の生成メソッドを作成します。

1. エクスプローラで「Restaurant（EJB）」論理ノード (Bean のアイコン ) を右クリックします。
2. コンテキストメニューから「生成メソッドを追加」を選択します。

   「新規生成メソッドを追加」ダイアログが表示されます。
3. 「追加」ボタンを使用して、Restaurant 表の各列に 1 つ、合計で 7 つのパラメータを作成します。

```
restaurantname (java.lang.String)
cuisine (java.lang.String)
neighborhood (java.lang.String)
address (java.lang.String)
phone (java.lang.String)
description (java.lang.String)
rating (java.lang.Integer)
```

注 - これらのパラメータを作成する順序は、テストアプリケーション機能を使用して Bean をテストするときに重要な意味を持ちます。このため、上記に列挙している順序でパラメータを作成してください。

デフォルトで作成される 2 つの例外はそのまま残します。ホームとローカルホームの両方のインタフェースにメソッドが追加されることを確認します。

4. 「了解」をクリックします。

   RestaurantHome と LocalRestaurantHome インタフェースのそれぞれに 生成メソッドが 1 つずつ伝達され、Restaurant Bean クラス (ReastaurantBean) に ejbCreate メソッドが伝達されます。この Bean クラスには、関連する ejbPostCreate メソッドも追加されます。

5. 「Restaurant(EJB)」論理ノードを開いて、「生成メソッド」フォルダを開き、生成メソッドをダブルクリックします。

   Bean の ejbCreate メソッドにカーソルが置かれた状態で、ソースエディタが表示されます。

注 - 「生成メソッド」ノードを右クリックして、「ヘルプ」を選択すると、生成メソッドに関するオンラインヘルプを見ることができます。

---

6. Bean インスタンスのフィールドを初期化するコード (太字部分) を ejbCreate メソッドの本体に追加します。

```
public String ejbCreate(java.lang.String restaurantname,
java.lang.String cuisine, java.lang.String neighborhood,
java.lang.String address, java.lang.String phone,
java.lang.String description, java.lang.Integer rating) throws
javax.ejb.CreateException {
    if (restaurantname == null) {
// ソースエディタでは、次の 2 行を 1 行にする
        throw new javax.ejb.CreateException("The restaurant name
is required.");
    }
    setRestaurantname(restaurantname);
    setCuisine(cuisine);
    setNeighborhood(neighborhood);
    setAddress(address);
    setPhone(phone);
    setDescription(description);
    setRating(rating);

    return null;
}
```

参照 - 入力またはコピー操作によってソースエディタにコードを入力した後、そのコードブロックを選択して Control+Shift F を押すと、書式を整えることができます。

Restaurant エンティティ Bean の 生成メソッドが呼び出されると、この Bean のコンテナ管理フィールドに基づいて、データベースに新しいレコードが作成されます。

Customerreview Bean の生成メソッドの作成

以下の手順で、Customerreview エンティティ Bean の生成メソッドを作成します。

1. 「Customerreview(EJB) 」論理ノード (Bean のアイコン ) を右クリックして、「生成メソッドを追加」を選択します。
2. 「追加」ボタンを使用して、CustomerReview 表の各列に 1 つ、合計で 3 つのパラメータを作成します。

```
restaurantname (java.lang.String)
customername (java.lang.String)
review (java.lang.String)
```

注 - 上記の手順 3 のとき同様、これらのパラメータは、列挙されている順序で作成してください。

デフォルトで作成される 2 つの例外はそのまま残します。ホームとローカルホームの両方のインタフェースにメソッドが追加されることを確認します。

3. 「了解」をクリックします。
4. 「Customerreview(EJB)」論理ノードを開いて、「生成メソッド」フォルダを開き、「生成メソッド」をダブルクリックします。

Bean の ejbCreate メソッドにカーソルが置かれた状態で、ソースエディタが表示されます。

5. Bean インスタンスのフィールドを初期化するコード (太字部分) を ejbCreate method メソッドの本体に追加します。

```
public CustomerreviewKey ejbCreate(java.lang.String
restaurantname, java.lang.String customername, java.lang.String
review) throws javax.ejb.CreateException {
    if ((restaurantname == null) || (customername == null)) {
// ソースエディタでは、次の 2 行を 1 行にする
        throw new javax.ejb.CreateException("Both the restaurant
name and customer name are required.");
    }
    setRestaurantname(restaurantname);
    setCustomername(customername);
    setReview(review);

    return null;
}
```

ejbCreate メソッドが呼び出されると、この Bean のコンテナ管理フィールドに基づいて、データベースに新しいレコードが作成されます。

6. 「ファイル」->「すべてを保存」を選択して、作業内容を保存します。

コンテキストから、すべてまたは選択されたインタンスを特定する検索メソッドを両方のエンティティ Bean に作成してください。

エンティティ Bean に対する検索メソッドの作成

ここでは、すべてのレストランデータを検出する findAll メソッドを Restaurant Bean に作成します。また、特定のレストランのコメントデータを検出する findByRestaurantName メソッドを Customerreview Bean に作成します。

findByPrimaryKey 以外の各検索メソッドは、配備記述子の照会要素に関連づける必要があります。2 つのエンティティ Bean の検索メソッドの作成では、 SQL 文には、データベースに依存しない EJB 2.0 仕様の言語、すなわち EJB QL を指定します。配備の際に、RI プラグインは、EJB QL をターゲットのデータベースの SQL に変換します。

Restaurant Bean の findAll メソッドの作成

Restaurant Bean の findAll メソッドを作成するには、次の操作を行います。

1. 「Restaurant （EJB）」論理ノードを右クリックして、「検索メソッドを追加」を選択します。

   「新規検索メソッドを追加」ダイアログが表示されます。

2. 「名前」フィールドに findAll と入力します。

3. 戻り値の型として「java.util.Collection」を選択します。

4. デフォルトの 2 つの例外をそのまま受け入れます。

5. 以下のように EJB QL 文を定義します。

EJB QL 文	テキスト
Select	Object(o)
From	Restaurant o

6. ホームとローカルホームの両方のインタフェースにメソッドが追加されることを確認します。

7. 「了解」をクリックします。

   Restaurant Bean のローカルおよびローカルホームインタフェースに findAll メソッドが作成されます。

注 - 「検索メソッド」ノードを右クリックして、「ヘルプ」を選択すると、検索メソッドに関するオンラインヘルプを見ることができます。

Customerreview Bean の findByRestaurantName メソッドの作成

Customerreview Bean の findByRestaurantName メソッドを作成するには、次の操作を行います。

1. 「Customerreview （EJB）」論理ノードを右クリックして、「検索メソッドを追加」を選択します。

   「新規検索メソッドを追加」ダイアログが表示されます。

2. 「名前」フィールドに findByRestaurantName と入力します。

3. 戻り値の型として「java.util.Collection」を選択します。
4. パラメータの「追加」ボタンをクリックします。

   「メソッドのパラメータを入力」ダイアログが表示されます。
5. フィールド名として restaurantname と入力します。
6. 型として「java.lang.String」を選択します。
7. 「了解」をクリックします。
8. デフォルトの 2 つの例外をそのまま受け入れます。
9. 以下のように EJB QL 文を定義します。

EJB QL 文	テキスト
Select	Object(o)
From	Customerreview o
Where	o.restaurantname = ?1

   使用する数値は、検索メソッド内のパラメータ位置によって異なります。この場合、パラメータは 1 つだけですから、数値は 1 になります。
10. ホームとローカルホームの両方のインタフェースにメソッドが追加されることを確認します。
11. 「了解」をクリックします。

    Customerreview Bean のローカルおよびローカルホームインタフェースに findByRestaurantName メソッドが作成されます。
12. 「ファイル」->「すべてを保存」を選択して、作業内容を保存します。

テスト用のビジネスメソッドの作成

ここでは、エンティティ Bean ごとに、そのパラメータの値を返すビジネスメソッドを作成します。ビジネスメソッドを作成することによって、後で Bean をテストすることができます。Restaurant に対しては getRating メソッドを、また Customerreview に対しては getReview メソッドを作成します。

Restaurant Bean の getRating メソッドの作成

Restaurant Bean に対する getRating ビジネスメソッドを作成するには、次の操作を行います。

1. 「Restaurant(EJB)」論理ノードを開いて、「ビジネスメソッド」ノードを開きます。

   このエンティティ Bean には、まだビジネスメソッドはありません。

2. Restaurant Bean のクラス (RestaurantBean) を開いて、「メソッド」ノードを開きます。

   Bean の各フィールドに、getRating メソッドなどの補助メソッドが用意されています。

   これらのメソッドは、データソースとの同期の際にコンテナによって使用されます。開発でこれらのメソッドを使用するには、メソッドをビジネスメソッドとして作成する必要があります。

3. Restaurant （EJB）論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。

   「新規ビジネスメソッドを追加」ダイアログが表示されます。

4. 「名前」フィールドに getRating と入力します。

5. 「戻り値の型」フィールドに java.lang.Integer と入力します。

   デフォルトの例外 (RemoteException) とメソッドの作成先をそのまま受け入れます。この作成先のローカルおよびローカルホーム両方のインタフェースにメソッドが作成されます。

6. 「了解」をクリックします。

7. 「Restaurant(EJB)」論理ノードを開いて、その「ビジネスメソッド」フォルダを開きます。

   これで getRating メソッドをビジネスメソッドとして使用できます。getRating メソッドを使用すると、選択されたレストランレコードの rating 列の値が返されます。

8. 「Restaurant(EJB)」論理ノードを右クリックし、コンテキストメニューから「EJB の検査」を選択します。

   Restaurant エンティティ Bean がコンパイルされます。続いて、Customerreview Bean に対する同様のボタンを作成します。

Customerreview Bean の getReview メソッドの作成

Customerreview Bean に対する getReview ビジネスメソッドを作成するには、次の操作を行います。

1. 「Customerreview (EJB)」論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。

   「新規ビジネスメソッドを追加」ダイアログが表示されます。

2. 「名前」フィールドに getReview と入力します。

3. 「戻り値の型」フィールドに java.lang.String と入力します。

   デフォルトの例外 (RemoteException) とメソッドの作成先をそのまま受け入れます。この作成先のローカルおよびローカルホーム両方のインタフェースにメソッドが作成されます。

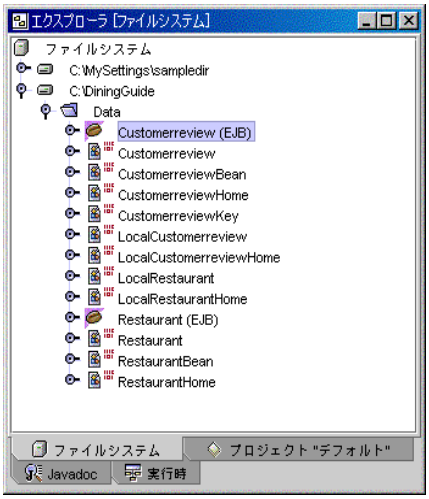
4. 「了解」をクリックします。

   これで getReview メソッドをビジネスメソッドとして使用できます。getReview メソッドを使用すると、選択されたレストランレコードの review 列の値が返されます。

5. 「Customerreview(EJB)」論理ノードを右クリックし、コンテキストメニューから「EJB の検査」を選択します。

   Customerreview エンティティ Bean がコンパイルされます。

6. エクスプローラから Bean が未コンパイルであることを示すアイコンがなくなったことを確認します。



エンティティ Bean データを表示する詳細クラスの作成

34 ページの「詳細クラス」で説明したように、このチュートリアルアプリケーションでは、詳細クラスを使用して表の行データを保持することによって、エンティティ Bean に対するメソッド呼び出しを表示したり、その回数を減らしたりします。それらの詳細クラスのフィールド、フィールドごとの補助メソッド、各フィールドを設定するコンストラクタは、対応する エンティティ Bean と同じである必要があります。

注 - 付録 A に、完成した詳細クラスのソースコードがあります。

詳細クラスの作成

最初に `RestaurantDetail` クラスと `CustomerreviewDetail` クラスを作成します。

1. エクスプローラで「Data」パッケージを右クリックして、「新規」->「Beans」->「Java Bean」を選択します。
2. この Bean に `RestaurantDetail` という名前を付けて、「完了」をクリックします。

   エクスプローラに新しい Bean が表示されます。
3. 手順 1 と 手順 2 を繰り返して `CustomerreviewDetail` Bean を作成します。

詳細クラスのプロパティとその補助メソッドの作成

詳細クラスを作成したら、対応する エンティティ Bean の CMP フィールドにあるのと同じ Bean プロパティをクラスに追加します。エンティティ Bean の Bean クラスの「Bean パターン」の内容を見ると、CMP フィールドがプロパティとして保存されていることが分かります。フィールドを追加しながら、各フィールド用の補助メソッドを自動的に作成することができます。

詳細クラスのプロパティとメソッドを作成するには、次の操作を行います。

1. 「RestaurantDetail」ノードを開いて、「クラス RestaurantDetail」ノードを開きます。
2. 「Bean パターン」ノードを右クリックして、「追加」->「プロパティ」を選択します。

   「新規プロパティパターン」ダイアログが表示されます。
3. 「名前」フィールドに restaurantname と入力します。
4. 「種類」として「String」を選択します。
5. 「フィールドを生成」オプションを選択します。
6. 「Return 文を生成」オプションを選択します。
7. 「Set 文を生成」オプションを選択します。
8. 「了解」をクリックします。
9. 手順 2 ～ 手順 8 を繰り返すことによって、以下のプロパティを作成します。

   ```
   cuisine (String)
   neighborhood (String)
   address (String)
   phone (String)
   description (String)
   rating (java.lang.Integer)
   ```
10. RestaurantDetail Bean の「メソッド」ノードを開きます。

    各フィールドに対する補助メソッドが生成されています。
11. 「CustomerreviewDetail」ノードから「クラス CustomerreviewDetail」ノードを開き、手順 2 から 手順 8 を繰り返して「Bean パターン」ノードに次のプロパティを作成します。

    ```
    restaurantname (String)
    customername (String)
    review (String)
    ```

詳細クラスのコンストラクタの作成

クラスのフィールドをインスタンス化する詳細クラスのコンストラクタを作成するには、次の操作を行います。

1. 「RestaurantDetail」Bean を開いて「クラス RestaurantDetail」ノードを右クリックし、「追加」->「コンストラクタ」を選択します。

   「新規コンストラクタを編集」ダイアログが表示されます。

2. 次のメソッドパラメータを追加して、「了解」をクリックします。

```
java.lang.String restaurantname
java.lang.String cuisine
java.lang.String neighborhood
java.lang.String address
java.lang.String phone
java.lang.String description
java.lang.Integer rating
```

   (java.lang.Integer は自分で入力する必要があります。型のリストから選択することはできません。)

3. フィールドを初期化するコード (コードの太字部分) を RestaurantDetail コンストラクタに追加します。

```
public RestaurantDetail(java.lang.String restaurantname,
java.lang.String cuisine, java.lang.String neighborhood,
java.lang.String address, java.lang.String phone,
java.lang.String description, java.lang.Integer rating){
    System.out.println("Creating new RestaurantDetail");
    setRestaurantname(restaurantname);
    setCuisine(cuisine);
    setNeighborhood(neighborhood);
    setAddress(address);
    setPhone(phone);
    setDescription(description);
    setRating(rating);
}
```

参照 - ソースエディタにコピーしたコードブロックを選択して、Control+Shift F を押すと、その部分の書式を整えることができます。

---

4. 同様に次のパラメータを使用して、CustomerreviewDetail クラスにコンストラクタを追加します。

```
java.lang.String restaurantname
java.lang.String customername
java.lang.String review
```

5. フィールドを初期化するコード (コードの太字部分) を CustomerreviewDetail コンストラクタに追加します。

```
public RestaurantDetail(java.lang.String restaurantname,
java.lang.String customername, java.lang.String review){
    System.out.println("Creating new CustomerreviewDetail");
    setRestaurantname(restaurantname);
    setCustomername(customername);
    setReview(review);
}
```

6. 「Data」パッケージを右クリックして、「すべてをコンパイル」を選択します。

   パッケージがコンパイルされます。

   続いて、詳細クラスのインスタンスを取り出す取得メソッドをエンティティ Bean に作成します。

エンティティ Bean に対する、詳細クラスをフェッチするビジネスメソッドの作成

ここでは、対応する詳細クラスのインスタンスを返すメソッドを各エンティティ Bean に作成します。

取得メソッドを作成するには、次の操作を行います。

1. エクスプローラ で「Restaurant (EJB)」論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。

   「新規ビジネスメソッドを追加」ダイアログが表示されます。

2. 「名前」フィールドに getRestaurantDetail と入力します。

3. 戻り値の型では、「ブラウズ」ボタンを使用して、「RestaurantDetail」クラスを選択します。

   「戻り値の型」フィールドに Data.RestaurantDetail と表示されます。

4. 「了解」をクリックしてメソッドを作成します。

5. メソッドをダブルクリックしてソースエディタでメソッドを開き、そのソースコードに次の太字部分を追加します。

```
public Data.RestaurantDetail getRestaurantDetail() {
    return (new RestaurantDetail(getRestaurantname(),
getCuisine(),getNeighborhood(), getAddress(), getPhone(),
getDescription(), getRating()));
}
```

6. エクスプローラ で「Customerreview (EJB)」論理ノードを右クリックして、「新規ビジネスメソッドを追加」を選択します。
7. 「名前」フィールドに getCustomerreviewDetail と入力します。
8. 戻り値の型では、「ブラウズ」ボタンを使用して、「CustomerreviewDetail」クラスを選択します。
9. 「了解」をクリックしてメソッドを作成します。
10. ソースエディタでメソッドを開き、そのソースコードに太字部分を追加します。

```
public Data.CustomerreviewDetail getCustomerreviewDetail() {
    return (new CustomerreviewDetail(getRestaurantname(),
getCustomername(), getReview()));
}
```

11. 「Data」パッケージを右クリックして、「すべてをコンパイル」を選択します。

    パッケージ全体がコンパイルされます。

    これで、このチュートリアルアプリケーションのエンティティ Bean およびその詳細クラスヘルパーの作成は完了しました。次に行うのは Bean のテストです。

エンティティ Bean のテスト

Forte for Java 4 IDE には、エンタープライズ Bean をテストする機能が用意されており、自分でクライアントを作成する必要はありません。この機能はアプリケーションサーバーとして J2EE RI を使用します。この環境では、エンタープライズ Bean は JavaServer Pages 技術を使用するアプリケーションの一部として配備され、テストク

ライアントは Web ブラウザに表示されます。このページを使用して、Bean のインスタンスを作成し、Bean の生成、検索およびビジネスメソッドを実行することができます。

このテスト機能を使用して、`Restaurant` Bean の **生成**および `getRating` メソッドを実行してみてください。

エンティティ Bean のテストクライアント作成

テストクライアントを作成すると、EJB モジュール 1 つと J2EE アプリケーションモジュール 1 つ、それに多数のサポート要素が自動的に生成されます。

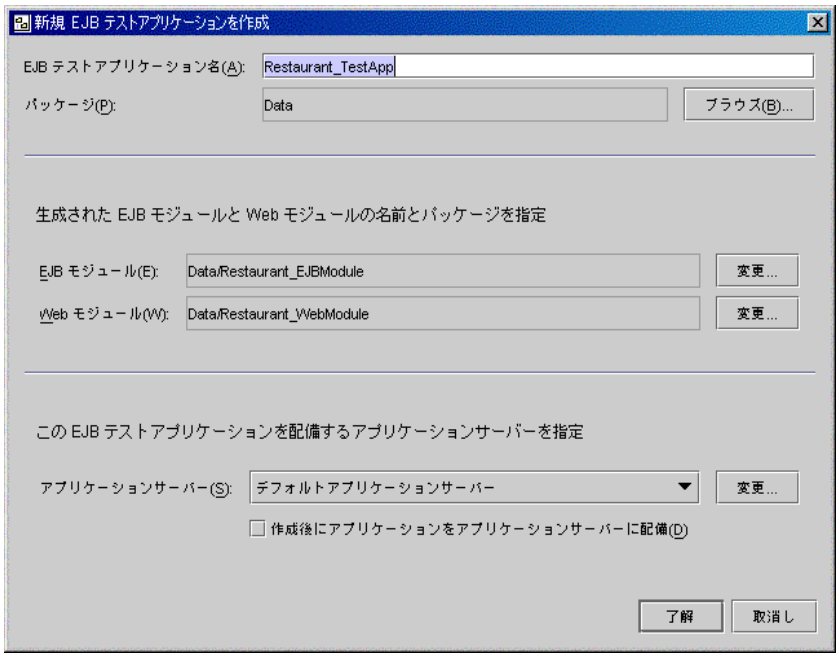
`Restaurant` エンティティ Bean のテストクライアントを作成するには、次の操作を行います。

1. 「`Restaurant (EJB)`」論理ノードを右クリックして、「新規 EJB テストアプリケーションを作成」を選択します。

   EJB テストアプリケーションウィザードが表示されます。

2. すべてのデフォルト値をそのまま受け入れます。

ウィザードが次のような表示になります。

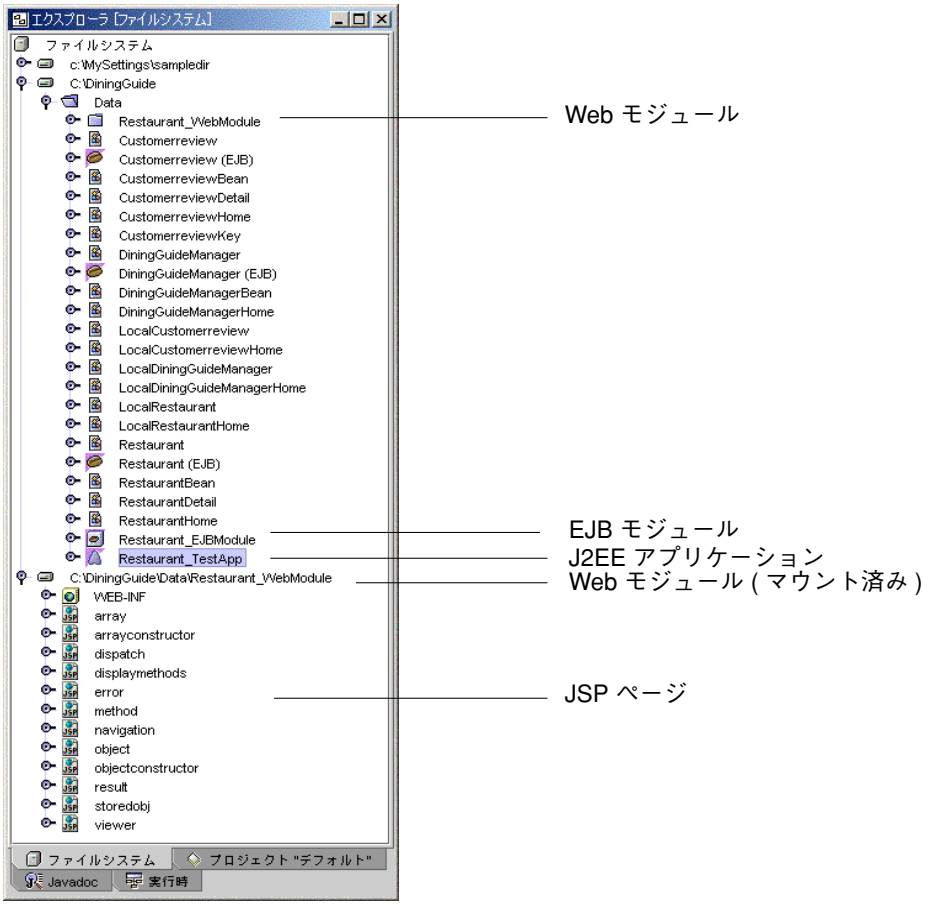


3. 「了解」をクリックします。

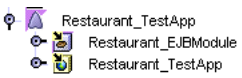
   進捗モニターが少しの間表示され、処理が完了すると消えます。別のウィンドウが表示され、作成された Web モジュールが「プロジェクト」タブでも見えることを示して自動的に消えます。消えない場合は、「了解」をクリックしてウィンドウを閉じてください。

4. 生成されたテストオブジェクトをエクスプローラで確認します。

`Restaurant_EJBModule` という EJB モジュールと `Restaurant_WebModule` という Web モジュール (別にマウントされている)、`Restaurant_TestApp` という J2EE アプリケーションが生成されています。Web モジュールには、テストクライアントをサポートする JSP ページがいくつか含まれています。J2EE アプリケーションには、EJB モジュールと Web モジュールに対する参照が含まれています。



また、作成された J2EE アプリケーションには、Web モジュールと EJB モジュールに対する参照が含まれています。これらのオブジェクトは、`Restaurant_TestApp` を開くと見ることができます。



RI プラグインに対する PointBase 情報の提供

テストクライアントがデータベースを検出して、ログオンできるようにするには、EJB モジュールのリファレンス実装 (RI) プロパティに PointBase 情報を追加する必要があります。

必要な情報を追加するには、次の操作を行います。

1. エクスプローラで EJB モジュール (`Restaurant_EJBModule`) を選択し、そのプロパティシートを表示します。

   「プロパティ」ウィンドウが表示されていない場合は、「表示」->「プロパティ」を選択します。

2. 「プロパティ」ウィンドウの「J2EE RI」タブを選択します。

注 - サーバーレジストリにリファレンス実装のインスタンスがない場合、「プロパティ」ウィンドウに「J2EE RI」タブは表示されません。この問題の解決については、8 ページの「デフォルトのアプリケーションサーバーと Web サーバーの確認」を参照してください。

---

3. 「データソースの JNDI 名」フィールドに `jdbc/Pointbase` と入力します。

注 - Pointbase というように最初の文字だけ大文字入力してください。J2EE RI デフォルトプロパティファイルのプロパティ指定と完全に一致する必要があります。

---

4. 「データソースのパスワード」フィールドに `PBPUBLIC` と入力します。

   これはアスタリスクで表示されます。

5. 「データソースのユーザー名」フィールドに `PBPUBLIC` と入力します。

6. 「SQL 配備の設定」フィールドを選択して、省略符号ボタン (...) を表示します。

7. 省略符号ボタン (...) をクリックして、「SQL 配備の設定」プロパティエディタを表示します。

   左の欄に `Restaurant` Bean が表示されます。

8. 「`Restaurant Bean`」を選択し、右の欄の 2 つのオプションの選択を解除します。

   次の 2 つのオプションです。

- 配備時の表の作成
- 配備取り消し時の表の削除

9 ページの「チュートリアルデータベース表の作成」では、データベースを使用して、Restaurant 表と Customerreview 表を作成しました。アプリケーションを配備するたびに、それらの表を再作成する必要はありません。

9. 「Restaurant」ノードを開きます。

「SQL 配備の設定」プロパティエディタの表示は以下のようになります。

10. メソッド用に作成された SQL を表示するには、そのメソッドを選択します。

選択されたデータベースの SQL が表示されます。findAll メソッド用に生成された SQL を調べると、このメソッドに指定した EJB QL との関係がわかります。

11. 「了解」をクリックして変更を適用し、エディタを閉じます。

12. 「SQL に区切り識別子を使用」プロパティの値を「False」に変更します。

この設定にすると、RI プラグインが引用符で囲った表名や列名を生成しなくなります。(引用符が必要なのは、表や列名に制限語が使用されている場合だけです。

テストアプリケーションの配備

注 - J2EE アプリケーションは PointBase データベースを使用するので、J2EE アプリケーションを配備する前に PointBase Server が動作していることを確認してください (9 ページの「チュートリアルデータベース表の作成」の手順 1 を参照)。また、J2EE RI サーバーが IDE の外部で動作していないことを確認してください。配備プロセスでは、J2EE RI サーバーが自動的に起動されます (すでに動作中の場合は再起動されます)。

`Restaurant` テストアプリケーションを配備するには、次の操作を行います。

1. 「`Restaurant_TestApp`」J2EE アプリケーションノードを右クリックし、コンテキストメニューから「配備」を選択します。

   「進捗モニター」ウィンドウに、J2EE RI サーバーの起動に続いて配備プロセスが表示されます。

2. アプリケーションが配備されていることを確認します。

   アプリケーションが配備されると、出力ウィンドウに
   「完了 `Restaurant_TestApp` .」というメッセージが表示されます。

参照 - 配備で問題が発生した場合は、デフォルトのアプリケーションサーバーが J2EE RI であるかどうかを調べた上で、アプリケーションを再配備してください。正しく設定する方法については、8 ページの「デフォルトのアプリケーションサーバーと Web サーバーの確認」を参照してください。

テストクライアントを使用した エンティティ Bean のテスト

ここでは、Web ブラウザでテストページを開いて、アプリケーションを起動します。テストクライアントの Web ページが表示されたら、`Restaurant` Bean のホームインタフェースの 生成メソッドを使用して、Bean のインスタンスを作成します。そして、そのインスタンスでビジネスメソッド (この場合は `getRating`) をテストします。

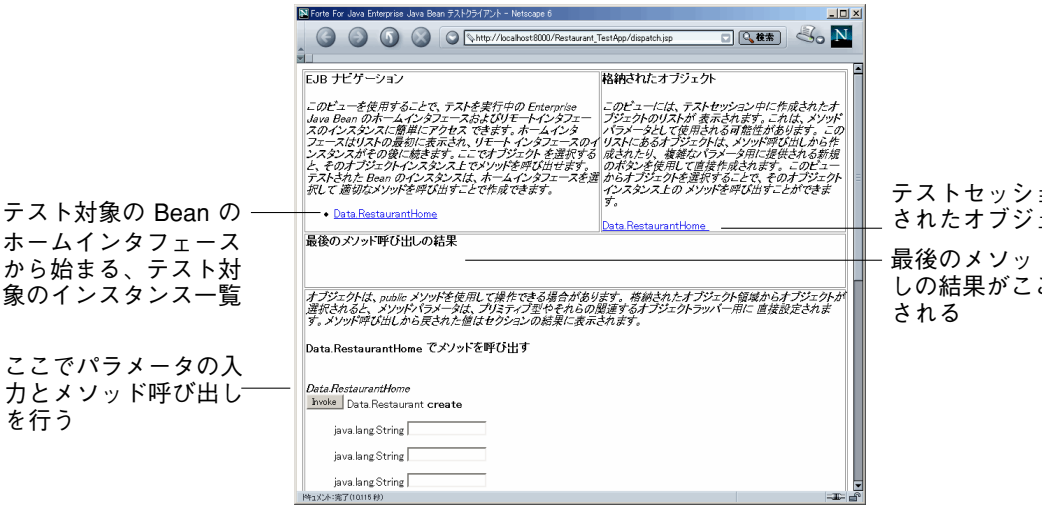
`Restaurant` Bean をテストするには、次の操作を行います。

1. オペレーティングシステムから Web ブラウザを起動し、次の URL を開きます。

   `http://localhost:8000/Restaurant_TestApp`

注 - Port8000 は、J2EE RI が動作するデフォルトのポートです。このポートが適切であるかどうかを調べる必要がある場合は、ブラウザで `http://localhost:8000` を開いてください。J2EE RI のデフォルトページが表示されます。

ブラウザにテストクライアントが表示されます。



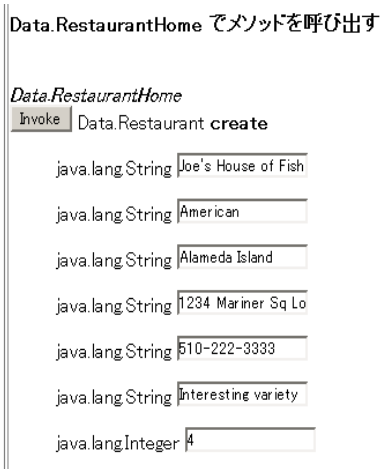
2. 生成メソッドを呼び出して、`Restaurant` Bean のインスタンスを作成します。

   生成メソッドは、「Data.RestaurantHome でメソッドを呼び出す」という見出しの下にあり、その下に 7 つのフィールドがあります。これらのフィールドには名前がありませんが、順序が 43 ページの「CMP エンティティ Bean の生成メソッドの作成」の手順 3 でフィールドを作成したのと同じ順序なので、どのようなフィールドであるかを推測することができます。

注 - `Restaurant.create` メソッドをダブルクリックすると、ソースエディタにそのメソッドが表示されます。フィールドの順序は、メソッドの定義にあるとおりです。

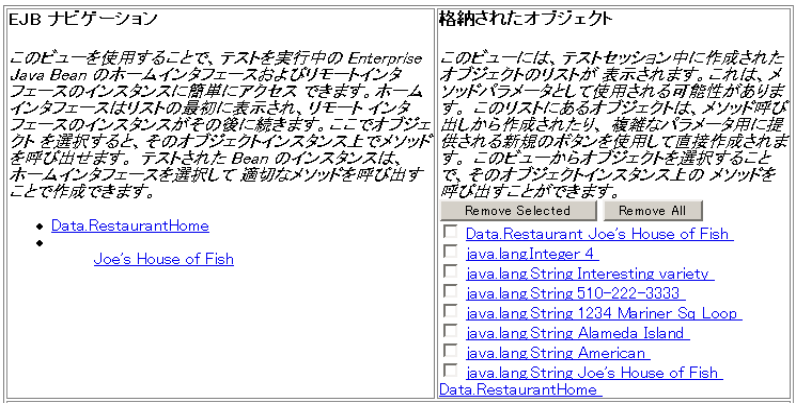
参照 - パラメータを別の順序で表示する場合は、エクスプローラで「Restaurant.create」メソッドノードを右クリックし、「カスタマイズ」を選択します。「カスタマイザ」ダイアログで「上へ」ボタンと「下へ」ボタンをクリックして選択し、パラメータの順序を調整します。その後で、エクスプローラでノードを右クリックして「配備」を選択し、テストアプリケーションを再配備します。

フィールドに任意のデータを入力します (フィールドの順序は異なる場合があります)。

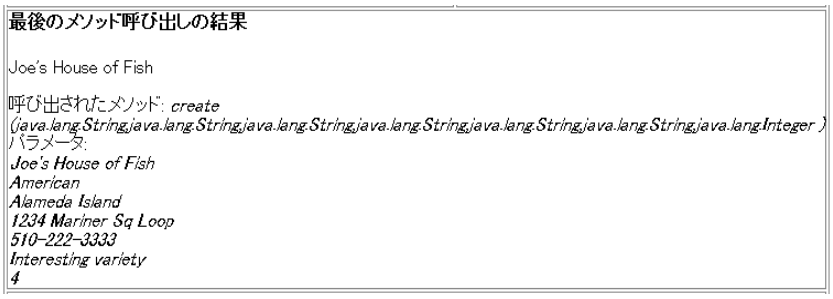


3. Createメソッドの「Invoke」ボタンをクリックします。

配備されたテストアプリケーションによって、作成したレコードがテストデータベースに追加されます。次のように、新しい Restaurant インスタンスの restaurantname の値が左上に表示され、データオブジェクトが右上に示されます。



結果は、結果領域に表示されます。



4. 「Invoke」ボタンをクリックし、 Restaurant Bean の findAll メソッドをテストします。

結果の領域には、次のように表示されます。

ここでは、3 つの内容が返されます。これは、手順 3 で作成した新しいデータベースレコードが、第 1 章で作成した 2 つのデータベース表に追加されたことを表しています。

5. `Bay Fox` と入力してメソッドの「Invoke」ボタンをクリックし、`findByPrimaryKey` メソッドをテストします。

   結果の領域には、`Bay Fox` レコードが返されたことが示されます。

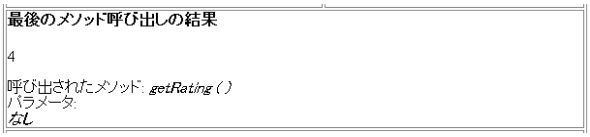
   次に、エンティティ Bean のビジネスメソッドをテストします。

6. 左上の インスタンス一覧で `Data.RestaurantHome` の下に表示されている「Joe's House of Fish」をクリックします。

   「`Invoke Methods`」領域に、`getRating` メソッドが表示されます。

7. `getRating` メソッドの「Invoke」ボタンをクリックします。

   この操作の結果は、結果領域に次のように表示されます。



   これは、データベースに新しいレコードが作成され、`getRating` メソッドを使用してフィールドのいずれかの値が取得されたことを表しています。

   作成されたオブジェクトを選択し、そのメソッドを呼び出してテストを続行します。たとえば、`Data.RestaurantDetail` オブジェクトのいずれかの値を選択すると、取得メソッドを呼び出してデータを表示することも、設定メソッドを呼び出してデータベースに新しいデータを書き込むこともできます。

8. テストが終了したら、Web ブラウザで別の URL を指定するかブラウザを終了し、テストクライアントを停止します。

   `Restaurant_TestApp` と `Customerreview_TestApp` は、第 4 章で使用します。

9. `Customerreview` エンティティ Bean のテストクライアントを作成し、`Customerreview` Bean について、56 ページの「エンティティ Bean のテストクライアント作成」から始まるすべての手順を再実行してテストします

   テスト URL は `http://localhost:8000/Customerreview_TestApp` です。

データベースへの追加内容の確認

`Restaurant_TestApp` アプリケーションによってデータベースにデータが追加されたことを確認するには、次の操作を行います。

1. PointBase コンソールを起動します。

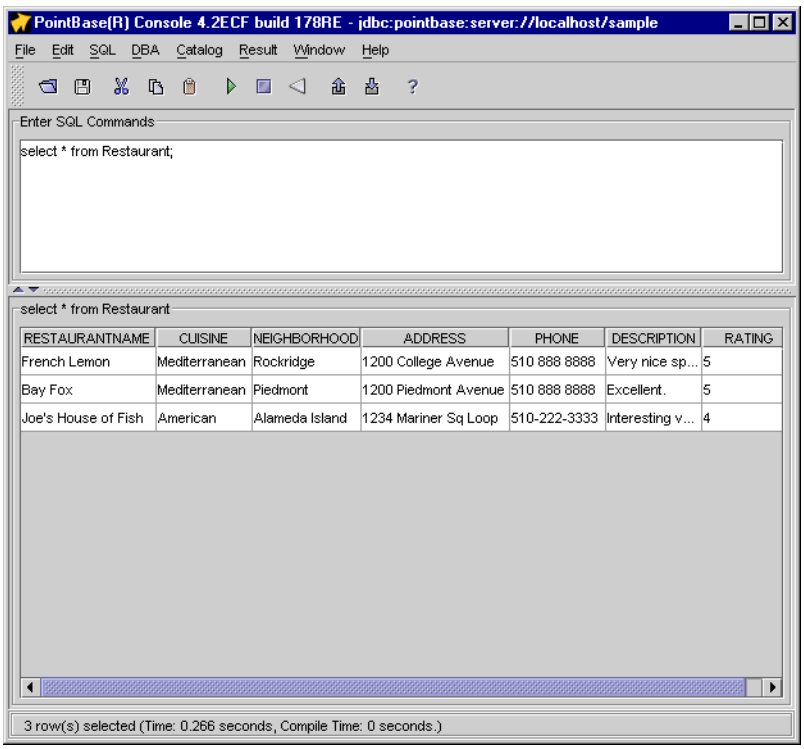
   9 ページの「チュートリアルデータベース表の作成」の手順 2 を参照してください。

2. 次の SQL を PointBase コンソールに入力します。

   ```
   select * from Restaurant;
   ```

3. 「SQL」->「Execute」を選択して SQL 文を実行します。

   61 ページの「テストクライアントを使用した エンティティ Bean のテスト」の手順 2 で実際に値を入力した場合、結果は次のようになります。



確認を終えたら、次節のセッション Bean の作成に進みます。

注 - J2EE RI プロセスを停止する必要はありません。配備のたびに、IDE はアプリケーションの配備を取り消し、J2EE RI サーバーを再起動します。IDE が終了すると、J2EE RI インスタンスのプロセスの終了を告げるダイアログが表示されます。必要ならば、「実行」ウィンドウで「RI インスタンス 1」ノードを右クリックし、「プロセスを終了」を選択すると、いつでも手動でプロセスを終了することもできます。

EJB ビルダーによるエンティティ Bean の作成

ここでは、クライアント (第 4 章で作成する Web サービスのこと) とエンティティ Bean 間の会話を管理するステートレスセッション Bean を作成します。

注 - 付録 A に、完成したセッション Bean のソースコードがあります。

EJB アーキテクチャのバージョン 2.0 では、セッション Bean はローカルインタフェースかリモートインタフェース、あるいはその両方を持つことができます。このチュートリアルでは、セッション Bean のメソッドは、テストアプリケーション (セッション Bean にローカル)、Web サービス (ローカル)、クライアント (リモート) によって呼び出されます。このため、ローカルとリモートの両方のインタフェースを持つセッション Bean を作成します。

1. Forte for Java 4 のエクスプローラで「Data」パッケージを右クリックして、「新規」->「J2EE」->「セッション EJB」を選択します。

   新規ウィザードの「セッション Bean 名とプロパティ」区画が表示されます。

2. 「名前」フィールドに `DiningGuideManager` と入力します。
3. 状態オプションとして「ステートレス」を選択します。
4. トランザクション型オプションとして「コンテナ管理」を選択します。
5. コンポーネントインタフェースオプションとして「リモートインタフェースとローカルインタフェースの両方」を選択します。

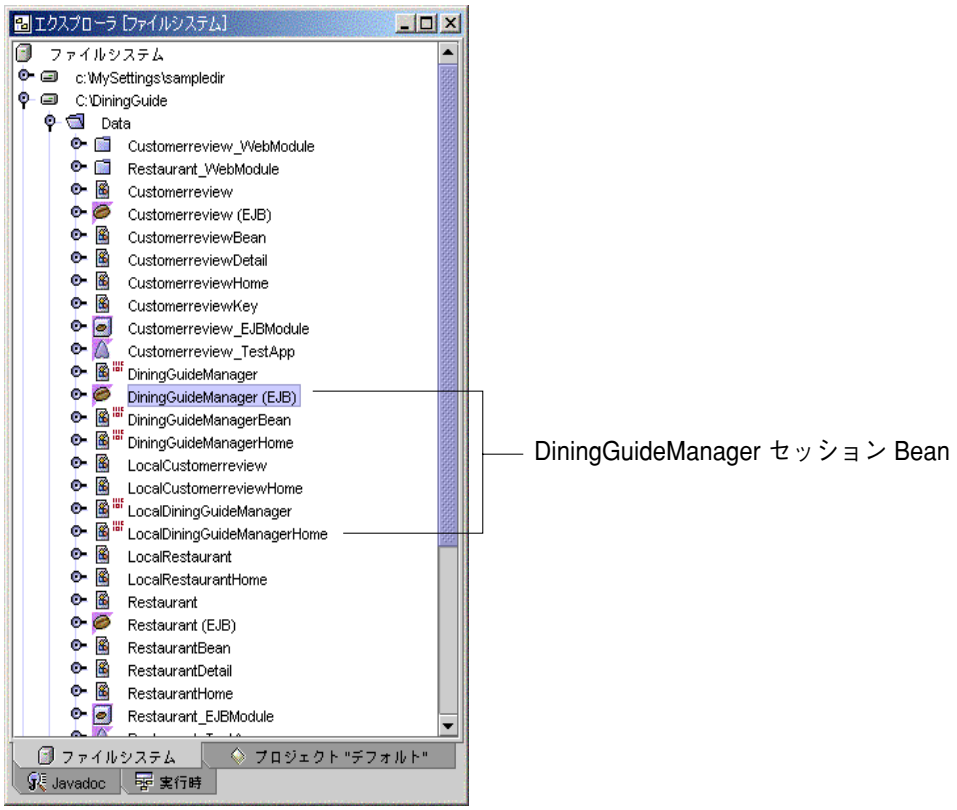
6. 「次へ」をクリックします。

   このセッション Bean について作成されるすべてのコンポーネントの一覧が入った「セッション Bean クラスファイル」区画が表示されます。

   すべてのコンポーネントに DiningGuideManager に基づく名前が付けられていることに注意してください。

7. 「完了」をクリックします。

   エクスプローラに `DiningGuideManager` セッション Bean が表示されます。



   続いて、このセッション Bean のメソッドを作成します。

セッション Bean の生成メソッドのコーディング

生成メソッドは、`DiningGuideManager` セッション Bean を作成したときにすでに作成されており、変更はしません。

ステートレスセッション Bean の生成メソッドには、初期化が必要な状態を持たないため、引数はありません。DiningGuideManager セッション Bean の生成メソッドは、最初に初期コンテキストを作成し、そのコンテキストを使用して必要なリモート参照を取得します。

1. DiningGuideManger の 生成メソッドをダブルクリックして、ソースエディタに表示します。

   論理ノード (DiningGuideManager(EJB)) を使用して、メソッドを探してください。

2. RestaurantHome インタフェースへのリモート参照用の JNDI ルックアップを使用し、メソッドのコーディングを開始します。

```
public void ejbCreate(){
// ソースエディタでは、次の 2 行を 1 行にする
    System.out.println("Entering
DiningGuideManagerEJB.ejbCreate()");
    Context c = null;
    Object result = null;

    if (this.myRestaurantHome == null) {
        try {
            c = new InitialContext();
            result = c.lookup("Restaurant");
            myRestaurantHome =
(RestaurantHome)javax.rmi.PortableRemoteObject.narrow (result,
RestaurantHome.class);
        }
        catch (Exception e) {System.out.println("Error: "+ e); }
    }
```

注 - ソースエディタに入力したコードは、その部分を選択して Control+Shift F を押すと、書式を整えることができます。

---

3. 同様に CustomerreviewHome 用の JNDI ルックアップを前のコードの下に追加します。

```
    Context crc = null;
    Object crresult = null;

    if (this.myCustomerreviewHome == null) {
        try {
            crc = new InitialContext();
            result = crc.lookup("Customerreview");
            myCustomerreviewHome =
(CustomerreviewHome)javax.rmi.PortableRemoteObject.narrow(result
, CustomerreviewHome.class);
        }
        catch (Exception e) {System.out.println("Error: "+ e); }
    }
```

4. javax.naming パッケージのインポート文を追加します。

   ファイルの先頭にインポート文を追加してください。javax.naming をインポートするのは、手順 2 で使用した ルックアップメソッドが含まれているためです。

```
import javax.naming.*;
```

5. myRestaurantHome フィールドと myCustomerreviewHome フィールドを宣言します。

   DiningGuideManagerEJB セッション Bean の定義の import 文の後に、これらの宣言を追加します。

```
public class DiningGuideManagerBean implements
javax.ejb.SessionBean {
    private javax.ejb.SessionContext Context;
    private RestaurantHome myRestaurantHome;
    private CustomerreviewHome myCustomerreviewHome;
```

6. 「ファイル」->「すべてを保存」を選択して、作業内容を保存します。

   続いて、DiningGuideManager のビジネスメソッドを作成します。

詳細データを取得するビジネスメソッドの作成

DiningGuideManager Bean には、クライアントからレストラン一覧の要求を受け取ったときにすべてのレストランデータを取り出すメソッドが必要です。同様に、クライアントから利用者コメント一覧の要求があったときに特定のレストランに関するコメントデータを取り出すためのメソッドも必要です。ここでは、これらの要求を満たす getAllRestaurants と getCustomerreviewsByRestaurant というメソッドを作成します。

getAllRestaurants メソッドの作成

getAllRestaurants ビジネスメソッドを作成するには、次の操作を行います。

1. 「DiningGuideManager」論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。

   「新規ビジネスメソッドを追加」ダイアログが表示されます。

2. 「名前」フィールドに getAllRestaurants と入力します。
3. 「戻り値の型」フィールドに java.util.Vector と入力します。
4. 「了解」をクリックします。

   DiningGuideManager セッション Bean のビジネスメソッドに、メソッドシェルが作成されます。

5. ソースエディタでメソッドを開き、そのソースコードに次の太字部分を追加します。

```
public java.util.Vector getAllRestaurants() {
// ソースエディタでは、次の 2 行を 1 行にする
    System.out.println("Entering
DiningGuideManagerEJB.getAllRestaurants()");
    java.util.Vector restaurantList = new java.util.Vector();
    try {
        java.util.Collection rl = myRestaurantHome.findAll();
        if (rl == null) { restaurantList = null; }
        else {
            RestaurantDetail rd;
            java.util.Iterator rli = rl.iterator();
            while ( rli.hasNext() ) {
                rd =
((Restaurant)rli.next()).getRestaurantDetail();
                System.out.println(rd.getRestaurantname());
                System.out.println(rd.getRating());
                restaurantList.addElement(rd);
            }
        }
    }
    catch (Exception e) {
// ソースエディタでは、次の 2 行を 1 行にする
        System.out.println("Error in
DiningGuideManagerEJB.getAllRestaurants(): " + e);
    }
// ソースエディタでは、次の 2 行を 1 行にする
    System.out.println("Leaving
DiningGuideManagerEJB.getAllRestaurants()");
    return restaurantList;
}
```

このコードは、コンテキスト内の Restaurant Bean のリモート参照ごとに RestaurantDetail のインスタンスを 1 つ取得し、そのインスタンスを restaurantList というベクトルに追加して、そのベクトルを返します。

続いて、利用者コメントの一覧を取得する、同様のメソッドを作成します。

getCustomerreviewsByRestaurant メソッドの作成

getCustomerreviewsByRestaurant メソッドを作成するには、次の操作を行います。

1. 「DiningGuideManager」論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。

   「新規ビジネスメソッドを追加」ダイアログが表示されます。

2. 「名前」フィールドに `getCustomerreviewsByRestaurant` と入力します。

3. 「戻り値の型」フィールドに `java.util.Vector` と入力します。

4. 「追加」ボタンをクリックして、パラメータを追加します。

   「メソッドのパラメータを入力」ダイアログが表示されます。

5. 「フィールド名」フィールドに `restaurantname` と入力します。

6. 「型」フィールドに `java.lang.String` と入力します。

7. 「了解」をクリックしてダイアログを閉じ、メソッドパラメータを作成します。

8. 再び「了解」をクリックしてビジネスメソッドを作成します。

   `DiningGuideManager` セッション Bean にメソッドが作成されます。

9. ソースエディタでメソッドを開き、そのソースコードに次の太字部分を追加します。

```
public java.util.Vector
getCustomerreviewsByRestaurant(java.lang.String
        restaurantname) {
// ソースエディタでは、次の 2 行を 1 行にする
    System.out.println("Entering
DiningGuideManagerEJB.getCustomerreviewsByRestaurant()");
    java.util.Vector reviewList = new java.util.Vector();
    try {
        java.util.Collection rl =
myCustomerreviewHome.findByRestaurantName(restaurantname);
        if (rl == null) { reviewList = null; }
        else {
            CustomerreviewDetail crd;
            java.util.Iterator rli = rl.iterator();
            while ( rli.hasNext() ) {
                crd =
((Customerreview)rli.next()).getCustomerreviewDetail();
                System.out.println(crd.getRestaurantname());
                System.out.println(crd.getCustomername());
                System.out.println(crd.getReview());
                reviewList.addElement(crd);
            }
        }
    }
    catch (Exception e) {
// ソースエディタでは、次の 2 行を 1 行にする
        System.out.println("Error in
DiningGuideManagerEJB.getCustomerreviewsByRestaurant(): " + e);
    }
// ソースエディタでは、次の 2 行を 1 行にする
    System.out.println("Leaving
DiningGuideManagerEJB.getCustomerreviewsByRestaurant()");
    return reviewList;
}
```

getAllRestaurants メソッド同様、このメソッドは、コンテキスト内の Customerreview Bean のリモート参照ごとに CustomerreviewDetail のインスタンスを 1 つ取得し、そのインスタンスを reviewList というベクトルに追加して、そのベクトルを返します。

10. 「ファイル」->「すべてを保存」を選択して、作業内容を保存します。

利用者コメントレコードを作成するビジネスメソッドの作成

ここでは、Customerreview エンティティ Bean の生成メソッドを呼び出して、データベースの新しいレコードを作成するビジネスメソッドを作成します。

createCustomerreview メソッドを作成するには、次の操作を行います。

1. 「DiningGuideManager」論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。

   「新規ビジネスメソッドを追加」ダイアログが表示されます。

2. 「名前」フィールドに createCustomerreview と入力します。
3. 「戻り値の型」フィールドに void と入力します。
4. 「追加」ボタンをクリックして、パラメータを追加します。

   「メソッドのパラメータを入力」ダイアログが表示されます。

5. 「フィールド名」フィールドに restaurantname と入力します。
6. 「型」フィールドに java.lang.String と入力します。
7. 「了解」をクリックしてダイアログを閉じ、メソッドパラメータを作成します。
8. 手順 4 ～ 手順 7 を 2 回繰り返し、次の 2 つのパラメータを作成します。

   ```
   java.lang.String customername
   java.lang.String review
   ```

9. 再び「了解」をクリックしてビジネスメソッドを作成します。

   DiningGuideManager セッション Bean にメソッドが作成されます。

10. ソースエディタでメソッドを開き、そのソースコードに次の太字部分を追加します。

```
public void createCustomerreview(java.lang.String restaurantname,
java.lang.String customername, java.lang.String review) {
// ソースエディタでは、次の 2 行を 1 行にする
    System.out.println("Entering
DiningGuideManagerEJB.createCustomerreview()");
    try {
        Customerreview customerrev =
myCustomerreviewHome.create(restaurantname, customername,
review);
    } catch (Exception e) {
// ソースエディタでは、次の 2 行を 1 行にする
        System.out.println("Error in
DiningGuideManagerEJB.createCustomerreview(): " + e);
    }
// ソースエディタでは、次の 2 行を 1 行にする
    System.out.println("Leaving
DiningGuideManagerEJB.createCustomerreview()");
}
```

11. 「ファイル」->「すべてを保存」を選択して、作業内容を保存します。

詳細クラス型を返すビジネスメソッドの作成

第 4 章で作成する Web サービス は SOAP RPC Web サービスです。この SOAP (Simple Object Access Protocol) というプロトコルは抽象メッセージ技術であり、Web サービス同士が HTTP と XML を使用して互いに通信することを可能にします。Java の型を XML に正しくマッピングするには、SOAP 実行環境は、Web サービスによって呼び出されるすべてのメソッドが使用する、Java のあらゆる型の情報を持っている必要があります。つまり、DiningGuide アプリケーションでも、 Web サービスは セッション Bean のメソッドを呼び出しますから、Web サービスはそれらのメソッドが使用するあらゆる型の情報を持っている必要があります。

SOAP 実行環境が情報を持つことができない型に、コレクションを構成するオブジェクトの型があります。この章で少し前に作成したメソッド (`getAllRestaurants` と `getCustomerreviewsByRestaurant`) はすべて詳細クラスのコレクションを返します。このため、詳細クラスごとにクラスを返すメソッドを作成することによって、SOAP 実行環境にそれらクラスに関する情報を提供する必要があります。ここでは、このためのメソッドとして `getRestaurantDetail` および `getCustomerreviewDetail` を作成します。

ここでは、エンティティ Bean に作成したメソッドと同じ名前を持つメソッドを作成しますが (54 ページの「エンティティ Bean に対する、詳細クラスをフェッチするビジネスメソッドの作成」)、ここで作成するメソッドは空で、その目的は、単に必要な戻り値の型を提供することにあります。

Forte for Java 4 Web サービスおよび SOAP 実行環境についての詳細は、Forte for Java プログラミングシリーズの『Web サービスのプログラミング』を参照してください。

getRestaurantDetail メソッドの作成

getRestaurantDetail メソッドを作成するには、次の操作を行います。

1. 「DiningGuideManager」論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。

   「新規ビジネスメソッドを追加」ダイアログが表示されます。

2. 「名前」フィールドに getRestaurantDetail と入力します。

3. 「戻り値の型」では、「ブラウズ」ボタンを使用して、「RestaurantDetail」クラスを選択します。

   「戻り値の型」フィールドに Data.RestaurantDetail と表示されます。

4. 「了解」をクリックしてビジネスメソッドを作成し、ダイアログを閉じます。

   DiningGuideManager セッション Bean にメソッドが作成されます。

5. ソースエディタでメソッドを開き、そのソースコードに次の太字部分を追加します。

```
public Data.CustomerreviewDetail getCustomerreviewDetail() {
    return null;
}
```

getCustomerreviewDetail メソッドの作成

getCustomerreviewDetail メソッドを作成するには、次の操作を行います。

1. 「DiningGuideManager」論理ノードを右クリックして、「ビジネスメソッドを追加」を選択します。

   「新規ビジネスメソッドを追加」ダイアログが表示されます。

2. 「名前」フィールドに getCustomerreviewDetail と入力します。

3. 「戻り値の型」では、「ブラウズ」ボタンを使用して、「CustomerreviewDetail」クラスを選択します。

   「戻り値の型」フィールドに Data.CustomerreviewDetail と表示されます。

4. 「了解」をクリックしてビジネスメソッドを作成し、ダイアログを閉じます。

   DiningGuideManager セッション Bean にメソッドが作成されます。

5. ソースエディタでメソッドを開き、そのソースコードに次の太字部分を追加します。

```
public Data.CustomerreviewDetail getCustomerreviewDetail() {
    return null;
}
```

6. 「DiningGuideManager(EJB)」を右クリックして、「EJB の検査」を選択します。

   DiningGuideManager セッション Bean が検査されます。

EJB 参照の追加

セッション Bean を配備するとき、その Bean のプロパティには、自身が呼び出すあらゆるエンティティ Bean のメソッドに対する参照が含まれている必要があります。ここでは、セッション Bean に対する参照の追加を行います。EJB モジュールにセッション Bean をアセンブルした後で追加することはできません。

1. エクスプローラで「DiningGuideManager(EJB)」論理ノードを選択します。
2. Bean のプロパティシートを表示します。

   「プロパティ」ウィンドウが表示されていない場合は、「表示」->「プロパティ」を選択します。

3. 「プロパティ」ウィンドウの「参照」タブを選択します。
4. 「EJB 参照」フィールドをクリックして、省略符号ボタン (...) をクリックします。

   「EJB 参照」プロパティエディタが表示されます。

5. 「追加」ボタンをクリックします。

   「追加 EJB 参照」プロパティエディタが表示されます。

6. 「参照名」フィールドに ejb/Restaurant と入力します。

7. 「参照される EJB 名」フィールドでは、「ブラウズ」ボタンをクリックします。

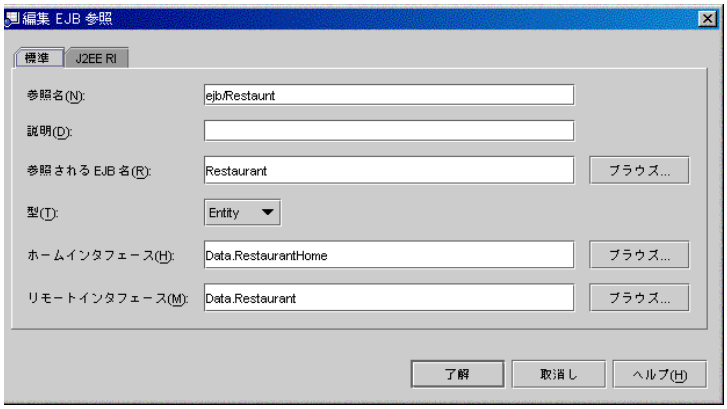
   「EJB を選択」ブラウザが表示されます。

8. 「DiningGuide/Data」ノードの下にある「Restaurant (EJB)」Bean を選択して、「了解」をクリックします。

   「ホームインタフェース」と「リモートインタフェース」フィールドの両方に自動的にデータが入力されることに注意してください。

9. 「型」フィールドを「Entity」に設定します。

   「追加 EJB 参照」プロパティエディタの表示は以下のようになります。



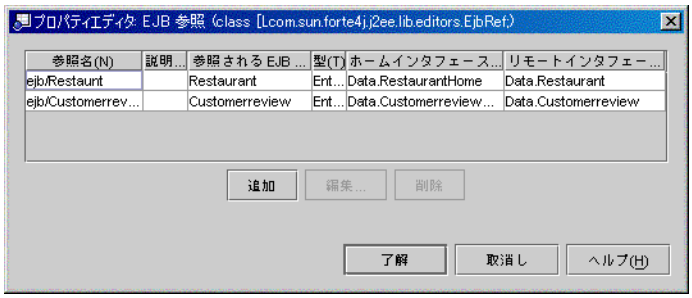
10. 「J2EE RI」タブを選択します。

11. 「JNDI 名」フィールドに jdbc/Pointbase と入力して、「了解」をクリックします。

    続いて、Customerreview エンティティ Bean に対する参照を追加します。

12. `Customerreview` エンティティ Bean に対して、手順 5 〜 手順 11 を繰り返します。

「EJB 参照」ダイアログは次のような表示になります。



13. 「了解」を選択して「プロパティエディタ」ウィンドウを閉じます。

これでチュートリアルアプリケーションの EJB 層が完成し、テストできる状態になりました。エンティティ Bean のテストのとき同様、クライアントがブラウザで読み取り可能な Web 層と JSP ページは、IDE のテストアプリケーション機能によって自動的に作成されます。

セッション Bean のテスト

ここでは、IDE のテストアプリケーション機能を使用して、`DiningGuideManager` セッション Bean をテストします。このセッション Bean のメソッドは EJB 層のすべてのコンポーネントのメソッドへのアクセスを可能にするため、EJB 層全体がテストされることになります。

セッション Bean のテストクライアントの作成

ここでは、`DiningGuideManager` Bean からテストアプリケーションを作成し、EJB モジュールに 2 つのエンティティ Bean を追加します。

セッション Bean 用のテストクライアントを作成するには、次の操作を行います。

1. 「`DiningGuideManager`」論理ノードを右クリックして、「新規 EJB テストアプリケーションを作成」を選択します。

EJB テストアプリケーションウィザードが表示されます。

2. すべてのデフォルト値をそのまま受け入れて、「了解」をクリックします。

   進捗モニターが少しの間表示され、処理が完了すると消えます。別のウィンドウが、作成された Web モジュールが「プロジェクト」タブでも見えることを示し、自動的に消えます。消えない場合は、「了解」をクリックしてウィンドウを閉じてください。

3. 生成されたテストオブジェクトをエクスプローラで確認します。

   IDE によって次のオブジェクトが作成されています。

   - EJB モジュール (`DiningGuideManager_EJBModule`)
   - Web モジュール (`DiningGuideManager_WebModule`)
   - J2EE アプリケーション (`DiningGuideManager_TestApp`)

   EJB モジュールと Web モジュールは、Data パッケージのサブノードとして、また J2EE アプリケーションに含まれるモジュールとして表示されます。また、Web モジュールは別にマウントされています。

   EJB モジュールには `DiningGuideManager` Bean しか含まれていないため、2 つのエンティティ Bean を追加する必要があります。

4. 「`DiningGuideManager_EJBModule`」を右クリックして、「EJB を追加」を選択します。

   「EJB を EJB モジュールに追加」ブラウザが表示されます。

5. 「`DiningGuide`」ファイルシステムを開いて、「Data」パッケージを開きます。

6. Control を押しながら、`Restaurant` および `Customerreview` 論理 Bean を順にクリックし、両方の Bean を選択します。

7. 「了解」をクリックします。

   `DiningGuideManager_EJBModule` は以下のような表示になります。

8. 「ファイル」->「すべてを保存」を選択します。

RI プラグインに対する PointBase 情報の提供

EJB モジュールのリファレンス実装 (RI) プロパティに PointBase 情報を追加する必要があります。これは、59 ページの「RI プラグインに対する PointBase 情報の提供」に対して行ったのと同じ作業です。

必要な情報を追加するには、次の操作を行います。

1. エクスプローラで EJB モジュール (`DiningGuideManager_EJBModule`) を選択し、そのプロパティシートを表示します。

   「プロパティ」ウィンドウが表示されていない場合は、「表示」->「プロパティ」を選択します。

2. 「プロパティ」ウィンドウの「J2EE RI」タブを選択します。

3. 「データソースの JNDI 名」フィールドに `jdbc/Pointbase` と入力します。

注 - Pointbase というように最初の文字だけ大文字入力してください。

---

4. 「データソースのパスワード」フィールドに `PBPUBLIC` と入力します。

   これはアスタリスクで表示されます。

5. 「データソースのユーザー名」フィールドに `PBPUBLIC` と入力します。

6. 「SQL 配備の設定」プロパティの値フィールドを選択して、省略符号ボタン (...) を表示します。

7. 省略符号ボタン (...) をクリックして、「SQL 配備の設定」プロパティエディタを表示します。

   左の欄に `Customerreview` と `Restaurant` Bean が表示されます。

8. Bean を選択して、右の欄の 2 つのオプションの選択を解除します。両方の Bean にこの操作を行ってください。

   - 配備時の表の作成
   - 配備取り消し時の表の削除

9. 「了解」をクリックして変更を適用し、エディタを閉じます。

10. 「SQL に区切り識別子を使用」プロパティの値を「False」に変更します。

11. 「ファイル」->「すべてを保存」を選択し、作業内容を保存します。

テストアプリケーションの配備

注 - J2EE アプリケーションは PointBase データベースを使用するので、J2EE アプリケーションを配備する前に PointBase Server が動作していることを確認してください (9 ページの「チュートリアルデータベース表の作成」の手順 1 を参照)。また、J2EE RI サーバーが IDE の外部で動作していないことを確認してください。配備プロセスでは、J2EE RI サーバーが自動的に起動されます (すでに動作中の場合は再起動されます)。

`Restaurant` テストアプリケーションを配備するには、次の操作を行います。

1. 「`DiningGuideManager_TestApp.`」J2EE アプリケーションノードを右クリックし、コンテキストメニューから「配備」を選択します。

   「進捗モニター」ウィンドウに、J2EE RI サーバーの起動に続いて配備プロセスが表示されます。

2. アプリケーションが配備されていることを確認します。

   アプリケーションが配備されると、出力ウィンドウに
   「完了 DiningGuideManager_TestApp.」というメッセージが表示されます。

参照 - 配備で問題が発生した場合は、デフォルトのアプリケーションサーバーが J2EE RI であるかどうかを調べた上で、アプリケーションを再配備してください。正しく設定する方法については、8 ページの「デフォルトのアプリケーションサーバーと Web サーバーの確認」を参照してください。

続いて、`DiningGuideManager` セッション Bean をテストします。

テストクライアントによるセッション Bean のテスト

テストクライアントの Web ページで 生成メソッドを実行して `DiningGuideManager` セッション Bean のインスタンスを作成し、そのインスタンスでビジネスメソッド (`getRating`) をテストします。

Web ブラウザでテストアプリケーションを指定し、アプリケーションを開始します。

1. オペレーティングシステムで Web ブラウザを起動し、次の URL を指定します。

   `http://localhost:8000/DiningGuideManager_TestApp`

   ブラウザに、テストクライアントとインスタンスリストが表示されます。左上のインスタンスリストには、`DiningGuideManagerHome` のインスタンスのみが示されています。

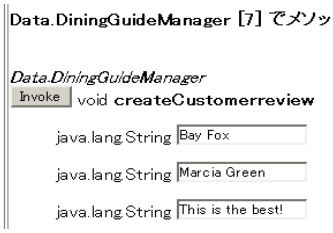
2. `DiningGuideManagerHome` の 生成メソッドを呼び出し、`DiningGuideManager` セッション Bean のインスタンスを作成します。

   `Data.DiningGuideManager[x]` インスタンスがインスタンスリストに表示されます。これで、Bean の取得メソッドをテストすることができます。

3. 新しい「`Data.DiningGuideManager[x]` 」をクリックします。

   `getAllRestaurants` メソッドおよび `getCustomerreviewsByRestaurant` メソッドが有効になります。

4. `createCustomerreview` フィールドに任意のデータを入力します。



5. `createCustomerreview` メソッドの「Invoke」ボタンをクリックします。

   配備されたテストアプリケーションによって、作成したレコードがテストデータベースに追加され、「Stored Objects」セクション (右上) にパラメータ値、結果領域に結果が表示されます。

6. `getAllRestaurants` メソッドの「Invoke」ボタンをクリックします。

   61 ページの「テストクライアントを使用した エンティティ Bean のテスト」でデータベースに Joe's House of Fish のレコードを作成した場合は、作成されたオブジェクトリスト (右上) にサイズ 3 のベクトルが表示され、メソッド呼び出しの結果が以下のように表示されます (実際の数字は異なるかもしれません。また、レコードを作成しなかった場合は、これとは異なる結果になります)。

7. `getCustomerreviewDetail` メソッドの「Invoke」ボタンをクリックします。

   結果領域に結果が表示されます。

8. `getCustomerreviewsByRestaurant` のフィールドに `Joe's House of Fish` と入力して、「Invoke」ボタンをクリックします。

   `CustomerreviewDetail` レコードは返されません。これは、データベースに利用者のコメントがないためです。今度は French Lemon のレコードで試してみます。

9. 同じフィールドに `French Lemon` と入力して、メソッドを呼び出します。

   2 つの `CustomerreviewDetail` レコードが返されます。

10. `getCustomerreviewDetail` メソッドの「Invoke」ボタンをクリックします。

    結果領域に結果が表示されます。

11. テストを終えたら、ブラウザで別の URL を開くかブラウザを終了することによって、テストクライアントを停止します。

注 - J2EE RI プロセスを停止する必要はありません。配備のたびに、IDE はアプリケーションの配備を取り消し、J2EE RI サーバーを再起動します。IDE が終了すると、J2EE RI インスタンスのプロセスの終了を告げるダイアログが表示されます。必要ならば、実行ウィンドウで RI インスタンス 1 のノードを右クリックし「プロセスを終了」を選択すると、いつでも手動でプロセスを終了することもできます。

データベースへの追加内容の確認

`DiningGuideManager_TestApp` アプリケーションによってデータベースにデータが追加されたことを確認するには、次の操作を行います。

1. PointBase コンソールを起動します。

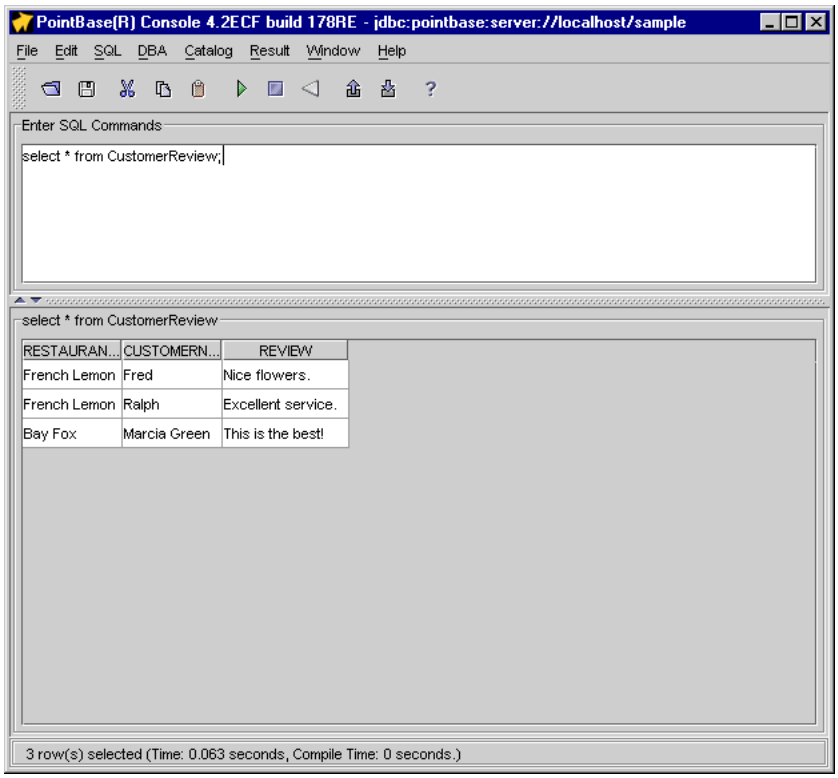
   9 ページの「チュートリアルデータベース表の作成」の手順 2 を参照してください。

2. 次の SQL を PointBase コンソールに入力します。

```
select * from CustomerReview;
```

3. 「SQL」->「Execute」を選択して SQL 文を実行します。

83 ページの「テストクライアントによるセッション Bean のテスト」の手順 4 で実際に値を入力した場合、結果は次のようになります。



これで、Web サービスの作成に進むことができます。

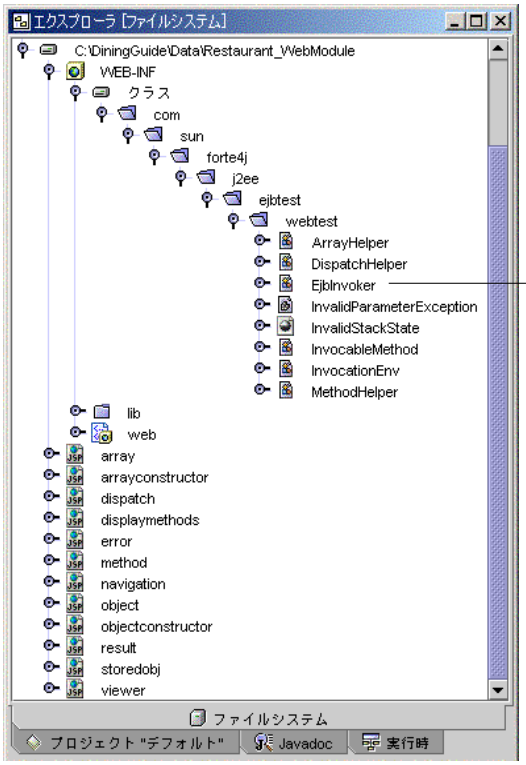
クライアントを作成する際の注意事項

これで、DiningGuide アプリケーションの EJB 層を正しく完成することができました。Forte for Java 4 IDE の Web サービスモジュールを使用したアプリケーションの Web サービスの作成 (第 4 章) と、クライアント用に提供されている Swing クラスのインストール (第 5 章) をすることができます。

独自の Web サービスとクライアントを作成する場合は、Forte for Java 4 テストアプリケーションにあるガイドラインを参照してください。

`DiningGuideManager` Bean のように、セッション Bean にアクセスする Web サービスには、ルックアップメソッドを持つサーブレットや JSP ページを定義しておく必要があります。これは、EJB 層で、エンティティ Bean のホームインタフェースとホームオブジェクトを取得するためです。テストアプリケーション機能によって作成された Web モジュールには、必要なコードのサンプルが用意されています。

ルックアップメソッドの例は、Web モジュールの `EjbInvoker` クラスで参照できます。`WEB-INF/Classes/com/sun/forte4j/j2ee/ejbtest/webtest` ディレクトリを参照してください。



たとえば、次のメソッドにルックアップコードの例があります。

- `EjbInvoker.getHomeObject`
- `EjbInvoker.getHomeInterface`
- `EjbInvoker.resolveEjb`

第4章

DiningGuide アプリケーションの Web サービスの作成

この章では、Forte for Java 4 IDE を使用して DiningGuide アプリケーション用の Web サービスを作成する方法を学びます。

この章の内容は以下のとおりです。

Forte for Java 4 の Web サービス機能についての全容は、Forte for Java プログラミングシリーズの『Web サービスのプログラミング』を参照してください。このマニュアルは、Forte for Java 4 のポータルサイトの Documentation ページ (`http://forte.sun.com/ffj/documentation/index.html`) から入手できます。また、個々の機能については、Forte for Java 4 のオンラインヘルプを参照してください。

Web サービスの概要

この章では、DiningGuide アプリケーションの Web サービスを作成します。この一環として、いくつかのコンポーネントを自分で作成したり、生成したりします。

自分で作成するコンポーネントは以下のとおりです。

- 論理 Web サービス (`DiningGuideWebService` Web サービス)

- J2EE アプリケーション - セッション Bean の EJB モジュールと Web サービスの両方を参照します。

生成するクラスは以下のとおりです。

- 実行時クラス - Web サービスを実装するための EJB コンポーネントです。
- テストクライアント
- テストクライアントのプロキシ

Web サービス

Web サービスとその作成・プログラミング方法についての全容は、『Web サービスのプログラミング』をご覧ください。また、個々の Web サービス項目と手順については、Forte for Java 4 のオンラインヘルプを参照してください。

IDE で作成する Web サービスは、Web サービス内のオブジェクトセット全体を表す論理的なエンティティであり、このエンティティを使用して、Web サービスをプログラミングすることができます。このチュートリアルでは、クライアントがアクセスできるようにするメソッドに対する参照を Web サービスに作成することによって、Web サービスの機能を実現します。また、サポート用の EJB コンポーネントや Web サービスが参照する必要があるあらゆるドキュメントファイル (JSP ページや HTML ファイルなど) を IDE を使用して生成します。DiningGuide アプリケーションでは必要ありませんが、他の Web サービスでは、画像ファイルなどの他の種類のドキュメントを追加することができます。

実行時クラス

Web サービスのプログラミングを終了したら、その実行時クラスを生成します。実行時クラスを直接いじるのではなく、論理 Web サービスを含むパッケージに生成されたものを確認します。

クライアントプロキシページ

Web サービスを配備すると、クライアントプロキシが生成され、論理 Web サービスの `Documents` ディレクトリにサポート用のクライアントページが書き込まれます。Web サービスのテストには、それらのクライアントページを使用します。また、それらのページは、フル機能の参照メソッドをプログラミングする際のスタート地点またはガイドとして利用することもできます。クライアントプロキシページとしては、各

参照メソッドの JSP ページや Web サービス のエラーを表示するための JSP ページ、Web ブラウザでメソッドの JSP ページをまとめて表示するための HTML 形式の開始ページなどがあります。

開始ページには、参照されたメソッド用に生成される各 JSP ページの HTML フォームが 1 つ含まれます。メソッドがパラメータを必要とする場合、HTML フォームにはそれに応じた入力フィールドが含まれます。メソッドのテストでは、必要に応じて各パラメータのデータを入力し、メソッドの「Invoke」ボタンをクリックします。この結果、次のアクションが発生します。

1. JSP ページが SOAP クライアントプロキシに要求を渡します。
2. SOAP クライアントプロキシがアプリケーションサーバー上の Apache SOAP 実行環境システムに要求を渡します。

   SOAP 要求は XML ラッパーで、Web サービスに対するメソッド呼び出しとシリアル化された形式の入力データを含みます。
3. アプリケーションサーバー上の Apache SOAP 実行環境システムが SOAP 要求を、DiningGuide Web サービスの参照する適切なメソッドに対するメソッド呼び出しに変換します。
4. メソッド呼び出しが、EJB 層の適切なビジネスメソッドに渡されます。
5. 処理された応答がチェーンをのぼって SOAP クライアントプロキシに返されます。
6. SOAP クライアントプロキシが応答を JSP ページに渡し、そのページが Web ページに表示されます。

Web サービス層の作成

チュートリアルアプリケーションの Web サービスの作成では、以下の作業を行います。

- Web サービスモジュールの作成 - この次の節で説明

  IDE の Web サービスウィザードを使用して、論理 Web サービスを作成し、参照するメソッドを指定します。

- Web サービスの SOAP RPC の URL を指定 - 94 ページ

  SOAP `rpcrouter` サーブレットは、アプリケーションサーバー上の Apache SOAP 実行環境です。Web サービスのプロパティとして URL を指定することによって、その場所を Web サービスに教えます。

- 「Web サービスの実行時クラスの生成」 - 95 ページ

  この作業では、Web サービスのテストと実装に使用されるサポート用 EJB コンポーネントを生成します。

Web サービスモジュールの作成

ここでは、新規 Web サービスウィザードを使用して、論理 Web サービスを作成します。このウィザードには、アーキテクチャの選択肢として多層と Web 主体の 2 つがあります。多層では、Web サービスによるビジネスメソッドの呼び出しがアプリケーションサーバー内のコンポーネントにのみ制限されるのに対し、Web 主体では、アプリケーションサーバーまたは Web サーバーどちらのコンポーネントにもメソッド呼び出しを行うことができます。DiningGuide アプリケーションの Web サービス は EJB 層のコンポーネント上のメソッドを呼び出すため、アーキテクチャの選択では多層を選択します。

ウィザードではまた、Web サービスが呼び出すメソッドを選択するよう求められます。これに対しては、EJB 層のセッション Bean の 5 つのビジネスメソッドを選択します。

チュートリアルアプリケーションの Web サービスモジュールを作成するには、次の操作を行います。

1. エクスプローラでマウントされている「`DiningGuide`」ファイルシステムを右クリックして、「新規」->「Java パッケージ」を選択します。

   新規パッケージダイアログが表示されます。

2. 名前として `WebService` と入力し、「完了」をクリックします。

   `DiningGuide` ディレクトリに `WebService` パッケージが表示されます。

3. 「`WebService`」パッケージを右クリックして、「新規」->「Web サービス」->「Web サービス」を選択します。

   新規ウィザードに「Web サービス」区画が表示されます。

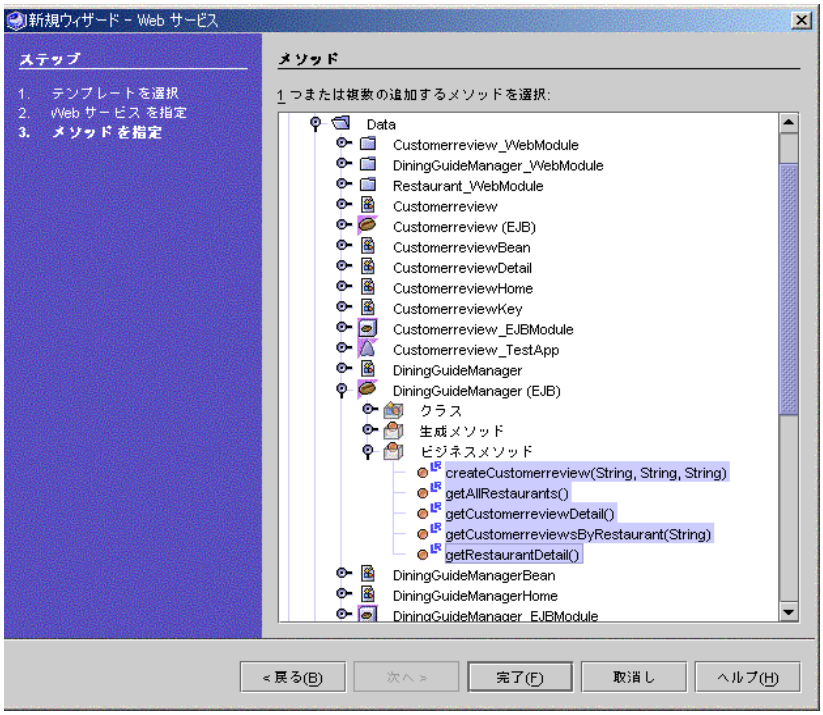
4. 「名前」フィールドに DiningGuideWebService と入力し、アーキテクチャとして「多層」オプションが選択されていることを確認し、「次へ」をクリックします。

   新規ウィザードの「メソッド」区画が表示されます。

5. 「Data」ノードを開いて、「DiningGuideManager(EJB)」->「ビジネスメソッド」ノードを開きます。

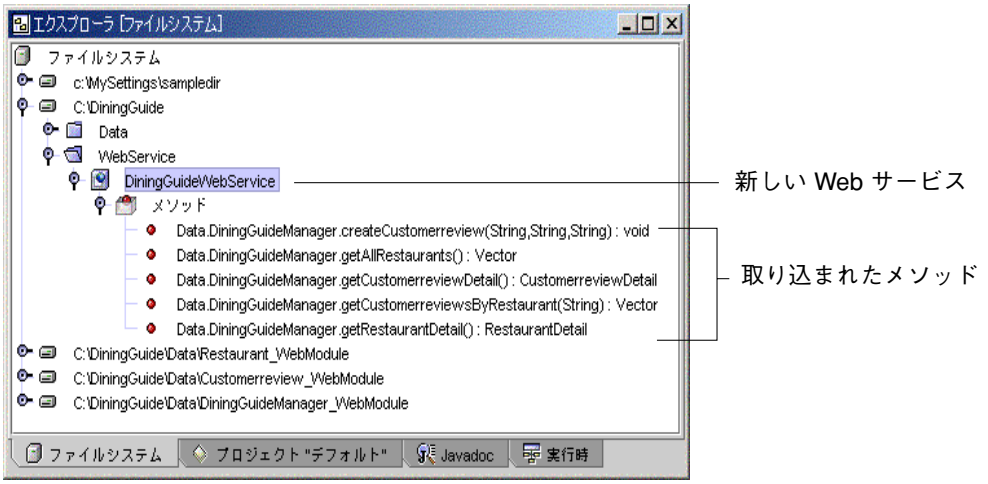
6. Control を押しながら、クリックすることによって、DiningGuideManger のビジネスメソッドをすべて選択します。

   「メソッド」区画が次のような表示になります。



7. 「完了」をクリックします。

エクスプローラの WebService パッケージの下に「DiningGuideWebService」Web サービス (青色の球が入ったアイコン) が表示されます。このノードを開くと、エクスプローラの表示が以下のようになります。



Web サービスの SOAP RPC の URL を指定

SOAP RPC URL プロパティは、アプリケーションサーバー上の Apache SOAP 実行環境の SOAP rpcrouter サーブレットの場所を示します。このプロパティには、「コンテキストルート」または「Web コンテキスト」と呼ばれる文字列が含まれます。この文字列は、後ほど 98 ページの「Web コンテキストプロパティの指定」で作成する J2EE アプリケーション WAR ノードの Web コンテキストプロパティと同じです。

SOAP RPC の URL プロパティを設定するには、次の操作を行います。

1. 「DiningGuideWebService」ノードのプロパティを表示します。

「DiningGuideWebService」ノードを選択すると、「プロパティ」ウィンドウにプロパティが表示されます。「プロパティ」ウィンドウが表示されていない場合は、「表示」->「プロパティ」を選択します。

2. 「SOAP RPC URL」プロパティのプロパティエディタを表示します。

値フィールド内を 1 回クリックし、表示される省略符号ボタンをクリックすると、プロパティエディタが表示されます。

3. URL 内の `DiningGuideWebService` という文字列の部分を `DiningGuideContext` に変更します。URL 全体は以下のようになります。

   ```
   http://localhost:8000/DiningGuideContext/servlet/rpcrouter
   ```

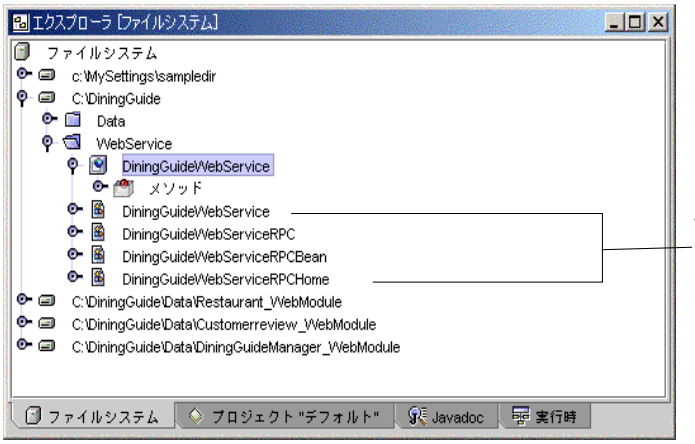
Web サービスの実行時クラスの生成

J2EE アプリケーションとして Web サービスをアセンブルしてテスト用に配備するには、Web サービスの実行時クラスを生成する必要があります。多層アーキテクチャの場合、IDE は 4 つのクラス (このうちの 3 つは生成された EJB コンポーネント用) を生成することによって、Web サービスを実装します。

Web サービスの実行時クラスを生成するには、次の操作を行います。

- 「`DiningGuideWebService`」ノードを右クリックし、「Java ファイルの生成 / コンパイル」を選択します。

  処理が完了すると、IDE の出力ウィンドウに「完了.」というメッセージが表示されます。SOAP RPC Web サービスを実装するための EJB コンポーネントとなる実行時クラスが、エクスプローラに表示されます。



Web サービスのテスト

Web サービスのテストでは、次の作業を行う必要があります。

1. 以下のテストクライアントの作成

- テストクライアント
- EJB モジュールと Web サービスの両方を参照する J2EE アプリケーション

2. Web サービス WAR ファイルの Web コンテキストプロパティの指定

3. テストアプリケーションの配備

4. テストアプリケーションを使用した Web サービスのテスト

Web サービステストアプリケーションは、Web サービスでの XML 操作ごとに 1 つの JSP ページと、ブラウザでそれらのページをまとめて表示する開始ページを生成します。テストクライアントを実行するということは、開始ページから XML 操作を行うということです。

テストクライアントおよびテストアプリケーションの作成

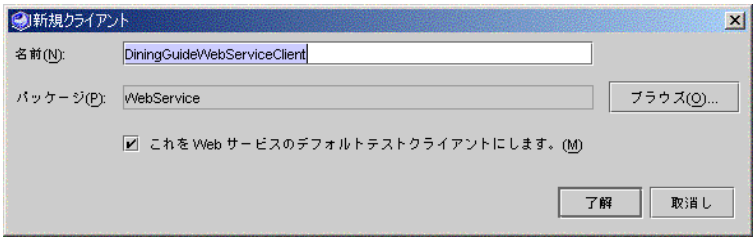
Web サービスをテストするには、テストクライアントと J2EE アプリケーションを作成します。このうちの J2EE アプリケーションに EJB モジュールと Web サービスモジュールを追加します。

参照 - テストクライアントの作成では、そのクライアントを Web サービス用のデフォルトのテストクライアントにします。 J2EE アプリケーションを配備すると、テストクライアントも配備されます。

Web サービスのクライアントアプリケーションを作成して配備するには、次の操作を行います。

1. エクスプローラで 「DiningGuideWebService」ノード ( ) を右クリックして、「新規クライアント」を選択します。

   「新規クライアント」ダイアログが表示されます。



   デフォルトでは、このクライアントを Web サービスのデフォルトのテストクライアントにするオプションが選択されています。

2. すべてのデフォルト値をそのまま受け入れて、「了解」をクリックします。

   エクスプローラにクライアントノード ( ) が表示されます。続いて、Web サービスの J2EE アプリケーションを作成します。

3. 「WebService」パッケージを右クリックして、「新規」->「J2EE」->「アプリケーション」を選択します。

   新規ウィザードが表示されます。

4. 「名前」フィールドに DiningGuideApp と入力して、「完了」をクリックします。

   「WebService」パッケージの下に「J2EE アプリケーション」ノード ( ) が表示されます。続いて、このアプリケーションに Web サービスを追加します。

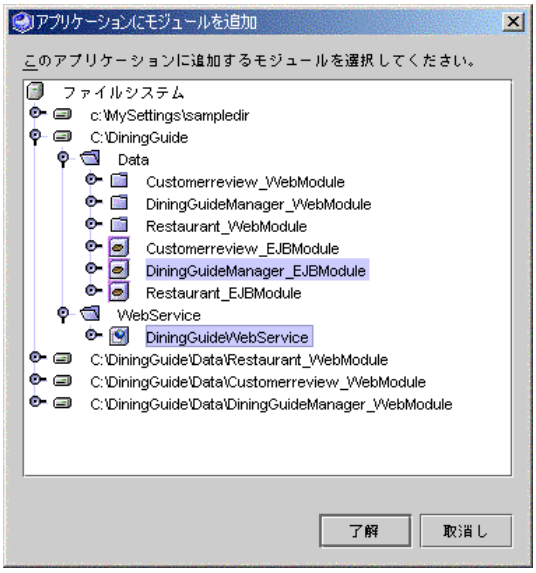
5. 「DiningGuideApp」ノードを右クリックして、「モジュールを追加」を選択します。

   「アプリケーションにモジュールを追加」ダイアログが表示されます。

6. 「DiningGuide」ファイルシステムを開き、「Data」および「WebService」パッケージの両方を開きます。

7. Control を押しながらクリックすることによって、「DiningGuideManager_EJBModule」と「DiningGuideWebService」の両方のノードを選択します。

   ダイアログは以下のような表示になります。



8. 「了解」をクリックして選択を確定し、ダイアログを閉じます。

9. エクスプローラで「DiningGuideApp」J2EE アプリケーションを開きます。

   DiningGuideWebService の WAR および EJB JAR の両方のファイルとともに DiningGuideManager_EJBModule がアプリケーションに追加されていることが分かります。

Web コンテキストプロパティの指定

ここでは、Web サービスの WAR ファイルに J2EE アプリケーションの Web コンテキストを指定します。このコンテキストは、94 ページの「Web サービスの SOAP RPC の URL を指定」で指定したコンテキストと同じである必要があります。

1. 「DiningGuideApp」アプリケーション内の「DiningGuideWebService_War」ファイルの「プロパティ」ウィンドウを表示します。

   ノードを選択すると、「プロパティ」ウィンドウにプロパティが表示されます。「プロパティ」ウィンドウが表示されていない場合は、「表示」->「プロパティ」を選択します。

2. 「Web コンテキスト」フィールドにプロパティ値として DiningGuideContext と入力します。

3. 「ファイル」->「すべてを保存」を選択します。

   これで、DiningGuideApp テストアプリケーションを配備できます。

テストアプリケーションの配備

注 - J2EE アプリケーションは、PointBase データベースを使用します。アプリケーションを配備する前に PointBase Server が動作していることを確認してください (9 ページの「チュートリアルデータベース表の作成」の手順 1 を参照)。また、J2EE RI サーバーが IDE の外部で動作していないことを確認してください。配備プロセスでは、J2EE RI サーバーが自動的に起動されます (すでに動作中の場合は再起動されます)。

DiningGuideApp アプリケーションを配備するには、次の操作を行います。

1. エクスプローラで「DiningGuideApp」ノード () を右クリックして、「配備」を選択します。

   「進捗モニター」ウィンドウに配備の進行状況が示されます。

2. アプリケーションが配備されていることを確認します。

   IDE の「出力」ウィンドウの「RI アプリケーションの配備」タブで、配備の進行状況を確認することができます。最終的に、出力ウィンドウに、DiningGuideApp の配備が完了したことを示すメッセージが表示されます。

   エクスプローラの「実行」ウィンドウに「RI インスタンス 1」ノードが表示されます。

3. IDE の「編集」タブをクリックしてエクスプローラに戻ります。

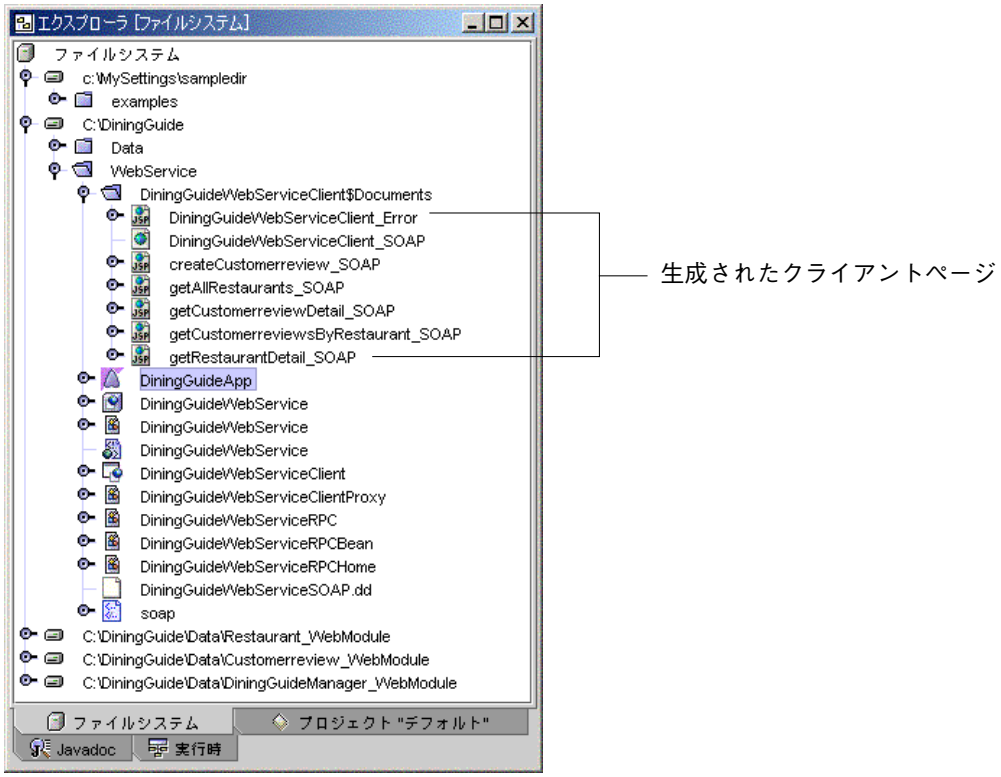
   エクスプローラに DiningGuideWebServiceClientProxy が表示されます。

4. 「WebService」パッケージの下の「DiningGuideWebServiceClient$Documents」ノードを開きます。

次のサポート用項目が作成されていることが分かります。

- 各メソッドの JSP ページ
- HTML 形式の開始ページ
- JSP エラーページ

エクスプローラの表示は以下のようになっています。



これらのファイルは、「DiningGuideWebServiceClient」ノードの下の「生成されたドキュメント」ノードの下でも参照されています。

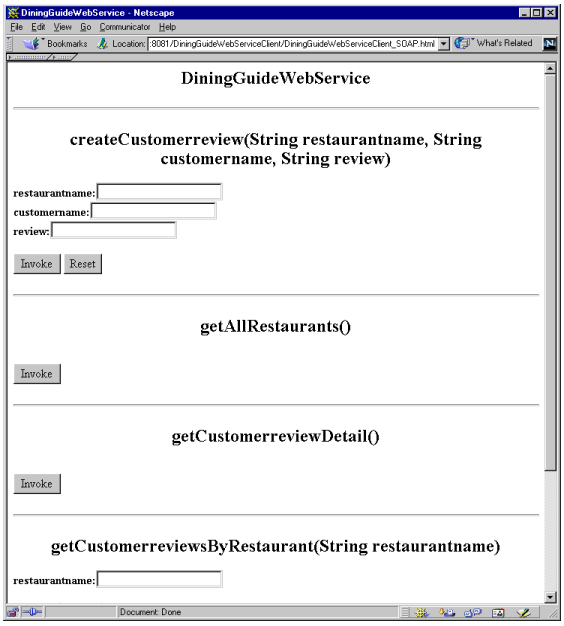
テストアプリケーションを使用した Web サービスのテスト

クライアントと Web サービスの間で SOAP 要求と応答がどのようにやりとりされるかについての詳細は、『Web サービスのプログラミング』を参照してください。このマニュアルは、Forte for Java 4 ポータルサイトの Documentation ページ (`http://forte.sun.com/ffj/documentation/index.html`) から入手できます。

Web サービスをテストするには、次の操作を行います。

1. エクスプローラで「`DiningGuideWebServiceClient`」ノード ( ) を右クリックして、「実行」を選択します。

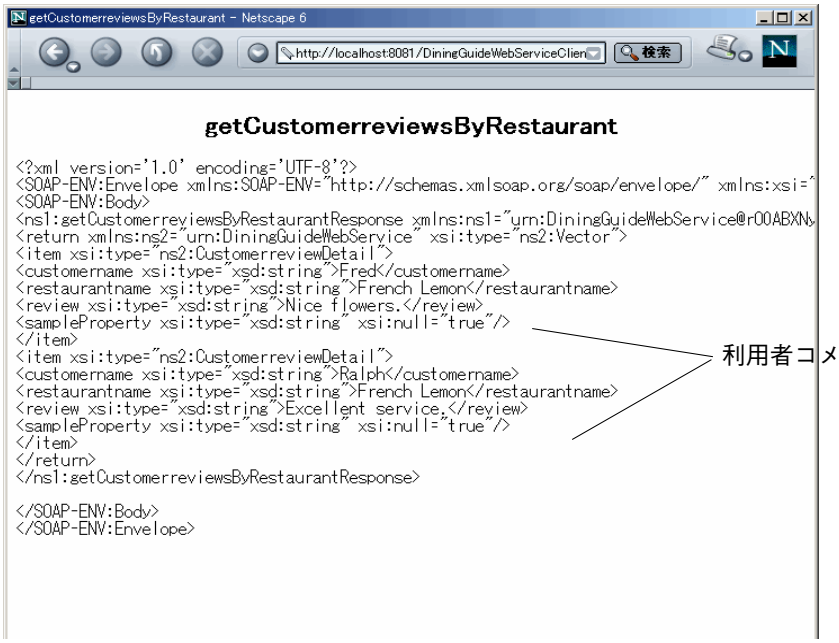
   IDE に組み込まれている Tomcat Web サーバーが自動的に起動され、デフォルトの Web ブラウザが開いて、クライアントの開始ページ (`DiningGuideWebServiceClient_SOAP.html`) が表示されます。



   このページから、操作が予想通りに機能するかどうかをテストすることができます。

2. テキストフィールドに `French Lemon` と入力して、「Invoke」ボタンをクリックし、「`getCustomerreviewsByRestaurant`」をテストします。

SOAP メッセージが作成されて、アプリケーションサーバーに送信されます。`DiningGuideApp` Web サービスはこの SOAP メッセージを `DiningGuideManager.getCustomerreviewsByRestaurant` メソッドのメソッド呼び出しに変換します。そして、このメソッドによってコレクションが返され、このコレクションが、生成された JSP ページ (`getCustomerreviewsByRestaurant_SOAP.jsp`) によって利用者コメントデータとして表示されます。次のような戻り値を含む XML ラッパーが表示されます。



このデータには、French Lemon レストランに関して入力されているすべてのレコードが含まれます。表 1-6 でデータを確認してください。PointBase コンソールを起動し、次の SQL 文を実行することによって確認することもできます。

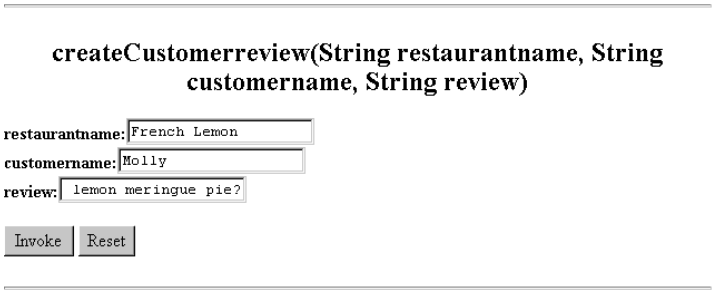
```
select * from CustomerReview;
```

結果として、入力した CustomerReview レコードが表示されます。

3. ブラウザの「戻る」ボタンを使用して、開始ページに戻ります。

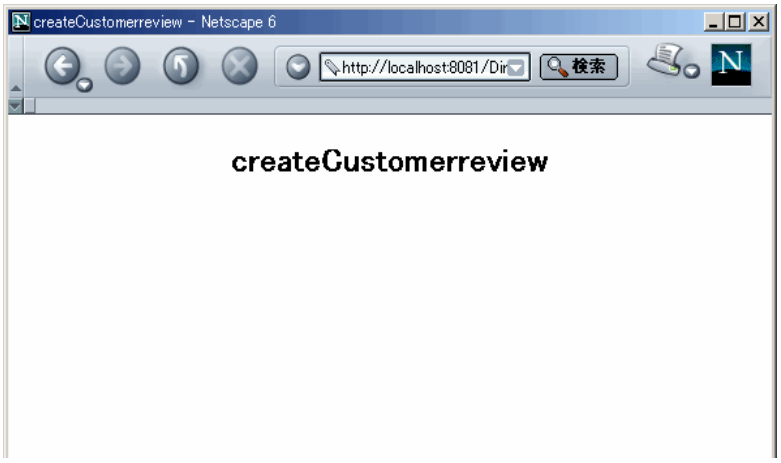
4. 「restaurantname」フィールドに French Lemon と入力し、残りの 2 つのフィールドに任意のデータを入力して「createCustomerreview」操作をテストします。

   例:



5. 「Invoke」ボタンをクリックします。

   このメソッドは、入力パラメータとして複雑な Java オブジェクトを受け取ります。生成された JSP ページである createCustomerreview_SOAP.jsp は、3 つのパラメータの入力を求めます。入力内容は XML に変換されて、SOAP プロトコルに渡され、Web サービス層に要求が送信されます。Web サービスは要求を受信して、データベースに書き込みます。このメソッドは void を返すため、JSP ページは空になります。



6. ブラウザの「戻る」ボタンを使用して、開始ページに戻ります。

7. 続いて、「getAllRestaurants」操作をテストします。開始ページでその「Invoke」ボタンをクリックしてください。

このメソッドは入力パラメータを必要としません。レストランデータのコレクションを返し、そのコレクションが「getAllRestaurants_SOAP.jsp」ページによって XML 表示されます。

ページの最後のレコードが Restaurant エンティティ Bean のテストで入力した Restaurant レコード (53 ページの「テストクライアントを使用したエンティティ Bean のテスト」を参照) であることに注意してください。

これまでの確認ができれば、DiningGuide アプリケーションの Web サービス が正しく作成されていることになります。第 5 章では、提供されている Swing クライアントを使用して、DiningGuide アプリケーションを実行します。

他の開発者が Web サービスを利用できるようにする

前節では、Web サービスの開発者にとって Web サービスをテストする便利な方法を紹介しました。しかし、所属する組織の他の開発グループ (特にクライアントの開発者) も、Web サービスに対して自分たちの作業内容をテストする必要があります。作成した Web サービス の WSDL ファイルを、他の開発者に簡単に提供することができます。他の開発者は、このファイルから、アプリケーションのクライアントの作成に使用可能なクライアントプロキシを生成し、Web サービスに対して自分が作成したクライアントをテストすることができます。他の開発者が Web サービスを利用するためには、配備されている Web サービスの URL が分かっていて、Web サーバーが動作している必要があります。

他の開発者が Web サービスを利用できるようにするには、次の作業を行います。

1. Web サービスの開発者の作業
   - Web サービスから WSDL ファイルを作成します。
   - Web サービスのユーザーが WSDL ファイルを利用できるようにします。
   - Web サービスのユーザーに配備した Web サービスの URL を教えます。
2. クライアントの開発者 (Web サービスのユーザー) の作業
   - エクスプローラでマウントしたファイルシステムに WSDL ファイルを追加します。
   - この WSDL から Web サービスのクライアントを作成します。
   - クライアントプロキシを生成します。
   - クライアントプロキシを基にクライアントを構築します。
   - クライアントプロキシの SOAP RPC の URL プロパティとして、Web サービスの URL を指定します。

クライアントプロキシを生成すると、アプリケーションの実際のクライアントの開発に必要な JSP ページが生成されます。

WSDL ファイルの生成

アプリケーションの Web サービスを共用できるようにするための第一歩は、Web サービスの WSDL ファイルを生成することです。この作業は、Web サービスの開発者が行います。

Web サービスの WSDL ファイルを生成するには、次の操作を行います。

1. エクスプローラで 「`DiningGuideWebService`」ノード () を右クリックして、「WSDL を生成」を選択します。

   「`WebService`」パッケージの下に「`DiningGuideWebService`」という名前の WSDL ファイル (緑色の球の入ったアイコンのノード ) が作成されます。

   このファイル (`DiningGuideWebService.wsdl`) は使用しているオペレーティングシステムのファイルシステム上で確認することができます。

2. 他の開発チームからこのファイルを利用できるようにします。

   電子メールにファイルを添付するか、Web サイトで配布してください。

WSDL ファイルからクライアントプロキシを作成

アプリケーションの Web サービスを共用できるようにするための第 2 段階の作業は、Web サービスサポート用のすべてのファイルを WSDL ファイルから生成することです。この作業は、クライアントの開発者が行います。

WSDL ファイルから Web サービスファイルとクライアントプロキシを生成するには、次の操作を行います。

1. 使用しているオペレーティングシステムのファイルシステムでディレクトリを作成して、「`DiningGuideWebService.wsdl`」ファイルをそのディレクトリに入れます。

2. Forte for Java 4 IDE で「ファイル」->「ファイルシステムをマウント」を選択します。

   新規ウィザードが表示されます。

3. 「ローカルディレクトリ」を選択して「次へ」をクリックします。

   新規ウィザードに「ディレクトリを選択」区画が表示されます。

4. 手順 1 で作成したディレクトリを探し、選択して「完了」をクリックします。

   エクスプローラにディレクトリがマウントされます。

5. マウントされたディレクトリを右クリックして、「新規」->「Java パッケージ」を選択します。

6. 「名前」フィールドに `MyClientPackage` と入力し、「完了」をクリックします。

   ディレクトリ内に `MyClientPackage` が表示されます。

7. エクスプローラで「新規」->「Web サービス」->「Web サービスクライアント」を選択します。

   新規ウィザードが表示されます。

8. 「名前」フィールドに `NewClient` と入力します。

9. 「パッケージ」フィールドが `MyClientPackage` になっていることを確認します。

10. ソースとして「ローカル WSDL ファイル」オプションを選択して、「次へ」をクリックします。

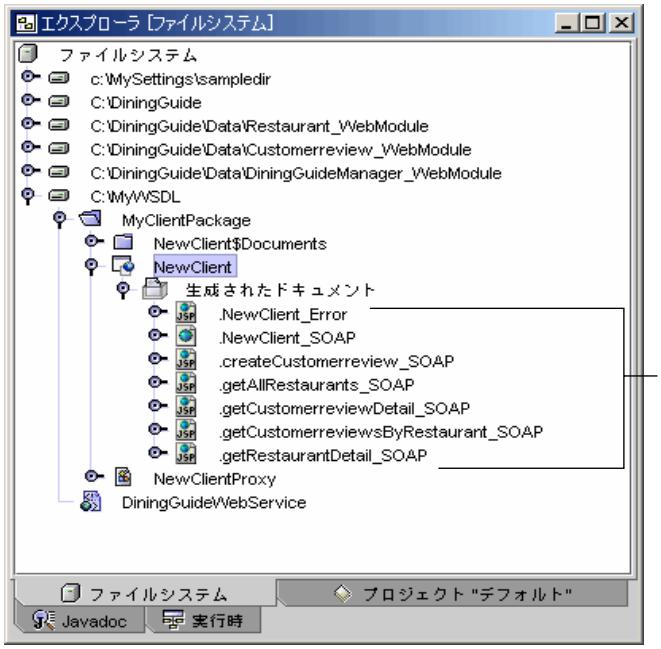
    新規ウィザードに「ローカル WSDL ファイルを選択」区画が表示されます。

11. マウントしたディレクトリの下の「`MyClientPackage`」パッケージ内の「`DiningGuideWebService`」WSDL ファイルを選択します。

    エクスプローラにクライアントノード ( ) が表示されます。

12. 「NewClient」クライアントノードを右クリックして、「クライアントプロキシを生成」を選択します。

エクスプローラに「生成されたドキュメント」ノードと「MyClientProxy」Bean が作成されます。開いた状態の「生成されたドキュメント」ノードに、クライアントに必要な JSP ページと開始ページが含まれていることが分かります。



これで、クライアントを使用して、Web サービスをテストすることができます (101 ページの「テストアプリケーションを使用した Web サービスのテスト」を参照)。

たいていの場合は、アプリケーションを完成したら、UDDI レジストリに Web サービスを公開し、外部の開発者がそのサービスを利用できるようにします。Forte for Java 4 には、このテストを行うためのシングルユーザー内部 UDDI レジストリと、StockApp という具体例が用意されており、この具体例は、Forte for Java 4 ポータルサイトの Examples and Tutorials ページ (http://forte.sun.com/ffj/documentation/tutorialsandexamples.html) から入手することができます。外部 UDDI レジストリへの公開については、『Web サービスのプログラミング』を参照してください。

第5章

チュートリアルアプリケーションのクライアントの作成

このチュートリアルには、第 4 章 で作成した Web サービスと通信する Swing クライアントが用意されています。この章では、その Swing クライアントを使用して、DiningGuide アプリケーションを実行する方法を学びます。

提供のクライアントには、「`RestaurantTable`」と「`CustomerReviewTable`」という 2 つの Swing クラスが含まれており、それらクラスのコードは付録 A に記載しています。この章では、まず 2 つのクラスを作成し、そのデフォルトのコードから付録 A のコードに、コピー & ペーストで置き換えます。 そして `RestaurantTable` クラスを実行することによってアプリケーションを実行します。

なお、このクライアントは非常に原始的なもので、Web サービス用に生成したクライアントプロキシのメソッドのアクセス方法を具体的に理解していただく目的に用意されています。

この章の内容は以下のとおりです。

提供コードを使用したクライアントの作成

提供コードを使用してクライアントを作成するには、2 つのクラスを作成して、そのコード全体を付録 A で提供しているソースコードに置き換えます。置き換えたクラスのコードは、クライアントプロキシをインスタンス化します。このプロキシは同じパッケージ内にあると想定するためクライアントクラスは `WebService` パッケージ内に作成する必要があります。

2 つのクラスを作成するには、次の操作を行います。

1. エクスプローラで 「`WebService`」ノードを右クリックして、「新規」-> 「Classes」->「Class」を選択します。

   新規ウィザードが表示されます。

2. このクラスに `RestaurantTable` という名前を付けて、「完了」をクリックします。

   「`WebService`」パッケージの下に「`RestaurantTable`」が作成されます。

3. 手順 1 と 手順 2 を繰り返して「`CustomerreviewTable`」クラスを作成します。

4. ソースエディタで「クラス」を開き、それぞれのクラスのデフォルトのコードをすべて削除します。

5. 138 ページから 4 ページにまたがって記載されている 「RestaurantTable.java のソース」のすべてのコードを `RestaurantTable` クラスの本体に入力します。

   参照 - この長いコードのコピーは慎重に行ってください。一度に 1 ページ全体を表示するように Acrobat Reader を設定します。`RestaurantTable` の最初のページのコード全体を選択してコピーし、ソースエディタの目的のファイルにペーストします。ペーストしたコードの最後で Enter を押して改行します。同様にして `RestaurantTable` の次のページのコード全体を、ソースエディタの改行で作成した新しい行にコピーします。すべてのコードをコピーし終えるまで、このコピー & ペースト操作を繰り返します。

   ---

6. ソースエディタでペーストしたコード全体を選択し、Control+Shift F を押すと

   コード全体の書式が整えられます。

7. 142 ページから 4 ページにまたがって記載されている 「CustomerReviewTable.java のソース」についても、手順 5 を繰り返すことによって、コード全体を `CustomerReviewTable` クラスの本体にコピーします。
8. 手順 6 を繰り返して、ペーストしたコードの書式を整えます。
9. 「`CustomerReviewTable`」クラスのノードを右クリックして、「コンパイル」を選択します。

   `CustomerReviewTable` クラスがコンパイルされます。
10. 「`RestaurantTable`」クラスのノードを右クリックして、「コンパイル」を選択します。

    `RestaurantTable` クラスがコンパイルされます。

    `RestaurantTable` と `CustomerReviewTable` コードを見ると、いくつかのセクションで、そのコードを変更しないよう警告するコメントがいくつかあることが分かります。それらのセクションはフォームエディタで作成された Swing コンポーネントコードにあたる部分で、ソースエディタで変更してはいけません。通常、そうした制限のあるコードの背景は青色になります。IDE を再起動すると、そうしたファイルのソースの制限がある部分の背景が青色で表示され、コードを編集できなくなります。

    IDE のフォームエディタで Swing クライアントを作成すると、`.form` ファイルと `.java` ファイルが 1 つずつ作成されます。このうちの `.form` ファイルでは、フォームエディタで GUI コンポーネントを編集することができます。しかし、`RestaurantTable` クラスと `CustomerReviewTable` クラスには `.form` ファイルはありません。このため、フォームエディタでそれらのクラスの GUI コンポーネントを変更することはできません。

チュートリアルアプリケーションの実行

DiningGuide アプリケーションを実行するには、以下の手順で `RestaurantTable` クラスを実行します。

1. IDE のエクスプローラの「実行時」タブをクリックします。

2. 「サーバーレジストリ」->「インストールされているサーバー」->「J2EE リファレンス実装 1.3.1 」ノードを順に開きます。

   すでに Web サービス を配備しているため (99 ページの「テストアプリケーションの配備」を参照)、再配備する必要はありません。 ただし、Web サービスを配備したサーバーは動作している必要があります。手順 3 では、そのことを確認します。

3. 「RI インスタンス 1」ノードを右クリックし、「サーバーを起動」コマンドを選択します (コマンドが選択可能な場合)。

   コマンドが選択不可表示になっている場合は、J2EE RI サーバーがすでに動作していますから、何もする必要はありません。

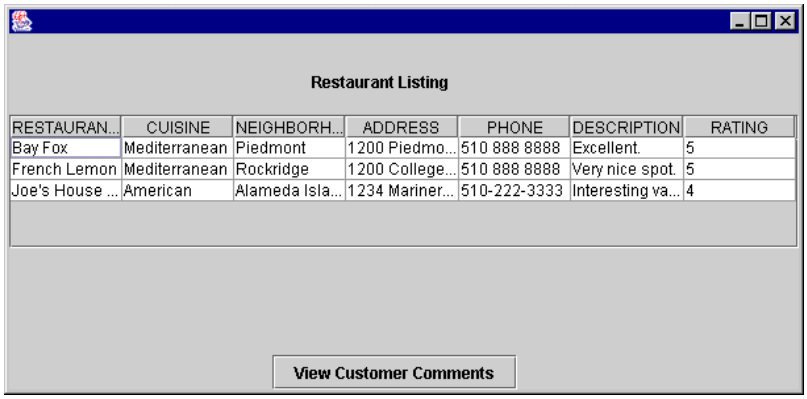
   PointBase サーバーも動作している必要があります。PointBase サーバーがすでに動作している場合は、手順 4 を飛ばして、手順 5 に進みます。

4. PointBase Network Server が動作していない場合は、「ツール」->「PointBase ネットワークサーバー」->「サーバーを起動」を選択します。

5. エクスプローラの「ファイルシステム」タブを選択します。

6. 「RestaurantTable」ノードを右クリックして、「実行」を選択します。

   IDE が実行時モードに切り替わります。実行ウィンドウに Restaurant ノードが現れ、以下のような「RestaurantTable」ウィンドウが表示されます。



7. 表のレストランをどれか選択して、「View Customer Comments」ボタンをクリックします。

   たとえば Bay Fox レストランを選択してください。「`CustomerReviewTable`」ウィンドウが表示されます。データベースにそのレストランに関するコメントが存在する場合は、以下のように表示されます。存在しない場合は、空の表が表示されます。

8. 「`Customer Name`」フィールドと「`Review`」フィールドに何か入力を行って、「Submit Customer Review」ボタンをクリックします。

   例:

レコードがデータベースに登録され、同じ「CustomerReviewTable」ウィンドウに表示されます。

9. 17 ページの「ユーザーから見たチュートリアルアプリケーション」で説明している機能を一通り試してみます。
10. どれかウィンドウを閉じることによってアプリケーションを終了します。

    アプリケーションを終了すると、「実行」ウィンドウに J2EE RI サーバープロセスがまだ動作中であることが示されます。J2EE RI サーバーを停止する必要はありません。チュートリアル用アプリケーションの J2EE アプリケーションをどれか再配備するか、この Swing クライアント以外のテストクライアントを再実行すると、この RI サーバーが自動的に再起動されます。

    IDE を終了すると、J2EE RI サーバーや Tomcat サーバーなどの動作中のプロセスを終了するためのダイアログが表示されます。動作中の各プロセスについて、プロセスを選択して、「タスクを終了」ボタンをクリックしてください。IDE の動作中にプロセスを手動で終了することもできます。このためには、「実行」ウィンドウでそのプロセスのノードを右クリックして、「プロセスを終了」を選択します。

クライアントコードの内容

DiningGuide アプリケーションに組み込んだ 2 つのクライアントクラスは、フォームエディタで作成された Swing コンポーネントおよびアクションと、ソースエディタで作成されたいくつかのメソッドから構成されます。ソースエディタで追加されたメソッドには、クライアントプロキシをインスタンス化して、クライアントからそのメソッドを利用できるようにする、重要なタスクが含まれています。

Swing クライアントと Web サービスがどのように対話するのかを理解するため、この節ではクライアントの主なアクションについて説明します。

- 「レストランデータの表示」 - 115 ページ
- 「選択されたレストランの利用者コメントデータの表示」 - 116 ページ
- 「新規利用者コメントレコードの作成」 - 119 ページ

レストランデータの表示

レストランデータの表示は、以下のようにクライアントプロキシをインスタンス化して、その `getAllRestaurants` メソッドを呼び出す、`RestaurantTable` クラスのメソッドによって行われます。

1. `RestaurantTable.getAllRestaurants` メソッドがクライアントプロキシをインスタンス化して、クライアントプロキシの `getAllRestaurants` メソッド (レストランデータをフェッチするメソッド) を呼び出し、フェッチされたレストランデータを 1 つの Vector として返します。

```
private Vector getAllRestaurants() {
    Vector restList = new Vector();
    try {
        DiningGuideWebServiceClientProxy restaurantCP = new
        DiningGuideWebServiceClientProxy();
        restList =
(java.util.Vector)restaurantCP.getAllRestaurants();
    }
    catch (Exception ex) {
        System.err.println("Caught an exception." );
        ex.printStackTrace();
    }
    return restList;
}
```

2. RestaurantTable コンストラクタが返されたレストランデータを restaurantList 変数に書き込み、RestaurantTable.putDataToTable を呼び出します。

```
public RestaurantTable() {
    initComponents();
    restaurantList=getAllRestaurants();
    putDataToTable();
}
```

3. RestaurantTable.putDataToTable メソッドは Vector を反復処理し、表を生成します。

```
private void putDataToTable() {
    Iterator j=restaurantList.iterator();
    while (j.hasNext()) {
        RestaurantDetail ci = (RestaurantDetail)j.next();
        String strRating = null;
        String[] str ={ci.getRestaurantname(), ci.getCuisine(),
ci.getNeighborhood(), ci.getAddress(), ci.getPhone(),
ci.getDescription(), strRating.valueOf(ci.getRating()),
        };
        TableModel.addRow(str);
    }
}
```

4. RestaurantTable.Main メソッドが Swing jTable コンポーネントとして表を表示します。

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

選択されたレストランの利用者コメントデータの表示

利用者コメントデータの表示は、RestaurantTable のボタンコンポーネントのアクションによって CustomerReviewTable がインスタンス化されることで始まります。CustomerReviewTable はクライアントプロキシのメソッドを利用して利用者

コメントデータをフェッチし、そのデータで表を生成します。そして RestaurantTable のボタンコンポーネントのアクションによって、その表が表示されます。

1. 利用者コメントデータを取得するために、RestaurantTable のボタンがクリックされると、RestaurantTable.jButton1ActionPerformed メソッドが CustomerReviewTable オブジェクトをインスタンス化して、putDataToTable メソッドを呼び出し、選択された列のデータを渡します。

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_jButton1ActionPerformed
    int r =jTable1.getSelectedRow();
    int c = jTable1.getSelectedColumnCount();
    String i =(String)TableModel.getValueAt(r,0);
    CustomerReviewTable crt = new CustomerReviewTable();
    crt.putDataToTable(i);
    crt.show();
    System.out.println(i);
}//GEN-LAST:event_jButton1ActionPerformed
```

2. CustomerReviewTable.putDataToTable メソッドが CustomerReviewTable.getCustomerReviewByName メソッドを呼び出して、選択されたレストラン名を渡し、返された Vector を customerList 変数に代入します。

```
public void putDataToTable(java.lang.String restaurantname) {
    RestaurantName = restaurantname;
    java.util.Vector customerList =
getCustomerReviewByName(restaurantname);
    Iterator j=customerList.iterator();
    while (j.hasNext()) {
        CustomerreviewDetail ci = (CustomerreviewDetail)j.next();
        String[] str = {ci.getCustomername(),ci.getReview()
        };
        TableModel.addRow(str);
    }
}
```

3. CustomerReviewTable.getCustomerReviewByName メソッドが、必要に応じてクライアントプロキシをインスタンス化し、その getCustomerreviewsByRestaurant メソッドを呼び出して、選択されたレストラン名を渡します。

```
private Vector getCustomerReviewByName(java.lang.String
restaurantname) {
    Vector custList = new Vector();
    try {
        DiningGuideWebServiceClientProxy custCP = new
        DiningGuideWebServiceClientProxy();
        custList =
(java.util.Vector)custCP.getCustomerreviewsByRestaurant(restaura
ntname);
    }
    catch (Exception ex) {
        System.err.println("Caught an exception." );
        ex.printStackTrace();
    }
    return custList;
}
```

4. コメントデータが CustomerReviewTable.putDataToTable メソッドに渡され、このメソッドによってデータが反復処理されて、そのデータで表が生成されます。

```
public void putDataToTable(java.lang.String restaurantname) {
    RestaurantName = restaurantname;
    java.util.Vector customerList =
getCustomerReviewByName(restaurantname);
    Iterator j=customerList.iterator();
    while (j.hasNext()) {
        CustomerreviewDetail ci = (CustomerreviewDetail)j.next();
        String[] str = {ci.getCustomername(),ci.getReview()
        };
        TableModel.addRow(str);
    }
}
```

5. RestaurantTable.jButton1ActionPerformed メソッドが表を表示します。

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {//GEN-FIRST:event_jButton1ActionPerformed
    int r =jTable1.getSelectedRow();
    int c = jTable1.getSelectedColumnCount();
    String i =(String)TableModel.getValueAt(r,0);
    CustomerReviewTable crt = new CustomerReviewTable();
    crt.putDataToTable(i);
    crt.show();
    System.out.println(i);
}//GEN-LAST:event_jButton1ActionPerformed
```

新規利用者コメントレコードの作成

「Customer Review」ウィンドウで名前とコメントを入力して、「Submit Customer Review」ボタンをクリックすると、以下のようにして、CustomerReviewTable の jButton1ActionPerformed メソッドが、プロキシのメソッドを利用してデータベースにそのコメントレコードを作成、「Customer Review」ウィンドウを再表示します。

1. CustomerReviewTable のボタンがクリックされて利用者コメントレコードが送信されると、CustomerReviewTable.jButton1ActionPerformed メソッドが必要に応じて新しいクライアントプロキシをインスタンス化して、その createCustomerreview メソッドを呼び出し、レストラン名と利用者名、コメントデータを渡します。

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
    try {
        DiningGuideWebServiceClientProxy reviewCP = new
        DiningGuideWebServiceClientProxy();
        reviewCP.createCustomerreview(RestaurantName,
customerNameField.getText(),reviewField.getText());
    }
    catch (Exception ex) {
        System.err.println("Caught an exception." );
        ex.printStackTrace();
    }
    refreshView();
}
```

2. 同じく jButton1ActionPerformed メソッドが
   CustomerReviewTable.refreshView メソッドを呼び出します。

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
    try {
        DiningGuideWebServiceClientProxy reviewCP = new
        DiningGuideWebServiceClientProxy();
        reviewCP.createCustomerreview(RestaurantName,
customerNameField.getText(),reviewField.getText());
    }
    catch (Exception ex) {
        System.err.println("Caught an exception." );
        ex.printStackTrace();
    }
    refreshView();
}
```

3. CustomerReviewTable.refreshView メソッドが putDataToTable メソッド
   を呼び出し、レストラン名を渡します。

```
void refreshView() {
    try{
        while(TableModel.getRowCount()>0) {
            TableModel.removeRow(0);
        }
        putDataToTable(RestaurantName);
        repaint();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

4. CustomerReviewTable.putDataToTable メソッドが渡されたデータで表を生成します。

```
public void putDataToTable(java.lang.String restaurantname) {
    RestaurantName = restaurantname;
    java.util.Vector customerList =
getCustomerReviewByName(restaurantname);
    Iterator j=customerList.iterator();
    while (j.hasNext()) {
        CustomerreviewDetail ci = (CustomerreviewDetail)j.next();
        String[] str = {ci.getCustomername(),ci.getReview()
        };
        TableModel.addRow(str);
    }
}
```

5. CustomerReviewTable.refreshView メソッドがウィンドウを再表示して、新しいデータを表示します。

```
void refreshView() {
    try{
        while(TableModel.getRowCount()>0) {
            TableModel.removeRow(0);
        }
        putDataToTable(RestaurantName);
        repaint();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

付録 A

DiningGuide のソースファイル

この付録では、次の DiningGuide コンポーネントのコードをまとめています。

- EJB 層のコンポーネント
  - `RestaurantBean.java` のソース - 124 ページ
  - `RestaurantDetail.java` のソース - 127 ページ
  - `CustomerreviewBean.java` のソース - 131 ページ
  - `CustomerreviewDetail.java` のソース - 133 ページ
  - `DiningGuideManagerBean.java` のソース - 135 ページ
- クライアントのコンポーネント
  - `RestaurantTable.java` のソース - 138 ページ
  - `CustomerReviewTable.java` のソース - 142 ページ

これらのソースファイルは、次の Forte for Java Developer Resources ポータルサイトからダウンロード可能な DiningGuide アプリケーションの zip ファイルにも含まれています。

`http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`

参照 – これらのファイルの内容を Forte for Java 4 のソースエディタにカット & ペーストすると、すべての書式設定が失われます。ソースエディタでコードの書式を整えるには、そのコードブロックを選択して、Control-Shift F を押してください。

RestaurantBean.java のソース

```
package Data;

import javax.ejb.*;

public abstract class RestaurantBean implements javax.ejb.EntityBean {

    private javax.ejb.EntityContext context;

    /**
     * @see javax.ejb.EntityBean#setEntityContext(javax.ejb.EntityContext)
     */
    public void setEntityContext(javax.ejb.EntityContext aContext) {
        context=aContext;
    }

    /**
     * @see javax.ejb.EntityBean#ejbActivate()
     */
    public void ejbActivate() {

    }

    /**
     * @see javax.ejb.EntityBean#ejbPassivate()
     */
    public void ejbPassivate() {

    }

    /**
     * @see javax.ejb.EntityBean#ejbRemove()
     */
    public void ejbRemove() {

    }
```

```
    /**
     * @see javax.ejb.EntityBean#unsetEntityContext()
     */
    public void unsetEntityContext() {
        context=null;
    }

    /**
     * @see javax.ejb.EntityBean#ejbLoad()
     */
    public void ejbLoad() {

    }

    /**
     * @see javax.ejb.EntityBean#ejbStore()
     */
    public void ejbStore() {

    }

    public abstract java.lang.String getRestaurantname();
    public abstract void setRestaurantname(java.lang.String restaurantname);

    public abstract java.lang.String getCuisine();
    public abstract void setCuisine(java.lang.String cuisine);

    public abstract java.lang.String getNeighborhood();
    public abstract void setNeighborhood(java.lang.String neighborhood);

    public abstract java.lang.String getAddress();
    public abstract void setAddress(java.lang.String address);

    public abstract java.lang.String getPhone();
    public abstract void setPhone(java.lang.String phone);

    public abstract java.lang.String getDescription();
    public abstract void setDescription(java.lang.String description);

    public abstract java.lang.Integer getRating();
    public abstract void setRating(java.lang.Integer rating);

    public java.lang.String ejbCreate(java.lang.String restaurantname,
java.lang.String cuisine, java.lang.String neighborhood, java.lang.String
address, java.lang.String phone, java.lang.String description,
java.lang.Integer rating) throws javax.ejb.CreateException {
```

```
        if (restaurantname == null) {
            throw new javax.ejb.CreateException("The restaurant name is
required.");
        }
        setRestaurantname(restaurantname);
        setCuisine(cuisine);
        setNeighborhood(neighborhood);
        setAddress(address);
        setPhone(phone);
        setDescription(description);
        setRating(rating);
        return null;
    }

   public void ejbPostCreate(java.lang.String restaurantname, java.lang.String
cuisine, java.lang.String neighborhood, java.lang.String address,
java.lang.String phone, java.lang.String description, java.lang.Integer rating)
throws javax.ejb.CreateException {
    }

    public Data.RestaurantDetail getRestaurantDetail() {
        return (new RestaurantDetail(getRestaurantname(),
        getCuisine(),getNeighborhood(), getAddress(), getPhone(),
        getDescription(), getRating()));
    }
}
```

RestaurantDetail.java のソース

```
package Data;

import java.beans.*;

public class RestaurantDetail extends Object implements java.io.Serializable {

    private static final String PROP_SAMPLE_PROPERTY = "SampleProperty";

    private String sampleProperty;

    private PropertyChangeSupport propertySupport;

    /** Holds value of property restaurantname. */
    private String restaurantname;

    /** Holds value of property cuisine. */
    private String cuisine;

    /** Holds value of property neighborhood. */
    private String neighborhood;

    /** Holds value of property address. */
    private String address;

    /** Holds value of property phone. */
    private String phone;

    /** Holds value of property description. */
    private String description;

    /** Holds value of property rating. */
    private Integer rating;

    /** Creates new RestaurantDetail */
    public RestaurantDetail() {
        propertySupport = new PropertyChangeSupport( this );
    }

    public RestaurantDetail(java.lang.String restaurantname, java.lang.String
cuisine, java.lang.String neighborhood, java.lang.String address,
java.lang.String phone, java.lang.String description, java.lang.Integer rating)
{
        System.out.println("Creating new RestaurantDetail");
```

```
        setRestaurantname(restaurantname);
        setCuisine(cuisine);
        setNeighborhood(neighborhood);
        setAddress(address);
        setPhone(phone);
        setDescription(description);
        setRating(rating);
    }

    public String getSampleProperty() {
        return sampleProperty;
    }

    public void setSampleProperty(String value) {
        String oldValue = sampleProperty;
        sampleProperty = value;
        propertySupport.firePropertyChange(PROP_SAMPLE_PROPERTY, oldValue,
sampleProperty);
    }

    public void addPropertyChangeListener(PropertyChangeListener listener) {
        propertySupport.addPropertyChangeListener(listener);
    }

    public void removePropertyChangeListener(PropertyChangeListener listener) {
        propertySupport.removePropertyChangeListener(listener);
    }

    /** Getter for property restaurantname.
     * @return Value of property restaurantname.
     */
    public String getRestaurantname() {
        return this.restaurantname;
    }

    /** Setter for property restaurantname.
     * @param restaurantname New value of property restaurantname.
     */
    public void setRestaurantname(String restaurantname) {
        this.restaurantname = restaurantname;
    }
```

```
    /** Getter for property cuisine.
     * @return Value of property cuisine.
     */
    public String getCuisine() {
        return this.cuisine;
    }

    /** Setter for property cuisine.
     * @param cuisine New value of property cuisine.
     */
    public void setCuisine(String cuisine) {
        this.cuisine = cuisine;
    }

    /** Getter for property neighborhood.
     * @return Value of property neighborhood.
     */
    public String getNeighborhood() {
        return this.neighborhood;
    }

    /** Setter for property neighborhood.
     * @param neighborhood New value of property neighborhood.
     */
    public void setNeighborhood(String neighborhood) {
        this.neighborhood = neighborhood;
    }

    /** Getter for property address.
     * @return Value of property address.
     */
    public String getAddress() {
        return this.address;
    }

    /** Setter for property address.
     * @param address New value of property address.
     */
    public void setAddress(String address) {
        this.address = address;
    }

    /** Getter for property phone.
     * @return Value of property phone.
     */
    public String getPhone() {
        return this.phone;
    }
```

```
    /** Setter for property phone.
     * @param phone New value of property phone.
     */
    public void setPhone(String phone) {
        this.phone = phone;
    }

    /** Getter for property description.
     * @return Value of property description.
     */
    public String getDescription() {
        return this.description;
    }

    /** Setter for property description.
     * @param description New value of property description.
     */
    public void setDescription(String description) {
        this.description = description;
    }

    /** Getter for property rating.
     * @return Value of property rating.
     */
    public Integer getRating() {
        return this.rating;
    }

    /** Setter for property rating.
     * @param rating New value of property rating.
     */
    public void setRating(Integer rating) {
        this.rating = rating;
    }

}
```

CustomerreviewBean.java のソース

```
package Data;

import javax.ejb.*;

public abstract class CustomerreviewBean implements javax.ejb.EntityBean {

    private javax.ejb.EntityContext context;


    /**
     * @see javax.ejb.EntityBean#setEntityContext(javax.ejb.EntityContext)
     */
    public void setEntityContext(javax.ejb.EntityContext aContext) {
        context=aContext;
    }

    /**
     * @see javax.ejb.EntityBean#ejbActivate()
     */
    public void ejbActivate() {

    }

    /**
     * @see javax.ejb.EntityBean#ejbPassivate()
     */
    public void ejbPassivate() {

    }

    /**
     * @see javax.ejb.EntityBean#ejbRemove()
     */
    public void ejbRemove() {

    }
```

```
    /**
     * @see javax.ejb.EntityBean#unsetEntityContext()
     */
    public void unsetEntityContext() {
        context=null;
    }

    /**
     * @see javax.ejb.EntityBean#ejbLoad()
     */
    public void ejbLoad() {

    }

    /**
     * @see javax.ejb.EntityBean#ejbStore()
     */
    public void ejbStore() {

    }

    public abstract java.lang.String getRestaurantname();
    public abstract void setRestaurantname(java.lang.String restaurantname);

    public abstract java.lang.String getCustomername();
    public abstract void setCustomername(java.lang.String customername);

    public abstract java.lang.String getReview();
    public abstract void setReview(java.lang.String review);

    public Data.CustomerreviewKey ejbCreate(java.lang.String restaurantname,
java.lang.String customername, java.lang.String review) throws
javax.ejb.CreateException {
        if ((restaurantname == null) || (customername == null)) {
            throw new javax.ejb.CreateException("Both the restaurant name and
customer name are required.");
        }
        setRestaurantname(restaurantname);
        setCustomername(customername);
        setReview(review);
        return null;
    }
```

```
    public void ejbPostCreate(java.lang.String restaurantname, java.lang.String
customername, java.lang.String review) throws javax.ejb.CreateException {
    }

    public Data.CustomerreviewDetail getCustomerreviewDetail() {
        return (new CustomerreviewDetail(getRestaurantname(),
        getCustomername(), getReview()));
    }
}
```

CustomerreviewDetail.java のソース

```
package Data;

import java.beans.*;

public class CustomerreviewDetail extends Object implements
java.io.Serializable {

    private static final String PROP_SAMPLE_PROPERTY = "SampleProperty";

    private String sampleProperty;

    private PropertyChangeSupport propertySupport;

    /** Holds value of property restaurantname. */
    private String restaurantname;

    /** Holds value of property customername. */
    private String customername;

    /** Holds value of property review. */
    private String review;

    /** Creates new CustomerreviewDetail */
    public CustomerreviewDetail() {
        propertySupport = new PropertyChangeSupport( this );
    }
```

```
    public CustomerreviewDetail(java.lang.String restaurantname,
java.lang.String customername, java.lang.String review) {
        System.out.println("Creating new CustomerreviewDetail");
        setRestaurantname(restaurantname);
        setCustomername(customername);
        setReview(review);
    }

    public String getSampleProperty() {
        return sampleProperty;
    }

    public void setSampleProperty(String value) {
        String oldValue = sampleProperty;
        sampleProperty = value;
        propertySupport.firePropertyChange(PROP_SAMPLE_PROPERTY, oldValue,
sampleProperty);
    }

    public void addPropertyChangeListener(PropertyChangeListener listener) {
        propertySupport.addPropertyChangeListener(listener);
    }

   public void removePropertyChangeListener(PropertyChangeListener listener) {
        propertySupport.removePropertyChangeListener(listener);
    }

    /** Getter for property restaurantname.
     * @return Value of property restaurantname.
     */
    public String getRestaurantname() {
        return this.restaurantname;
    }

    /** Setter for property restaurantname.
     * @param restaurantname New value of property restaurantname.
     */
    public void setRestaurantname(String restaurantname) {
        this.restaurantname = restaurantname;
    }

    /** Getter for property customername.
     * @return Value of property customername.
     */
    public String getCustomername() {
        return this.customername;
    }
```

```
    /** Setter for property customername.
     * @param customername New value of property customername.
     */
    public void setCustomername(String customername) {
        this.customername = customername;
    }

    /** Getter for property review.
     * @return Value of property review.
     */
    public String getReview() {
        return this.review;
    }

    /** Setter for property review.
     * @param review New value of property review.
     */
    public void setReview(String review) {
        this.review = review;
    }
}
```

DiningGuideManagerBean.java のソース

```
package Data;

import javax.ejb.*;
import javax.naming.*;

public class DiningGuideManagerBean implements javax.ejb.SessionBean {
    private javax.ejb.SessionContext context;
    private RestaurantHome myRestaurantHome;
    private CustomerreviewHome myCustomerreviewHome;

    /**
     * @see javax.ejb.SessionBean#setSessionContext(javax.ejb.SessionContext)
     */
    public void setSessionContext(javax.ejb.SessionContext aContext) {
        context=aContext;
    }

    /**
```

```
     * @see javax.ejb.SessionBean#ejbActivate()
     */
    public void ejbActivate() {

    }

    /**
     * @see javax.ejb.SessionBean#ejbPassivate()
     */
    public void ejbPassivate() {

    }

    /**
     * @see javax.ejb.SessionBean#ejbRemove()
     */
    public void ejbRemove() {

    }

    /**
     * See section 7.10.3 of the EJB 2.0 specification
     */
    public void ejbCreate() {
        System.out.println("Entering DiningGuideManagerEJB.ejbCreate()");
        Context c = null;
        Object result = null;
        if (this.myRestaurantHome == null) {
            try {
                c = new InitialContext();
                result = c.lookup("Restaurant");
                myRestaurantHome =
                (RestaurantHome)javax.rmi.PortableRemoteObject.narrow(result,
                RestaurantHome.class);
            }
            catch (Exception e) {System.out.println("Error: "+ e); }
        }
        Context crc = null;
        Object crresult = null;
        if (this.myCustomerreviewHome == null) {
            try {
                crc = new InitialContext();
                result = crc.lookup("Customerreview");
                myCustomerreviewHome =
             (CustomerreviewHome)javax.rmi.PortableRemoteObject.narrow(result,
                CustomerreviewHome.class);
            }
            catch (Exception e) {System.out.println("Error: "+ e); }
        }
```

```
    }

    public java.util.Vector getAllRestaurants() {
        System.out.println("Entering
DiningGuideManagerEJB.getAllRestaurants()");
        java.util.Vector restaurantList = new java.util.Vector();
        try {
            java.util.Collection rl = myRestaurantHome.findAll();
            if (rl == null) { restaurantList = null; }
            else {
                RestaurantDetail rd;
                java.util.Iterator rli = rl.iterator();
                while ( rli.hasNext() ) {
                    rd =((Restaurant)rli.next()).getRestaurantDetail();
                    System.out.println(rd.getRestaurantname());
                    System.out.println(rd.getRating());
                    restaurantList.addElement(rd);
                }
            }
        } catch (Exception e) {
            System.out.println("Error in
DiningGuideManagerEJB.getAllRestaurants(): " + e);
        }
       System.out.println("Leaving DiningGuideManagerEJB.getAllRestaurants()");
        return restaurantList;
    }

    public java.util.Vector getCustomerreviewsByRestaurant(java.lang.String
restaurantname) {
        System.out.println("Entering
DiningGuideManagerEJB.getCustomerreviewsByRestaurant()");
        java.util.Vector reviewList = new java.util.Vector();
        try {
            java.util.Collection rl =
            myCustomerreviewHome.findByRestaurantName(restaurantname);
            if (rl == null) { reviewList = null; }
            else {
                CustomerreviewDetail crd;
                java.util.Iterator rli = rl.iterator();
                while ( rli.hasNext() ) {
                    crd =
                    ((Customerreview)rli.next()).getCustomerreviewDetail();
                    System.out.println(crd.getRestaurantname());
                    System.out.println(crd.getCustomername());
                    System.out.println(crd.getReview());
                    reviewList.addElement(crd);
                }
            }
        } catch (Exception e) {
```

```
            System.out.println("Error in
DiningGuideManagerEJB.getCustomerreviewsByRestaurant(): " + e);
        }
        System.out.println("Leaving
DiningGuideManagerEJB.getCustomerreviewsByRestaurant()");
        return reviewList;
    }

    public void createCustomerreview(java.lang.String restaurantname,
java.lang.String customername, java.lang.String review) {
        System.out.println("Entering
DiningGuideManagerEJB.createCustomerreview()");
        try {
            Customerreview customerrev =
myCustomerreviewHome.create(restaurantname, customername, review);
        } catch (Exception e) {
            System.out.println("Error in
DiningGuideManagerEJB.createCustomerreview(): " + e);
        }
        System.out.println("Leaving
DiningGuideManagerEJB.createCustomerreview()");
    }

    public Data.RestaurantDetail getRestaurantDetail() {
        return null;
    }

    public Data.CustomerreviewDetail getCustomerreviewDetail() {
        return null;
    }
}
```

RestaurantTable.java のソース

```
package WebService;
import javax.swing.table.*;
import  java.util.*;
import Data.*;

public class RestaurantTable extends javax.swing.JFrame {

    /** Creates new form RestaurantTable */
```

```
    public RestaurantTable() {
        initComponents();
        restaurantList=getAllRestaurants();
        putDataToTable();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    private void initComponents() {//GEN-BEGIN:initComponents
        jButton1 = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTable1 = new javax.swing.JTable();
        jLabel1 = new javax.swing.JLabel();

        getContentPane().setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());

        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });

        jButton1.setText("View Customer Comments");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        getContentPane().add(jButton1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(200, 240, -1, -1));

        TableModel = (new javax.swing.table.DefaultTableModel (
            new Object [][] {

            },
            new String [] {
              "RESTAURANT NAME", "CUISINE", "NEIGHBORHOOD", "ADDRESS", "PHONE",
"DESCRIPTION", "RATING"
            }
        ) {
```

```
            Class[] types = new Class [] {
                java.lang.String.class, java.lang.String.class,
java.lang.String.class,
java.lang.String.class,java.lang.String.class,java.lang.String.class,java.lang
.String.class
            };

            public Class getColumnClass (int columnIndex) {
                return types [columnIndex];
            }
        });
        jTable1.setModel(TableModel);
        jScrollPane1.setViewportView(jTable1);

        getContentPane().add(jScrollPane1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(0, 60, 600, 100));

        jLabel1.setText("Restaurant Listing");
        getContentPane().add(jLabel1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(230, 20, 110, 30));

        pack();
    }//GEN-END:initComponents

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton1ActionPerformed
        int r =jTable1.getSelectedRow();
                int c = jTable1.getSelectedColumnCount();

                String i =(String)TableModel.getValueAt(r,0);
                CustomerReviewTable crt = new CustomerReviewTable();
                crt.putDataToTable(i);
                crt.show();
             System.out.println(i);
    }//GEN-LAST:event_jButton1ActionPerformed

    /** Exit the Application */
    private void exitForm(java.awt.event.WindowEvent evt)
{//GEN-FIRST:event_exitForm
        System.exit(0);
    }//GEN-LAST:event_exitForm

     private void putDataToTable()
    {
        Iterator j=restaurantList.iterator();
            while (j.hasNext()) {
                RestaurantDetail ci = (RestaurantDetail)j.next();
                String strRating = null;
```

```
                String[] str =
{ci.getRestaurantname(),ci.getCuisine(),ci.getNeighborhood(),ci.getAddress(),
ci.getPhone(),ci.getDescription(),
                                  strRating.valueOf(ci.getRating()),
                                 };
                TableModel.addRow(str);
            }
    }
      private Vector getAllRestaurants()
    {
         Vector restList = new Vector();
        try
        {
            DiningGuideWebServiceClientProxy restaurantCP = new
DiningGuideWebServiceClientProxy();
            restList = (java.util.Vector)restaurantCP.getAllRestaurants();
        }
        catch (Exception ex)
        {
            System.err.println("Caught an exception." );
            ex.printStackTrace();
        }
        return restList;
    }

       private Vector getCustomerreviewByRestaurant(java.lang.String
restaurantname)
    {
         Vector reviewList = new Vector();
        try
        {
            DiningGuideWebServiceClientProxy restaurantCP = new
DiningGuideWebServiceClientProxy();
            reviewList =
(java.util.Vector)restaurantCP.getCustomerreviewsByRestaurant(restaurantname);
        }
        catch (Exception ex)
        {
            System.err.println("Caught an exception." );
            ex.printStackTrace();
        }
        return reviewList;
    }
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        new RestaurantTable().show();
    }
```

```
    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButton1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTable jTable1;
    private javax.swing.JLabel jLabel1;
    // End of variables declaration//GEN-END:variables
    private DefaultTableModel TableModel;
    private java.util.Vector restaurantList = null;
}
```

CustomerReviewTable.java のソース

```
package WebService;
import javax.swing.table.*;
import  java.util.*;
import Data.*;

public class CustomerReviewTable extends javax.swing.JFrame {

    /** Creates new form CustomerReviewTable */
    public CustomerReviewTable() {
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    private void initComponents() {//GEN-BEGIN:initComponents
        jScrollPane1 = new javax.swing.JScrollPane();
        jTable1 = new javax.swing.JTable();
        jButton1 = new javax.swing.JButton();
        customerNameLabel = new javax.swing.JLabel();
        customerNameField = new javax.swing.JTextField();
        reviewLabel = new javax.swing.JLabel();
        reviewField = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
```

```
        getContentPane().setLayout(new
org.netbeans.lib.awtextra.AbsoluteLayout());

        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });

        TableModel = (new javax.swing.table.DefaultTableModel (
            new Object [][] {

            },
            new String [] {
                "CUSTOMER NAME", "REVIEW"
            }
        ) {
            Class[] types = new Class [] {
                java.lang.String.class,java.lang.String.class
            };

            public Class getColumnClass (int columnIndex) {
                return types [columnIndex];
            }
        });
        jTable1.setModel(TableModel);
        jScrollPane1.setViewportView(jTable1);

        getContentPane().add(jScrollPane1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(0, 60, 400, 100));

        jButton1.setText("Submit Customer Review");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        getContentPane().add(jButton1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(100, 250, 190, -1));

        customerNameLabel.setText("Customer Name");
        getContentPane().add(customerNameLabel, new
org.netbeans.lib.awtextra.AbsoluteConstraints(40, 170, -1, -1));

        getContentPane().add(customerNameField, new
org.netbeans.lib.awtextra.AbsoluteConstraints(153, 170, 170, -1));

        reviewLabel.setText("Review");
```

```
        getContentPane().add(reviewLabel, new
org.netbeans.lib.awtextra.AbsoluteConstraints(40, 200, 80, -1));

        getContentPane().add(reviewField, new
org.netbeans.lib.awtextra.AbsoluteConstraints(153, 200, 170, 20));

        jLabel1.setText("All Customer Review By Restaurant Name");
        getContentPane().add(jLabel1, new
org.netbeans.lib.awtextra.AbsoluteConstraints(80, 10, 240, -1));

        pack();
    }//GEN-END:initComponents

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton1ActionPerformed

        try {
            DiningGuideWebServiceClientProxy reviewCP = new
DiningGuideWebServiceClientProxy();
            reviewCP.createCustomerreview(RestaurantName,
customerNameField.getText(),reviewField.getText());
        }
        catch (Exception ex) {
            System.err.println("Caught an exception." );
            ex.printStackTrace();
        }

        refreshView();
    }//GEN-LAST:event_jButton1ActionPerformed
    void refreshView() {
        try{
            while(TableModel.getRowCount()>0) {
                TableModel.removeRow(0);
            }
            putDataToTable(RestaurantName);
            repaint();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    /** Exit the Application */
    private void exitForm(java.awt.event.WindowEvent evt)
{//GEN-FIRST:event_exitForm
        System.exit(0);
    }//GEN-LAST:event_exitForm
    public void putDataToTable(java.lang.String restaurantname) {
        RestaurantName = restaurantname;
       java.util.Vector customerList =getCustomerReviewByName(restaurantname);
```

```
        Iterator j=customerList.iterator();
        while (j.hasNext()) {
            CustomerreviewDetail ci = (CustomerreviewDetail)j.next();
            String[] str = {ci.getCustomername(),ci.getReview()

            };
            TableModel.addRow(str);
        }
    }
    private Vector getCustomerReviewByName(java.lang.String restaurantname) {
        Vector custList = new Vector();

        try {
            DiningGuideWebServiceClientProxy custCP = new
DiningGuideWebServiceClientProxy();
            custList =
(java.util.Vector)custCP.getCustomerreviewsByRestaurant(restaurantname);
        }
        catch (Exception ex) {
            System.err.println("Caught an exception." );
            ex.printStackTrace();
        }
        return custList;
    }
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        new CustomerReviewTable().show();
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JLabel reviewLabel;
    private javax.swing.JButton jButton1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextField customerNameField;
    private javax.swing.JTable jTable1;
    private javax.swing.JLabel customerNameLabel;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JTextField reviewField;
    // End of variables declaration//GEN-END:variables
    private DefaultTableModel TableModel;
    private java.lang.String RestaurantName = null;
    //private java.util.Vector restaurantList = null;
}
```

付録B

DiningGuide のデータベーススクリプト

DiningGuide チュートリアルの、PointBase データベースのスクリプトは次のとおりです。

```
drop table CustomerReview;
drop table Restaurant;

create table Restaurant(
    restaurantNamevarchar(80),
    cuisine varchar(25),
    neighborhoodvarchar(25),
    address varchar(30),
    phone   varchar(12),
    descriptionvarchar(200),
    rating  tinyint,
constraint pk_Restaurant primary key(restaurantName));

create table CustomerReview(
    restaurantNamevarchar(80) not null references Restaurant(restaurantName),
    customerName varchar(25),
    review  varchar(200),
constraint pk_CustomerReview primary key(customerName, restaurantName));

insert into Restaurant (restaurantName, cuisine, neighborhood, address, phone,
description, rating) values ('French Lemon','Mediterranean','Rockridge', '1200
College Avenue', '510 888 8888', 'Very nice spot.', 5);
insert into Restaurant (restaurantName, cuisine, neighborhood, address, phone,
description, rating) values
('Bay Fox','Mediterranean','Piedmont', '1200 Piedmont Avenue', '510 888 8888',
'Excellent.', 5);

insert into CustomerReview (restaurantName, customerName, review) values
('French Lemon','Fred', 'Nice flowers.');
```

```
insert into Customerreview (restaurantname, customername, review) values
('French Lemon','Ralph', 'Excellent service.');
```

索引

E

F

I

J

L

M

N

P

R

S

T

W

い

え

は

ひ

ふ

ほ

も

ゆ