Web サービスを使用するリッチ Java クライアントをビルドする

目次

Web サービスを使用するリッチ Java クライアントをビルドする

このチュートリアルでは、Java ビジュアル・エディター を使用して、Web サービスに接続するリッチ Java クライアントをビルドする方法について学習します。このチュートリアルでビルドするクライアントは、My Company Directory と呼ばれます。

My Company Directory は、会社の従業員登録簿を管理するための Java アプリケーションです。このアプリケーションは、従業員レコードの作成、検索、更新、および削除を行うためのメソッドを提供するサンプルの Web サービスに接続します。

クライアントは、Swing コンポーネントを使用して Java ビジュアル・エディターで視覚的にビルドされます。 Java ビジュアル・エディターは、Web サービスとの接続および連動を行うために、ヘルパー・クラス (データ・ソース、データ・オブジェクト、およびバインダー) のセットを提供します。 Web サービスは、ローカルでご使用のインストール済み IBM WebSphere Application Server v6.0 にデプロイされます。またツールは、Web サービス記述言語 (WSDL) ファイルをベースにしてクライアントのための Java プロキシーの生成を支援します。

製品を参照する

学習目標

このチュートリアルでは、以下のことを学習します。

- Java ビジュアル・エディターを使用した、ユーザー・インターフェースの設計およびレイアウト方法
- データ・オブジェクトおよび Web サービスに、インターフェース要素をバインドする方法

所要時間

2 時間 15 分

関連情報

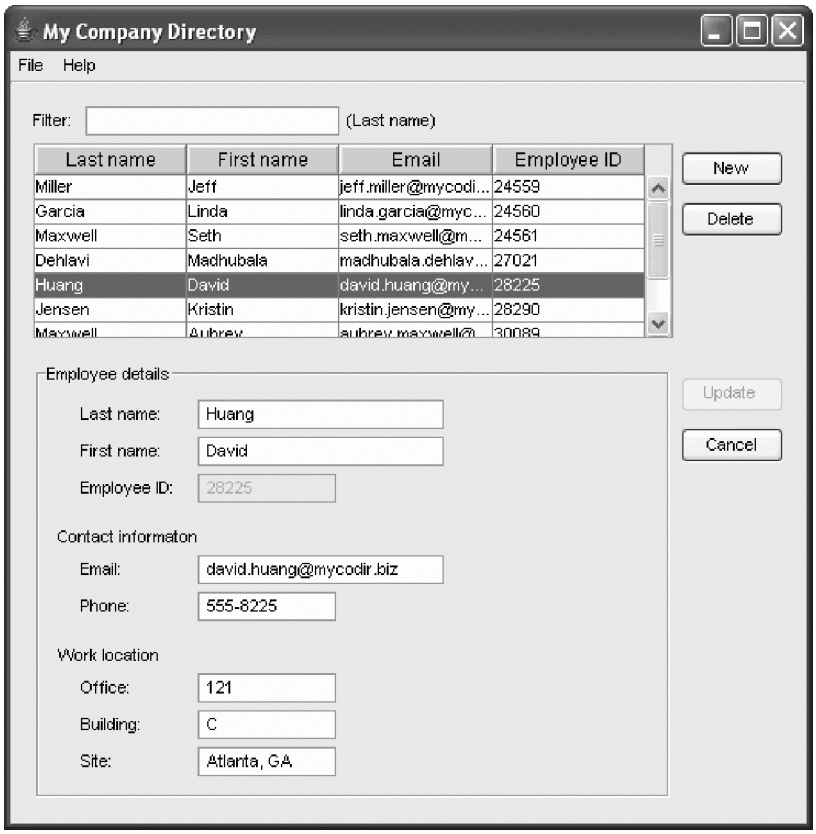
 PDF バージョンを表示する

チュートリアル: Hello World Java

概要: Web サービスを使用するリッチ Java クライアントをビルドする

クライアントのグラフィカル・ユーザー・インターフェース (GUI) は、見える部分には Swing コンポーネントを使用し、ほとんど作成済みとなっています。最初のモジュールでは、Java ビジュアル・エディターを使用して必須 GUI コンポーネントのレイアウトを完成します。 2 番目のモジュールでは、GUI コンポーネントを Web サービス・データ・ソース、サービス、およびデータ・ソースから戻されるオブジェクトにバインドします。 Java ビジュアル・エディターにアプリケーションをビルドするときは、データ・ソース、データ・オブジェクト、およびバインダーを使用します。これらは Java ビジュアル・エディターが生成し、このアプリケーションが使用するヘルパー・クラスのインスタンスです。

製品の画面図を参照する:



学習目標

このチュートリアルでは、以下のことを学習します。

- Java ビジュアル・エディターを使用した、ユーザー・インターフェースの設計およびレイアウト方法
- データ・オブジェクトおよび Web サービスに、インターフェース要素をバインドする方法

所要時間

このチュートリアルをすべて完了するための所要時間は約 2 時間 30 分です。

システム要件

- WebSphere Application Server v6.1。このサーバーは、ご使用の製品とともにインストール済みの場合がありますが、スタンドアロンとしてご自身でインストールしたものを使用することも可能です。このチュートリアルのシナリオでは、サンプル Web サービスをローカルで実行されている WebSphere Application Server にデプロイするように求められます。

サンプル Web サービスは、その他のサーバーでも実行できますが、このチュートリアルは WebSphere Application Server v6.0 と v6.1 でのみテストされています。

前提条件

このチュートリアルで学習される方は、次の概念に精通している必要があります。

- Java の基本的な開発
- Web サービスの基本原理

- ワークベンチの基本的なスキル (プロジェクトの作業、パースペクティブおよびビューのナビゲートなど)。

モジュール 1: ビジュアル・エディターでクライアント GUI を設計する

このモジュールでは、Java ビジュアル・エディターを使用して、ビジュアル・コンポーネントをアプリケーションに追加し、このコンポーネントを視覚的にレイアウトして、配置の制約を設定する方法について学習します。このモジュールの最後の演習では、Java ファイルを実行して、それが実際のアプリケーションとしてどのように見えるかを確認する方法を学習します。

要確認: このモジュールを始めるには、『チュートリアルの概要』で概略を示した前提知識をもっている必要があります。

学習目標

このモジュールの各演習完了後に、概念および次の作業の実行方法について理解していることが目標です。

- Java インターフェースに JTable を追加して、レイアウトする
- ビジュアル・クラスを実行して、行なった作業をテストする

所要時間

このモジュールを完了するには、約 15 分かかります。

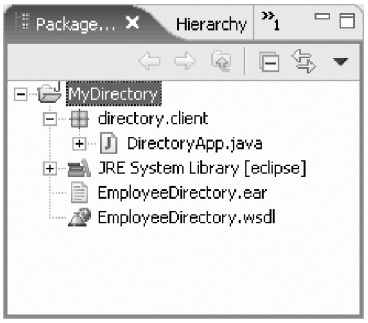
演習 1.1: Java プロジェクトをセットアップする

この演習では、プロジェクトをワークスペースにインポートして、MyDirectory プロジェクトをセットアップします。このプロジェクトには、1 個の Java クラス、および後で使用する他のファイルが含まれています。

ビジュアル・コンポーネントを Web サービスにバインドすることがこのチュートリアルの主な目的なので、「My Company Directory」アプリケーションのほとんどの Java GUI は、既に設計済みです。

MyDirectory プロジェクトは、このチュートリアルで扱う主要な Java プロジェクトです。これには、DirectoryApp.java ファイルが含まれています。この Java ファイルには、ビルドする主要な Java アプリケーションが入っています。このチュートリアルには、MyDirectory プロジェクトのいくつかのバージョンが組み込まれています。 1 つは、それぞれのモジュールを開始するためのバージョンであり、もう 1 つは完成したプロジェクトの完成バージョンです。

1. MyDirectory プロジェクトをインポートします。
2. Java パースペクティブのパッケージ・エクスプローラーで、MyDirectory プロジェクトが次のイメージのように表示されていることを確認します。



演習のチェックポイント

この演習では、このチュートリアルの開始点となる、サンプルの MyDirectory プロジェクトをインポートしました。

MyDirectory プロジェクトには、以下のリソースが含まれています。

- DirectoryApp.java: このチュートリアルで開発するアプリケーションが含まれている Java ファイル。DirectoryApp.java ファイルは、directory.client という名前の Java パッケージに入っています。
- EmployeeDirectory.ear: サンプル Web サービスが入っているエンタープライズ・アプリケーション。モジュール 2 では、ローカルにインストールされている WebSphere Application Server v6.0 に、この Web サービスをデプロイします。
- EmployeeDirectory.wsdl: Web サービス記述言語 (WSDL) を使用して、デプロイするサンプル Web サービスを記述するための XML ファイル。モジュール 2 では、この WSDL ファイルを使用して、アプリケーション用の Java プロキシーを生成します。

演習 1.2: 従業員テーブルを追加および配置する

この演習では、Java ビジュアル・エディターを使用して、JScrollPane と JTable をアプリケーションに追加します。この後に行う課題で、会社の登録簿にある全従業員のリストを戻す Web サービスからデータを取得するように、この JTable をプログラムします。

JTable を追加したならば、Java ビジュアル・エディターのデザイン・ビューを使用して、JTable のレイアウトを次の指定に一致するようにカスタマイズします。

- JTable を横方向に 3 セル、縦方向に 2 セル拡大する。
- 15 ピクセルの左側インセットを追加する。
- JTable を employeesTable に名前変更する。

デモを見る

Java ビジュアル・エディターで DirectoryApp.java ファイルを開く

Java ビジュアル・エディターで DirectoryApp.java ファイルを開くには、以下の手順を行います。

1. Java パースペクティブの「パッケージ・エクスプローラー」ビューを使用して、「MyDirectory」プロジェクトと「directory.client」パッケージを展開する。
2. 「DirectoryApp.java」ファイルを右クリックして、**「アプリケーションから開く」 → 「ビジュアル・エディター」**を選択する。 Java ビジュアル・エディターは、Java クラスをロードし、設計をグラフィック・キャンバス域に表示します。

ヒント:

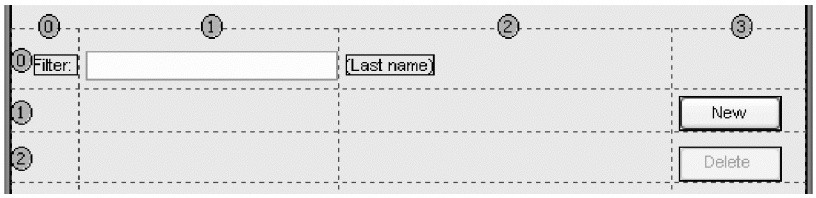
- Java ビジュアル・エディターが使用するルック・アンド・フィールを変更するには、**「ウィンドウ」 → 「設定」 → 「Java」 → 「ビジュアル・エディター」** の順に選択し、「Swing ルック・アンド・フィール」を指定する。この設定は、次にこのクラスを開いたときに有効になります。このチュートリアルは、Windows のルック・アンド・フィールを使用します。
- すべての Java ファイルについてこのビジュアル・エディターをデフォルト・エディターにするには、**「ウィンドウ」 → 「設定」**」をクリックして、**「一般」 → 「エディター」 → 「ファイルの関連付け」**ページに進み、設定を定義する。

JScrollPane 上の JTable を追加する

DirectoryApp.java のメイン・ウィンドウでは、メイン・コンテンツ・ペインに JFrame と JPanel を使用しています。このアプリケーションでの JPanel は jContentPane という名前です。 jContentPane は、GridBagLayout と呼ばれるレイアウト・マネージャーのタイプを使用するように設定されています。GridBagLayout は、ビジュアル・コンポーネントが占有できるセルのグリッドをベースにした強力なレイアウト方式です。 Java ビジュアル・エディターは、グリッド枠を表示することによって、GridBagLayout の作業を簡単にしています。このエディターは、また、グリッドに新しいコンポーネントをドロップするときに配置マーカーを表示し、さらに GridBagLayout 上でサイズ変更または移動を行うコンポーネント上でハンドルを表示します。

従業員のテーブル (javax.swing.JTable) を DirectoryApp.java ユーザー・インターフェースに追加するには、次のようにします。次の手順を実行します。

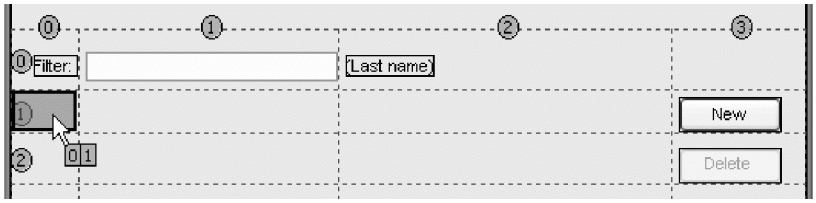
1. 「デザイン」ビューまたは 「Java Bean」ビューで、「jContentPane」を右クリックして、「**グリッドの表示**」を選択する。赤色の点線はグリッド枠を示し、数値がついた青色の丸印は行番号と列番号を示します。たとえば、「**New**」ボタンは、行 1 (Y 軸グリッド)、列 3 (X 軸グリッド) のセルにあります。



2. Java ビジュアル・エディター のパレットで、「**JScrollPane の JTable**」 Swing コンポーネントを選択する。このコンポーネントは、パレットの「**Swing コンポーネント**」ドロワーに分類されています。

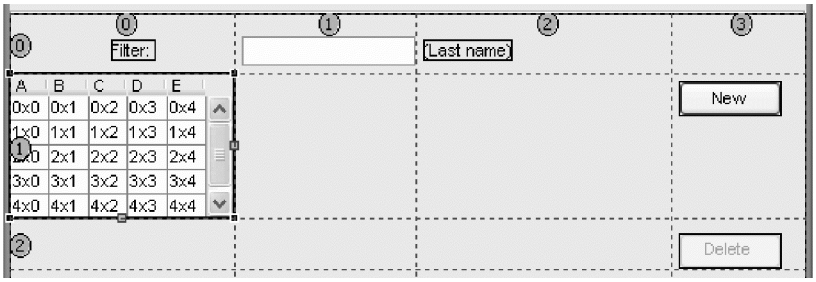
   ヒント: デフォルトでは、このパレットはデザイン域の右側に縮小表示されます。このパレットのサイズ変更したり、移動させたりすることができます。

3. マウス・ポインターを以下のようにグリッドの列 0、行 1 のセル上に移動する。



   - グリッド上でマウス・ポインターを移動しているときに、マウス・ポインターの位置に応じて、グリッドでの X 座標と Y 座標を知らせる 2 つの番号付きの四角をマウス・ポインターが示します。
   - マウス・ポインターをグリッド枠のすぐ上に置くと、新しい行と列を作成することができ、既存の行と列には再番号付けが行われます。この場合、マウス・ポインター上には黄色の四角が、グリッドとグリッドの間には黄色のバーが現れてこの動作を示し、黄色の列と行のラベルが新しい番号付けを示します。

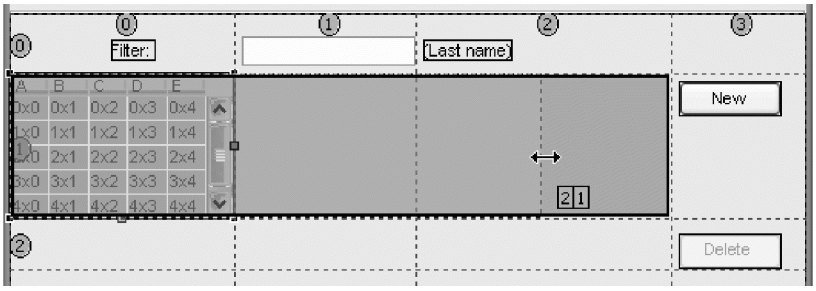
4. 列 0、行 1 のセルを左クリックして、JScrollPane と JTable をドロップする。



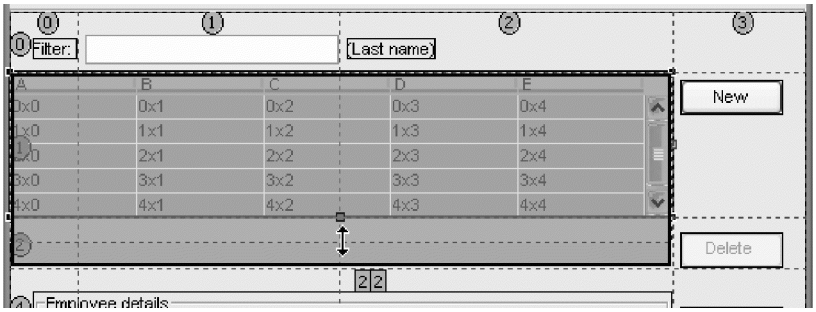
JScrollPane と JTable をグリッドの複数の列と行に拡大する

ここで、JScrollPane (および子の JTable) を 3 列 2 行に拡大して、スペーシングとサイズ変更動作を改善します。 テーブルを複数の列と行に拡大するには、次のようにします。

1. デザイン域または「Java Bean」ビューで、JScrollPane を選択する (これは追加したばかりなので、まだ選択されたままになっているはずです)。 JScrollPane の右と下に、小さな緑色の四角があることに注目してください。このサイズ変更ハンドルを使用し、JScrollPane をドラッグして、複数の列と行に拡大します。
2. JScrollPane の右側にある緑色のハンドルの上で左マウス・ボタンをクリックし、押したままにする。
3. マウス・ポインターを、その位置が列 2、行 1 を示すまで右方向にドラッグする。ダーク・グレーのシャドーも、マウス・ボタンを放したときにコンポーネントが占有するセルの位置を示しています。



4. マウス・ボタンを放す。これで、JScrollPane が 3 列に拡大されました。
5. 同様のプロセスを繰り返して、JScrollPane の下にあるハンドルをドラッグし、以下のように JScrollPane を 2 行に拡大する。



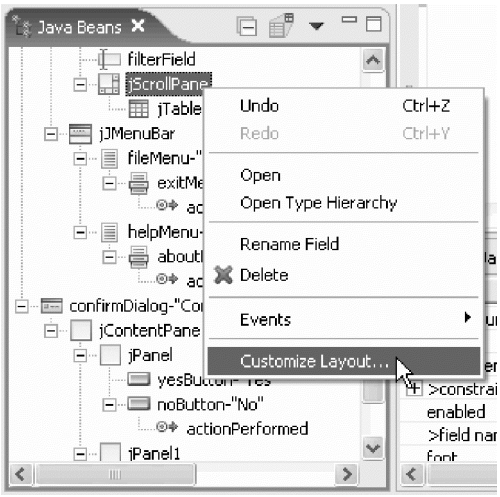
JScrollPane 内の余白を GridBag 内でカスタマイズする

GridBagLayout マネージャーのもう 1 つの機能として、さまざまな制約を指定して、レイアウトをさらにカスタマイズできるというものがあります。たとえば、以下の制約を指定できます。

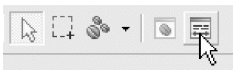
- **アンカー:** セルの中で、コンポーネントにアンカー方向を指定できる。これは、ユーザーがアプリケーションのサイズ変更をしたときのコンポーネントの移動方法に影響します。たとえば、コンポーネントのアンカー位置を左上、中央左、中央、または右下に設定できます。
- **塗りつぶし:** コンポーネントが、1 つまたは複数のセル内で、水平方向、垂直方向、あるいはその両方向で、使用可能な場所を占有するように指定できる。
- **インセット:** コンポーネントには、上、下、左、右に、固有のセル余白を与え、コンポーネントとグリッドの端の間に余白をおくことができる。

JScrollPane のアンカー、塗りつぶし、およびインセットをカスタマイズするには、次のようにします。

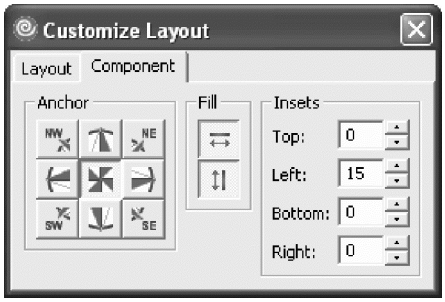
1. デザイン・ビューまたは「Java Bean」ビューで、「JScrollPane」を右クリックして、「**レイアウトのカスタマイズ**」を選択する。



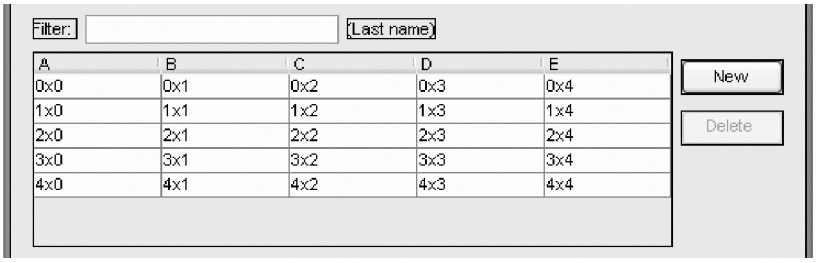
ヒント: さまざまなコンポーネントのレイアウトを選択して変更しているときに、「レイアウトのカスタマイズ」ダイアログ・ボックスを開いたままにしておくことができます。「レイアウトのカスタマイズ」ダイアログ・ボックスは、メニュー・バーの「レイアウトのカスタマイズ」ボタンをクリックして、いつでも開くことができます。



2. 「レイアウトのカスタマイズ」ダイアログ・ボックスにある「コンポーネント」タブの「アンカー」の中央のボタンが押されていることを確認する。
3. 「**水平に塗りつぶす**」ボタンと「**垂直に塗りつぶす**」ボタンの両方が押されていることを確認する。
4. アプリケーションにある他のビジュアル・コンポーネントと同様に、JScrollPane の左側に 15 ピクセルの左インセットを追加して余白を作る。



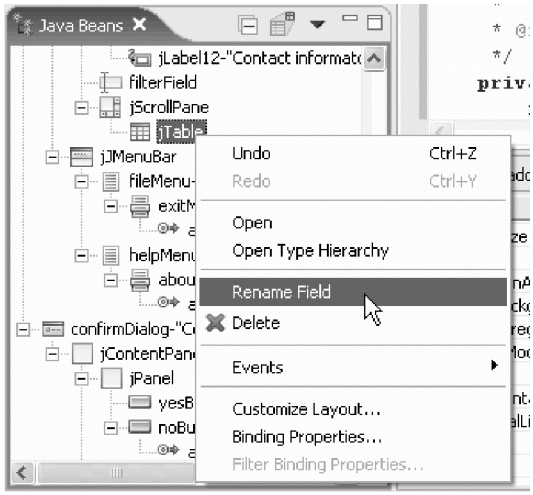
たとえば、テーブルは、この時点で「**Filter**」ラベルと位置合わせが行われています。



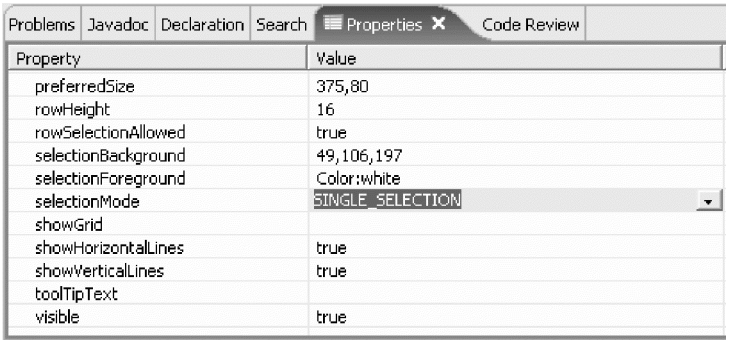
新規の JTable を便利な値に名前変更して、単一行を選択するように設定する

後でこのテーブルの作業をするので、JTable インスタンスとこのインスタンスの getter メソッドの名前を変更しておくと便利です。このテーブルの名前を変更するには、以下のようにします。

1. 「Java Bean」ビューで、「jTable」コンポーネントを右クリックし、ポップアップ・メニューから「**フィールド名の変更**」を選択する。



2. `employeesTable` と入力して、「**OK**」をクリックする。これで、JTable の名前が employeesTable となり、それをインスタンス化するメソッドの名前は getEmployeesTable となりました。
3. 単一行だけを選択できるよう、次のようにテーブルを設定する。
   a. デザイン・ビューで、「employeesTable」を選択する。
   b. 「プロパティー」ビューで、「**selectionMode**」プロパティーを選択し、それを「SINGLE_SELECTION」に設定する。



c. DirectoryApp.java ファイルを保管する。

演習のチェックポイント

この演習では、ビジュアル・エディターを使用して、既存のユーザー・インターフェースにテーブルを追加する方法を学習しました。次に、そのレイアウト、位置決め、および余白をカスタマイズする方法を学習しました。

演習 1.3: ビジュアル・クラスを実行する

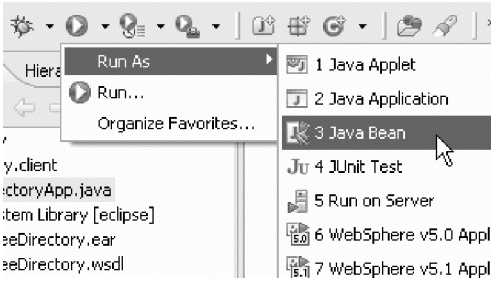
これで、Java アプリケーションを実行して外観をプレビューする準備が完了しました。ワークベンチとビジュアル・エディターを使用すると、対象のアプリケーションを簡単に素早く実行できます。また、開発中は、この手順をいつでも何回でも繰り返して、実際のランタイムの外観とクラスの動作を確認することができます。

デモを見る

Java ビジュアル・エディターには、main() メソッドがなくても、クラスの実行が可能な Java Bean ランチャーが備わっています。ビジュアル・クラスを実行する場合は、個別の仮想マシン (VM) でアプリケーションを起動します。ビジュアル・クラスを Java アプリケーションとして実行すると、ランチャーはクラス内の main() メソッドの実行を試行します。このチュートリアルでは、アプリケーションは、DirectoryApp JFrame の呼び出しと表示を行う main() メソッドをインクルードしているため、これをアプリケーションまたは Java Bean として実行できます。

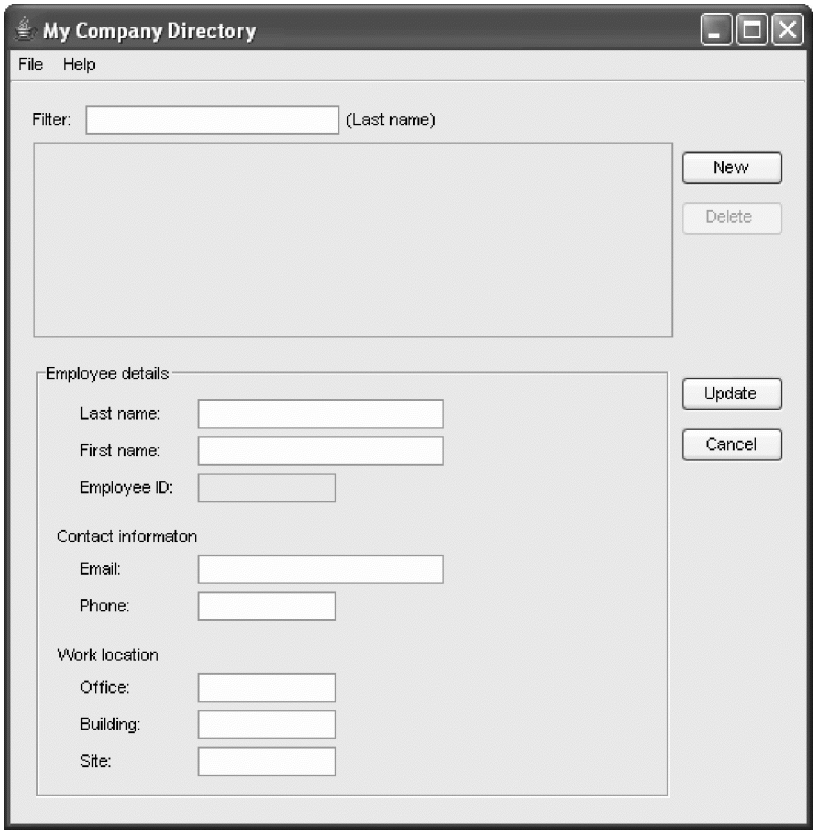
DirectoryApp.java ファイルを Java Bean として実行するには、次のようにします。

1. Java ビジュアル・エディター で DirectoryApp.java ファイルが開かれていることを確認する。
2. メニュー・バーから、「**実行**」 → 「**Java Bean**」をクリックする。



ヒント: アプリケーションは、ご使用のビジュアル・エディター設定 (「**ウィンドウ**」 → 「**設定**」 → 「**Java**」 → 「**ビジュアル・エディター**」」) に定義した Swing ルック・アンド・フィールを使用して、デスクトップ上で開きます。あるいはその代わりに、「**実行**」 → 「**構成および実行**」をクリックし、この Java Bean を起動するための固有の構成を行い、ルック・アンド・フィールを定義することも

可能です。このアプリケーションを、Bean ではなくアプリケーションとして実行する場合、これは main() メソッド内に定義されているため、Windows ルック・アンド・フィールも使用します。このチュートリアルで使用されたスクリーン・ショットは、Windows ルック・アンド・フィールを表示しています。



演習のチェックポイント

インターフェースの設計だけを行いましたが、まだ何のデータ接続またはイベント機能もプログラムしていないため、このアプリケーションでは何もできません。しかし、基本的なレイアウトや外観はユーザーが見る場合と同じように見えます。いくつかのボタンにクリックを試みることはできますが、その試行は何もしません。「ファイル」メニューと「ヘルプ」メニューは、すでにインプリメントされています。これらが行うことを知るために試行したり、Java コードを調べて actionPerformed イベントについてそれらがインプリメントされている方法を確認することができます。

演習での学習事項

このモジュールでは、Java ビジュアル・エディターを使用して、リッチ・クライアントのインターフェースを設計する方法を紹介しました。ただし、クライアントのビジュアル外観を設計する以外に、クライアントを実際に有用なものにするためには、さらにいろいろなことを行う必要があります。それについて、通常、イベント動作、またはその他のロジックを組み込んだり、さらに、この場合、ビジュアル要素をある種のデータ・ソースにバインドする必要があります。

このモジュールでは、以下のタスクを実行する方法を学習しました。

- プロジェクト交換インポートを使用して Java プロジェクトをインポートする。
- JScrollPane 上の JTable をビジュアル・クラスに追加する。

- GridBagLayout マネージャーを使用して、リッチ・クライアント上でテーブルを視覚的にレイアウトする。
- アプリケーションを実行して、リッチ Java クライアントの実際の外観を表示する。

次のモジュール「モジュール 2: ビジュアル・コンポーネントを Web サービスにバインドする」では、単純な My Company Directory インターフェースを、会社の登録簿にある従業員レコードを作成、検索、更新、削除するための Web サービス・メソッドにアクセスする、強力なリッチ・クライアントに変えます。

モジュール 2: ビジュアル・コンポーネントを Web サービスにバインドする

このモジュールでは、My Company Directory のビジュアル要素 (ボタン、従業員テーブル、フィールド、その他のアクション) を Web サービスにバインドする方法を学習します。 Web サービスは、実際に従業員をサンプルの登録簿から作成、検索、更新、および削除する機能を提供します。

学習目標

このモジュールの各演習完了後に、概念および次の作業の実行方法について理解していることが目標です。

- テーブルをデータ Web サービス・データ・ソースにバインドする
- フィールドをオブジェクトにバインドする
- アクションを使用して、ボタンをプログラムする

このモジュールを完了するには、約 **2 時間**要します。

演習 2.1: Web サービスをインストールおよびデプロイする

この演習では、サンプルのエンタープライズ・アプリケーション (EAR) ファイルを WebSphere Application Server v6.1 にインストールし、EmployeeDirectory Web サービスをデプロイします。ユーザーのアプリケーションは、この Web サービスを使用して、従業員レコードの作成、読み取り、更新、削除を行います。

課題を始める前に、*次のオプションのうちの 1 つ* を完了していて、この MyDirectory プロジェクトを開始することが適切であることを確認してください。

- 3 ページの『モジュール 1: ビジュアル・エディターでクライアント GUI を設計する』を完了する。

  または

- モジュール 2 の開始点で MyDirectory プロジェクトをインポートする

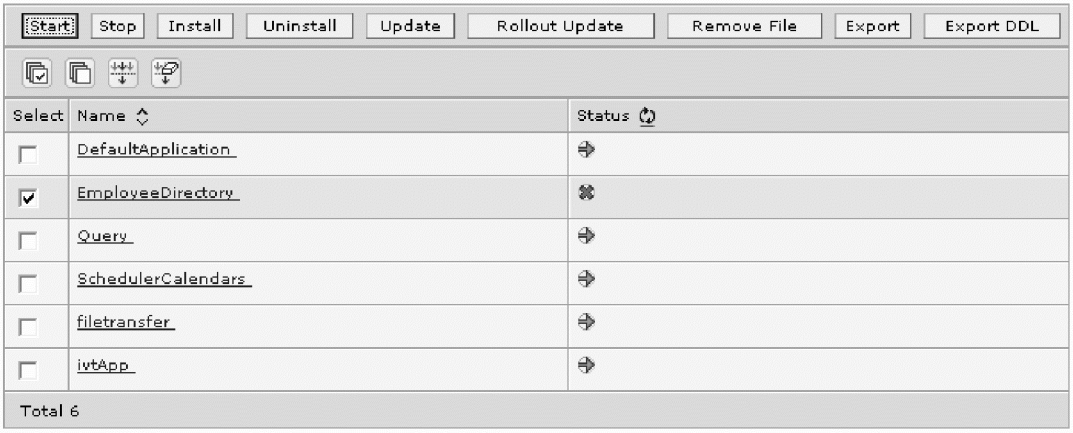
  ヒント: インポートをする際に別のプロジェクト名を指定しない限り、MyDirectory プロジェクト・コンテンツは上書きされます。

MyDirectory Java プロジェクトには、EmployeeDirectory.ear ファイルが組み込まれています。 WebSphere 管理コンソールを使用して、EAR ファイルに入っている EmployeeDirectory エンタープライズ・アプリケーションをインストールします。アプリケーションをインストールするときに、アプリケーションに組み込まれている Web サービスもデプロイします。完成した My Company Directory アプリケーションは、このデプロイされた Web サービスを使用します。

WebSphere Application Server v6.1 環境にサンプルの EmployeeDirectory アプリケーションをインストールし、Web サービスをデプロイするには、次のようにします。

1. アプリケーション・サーバーのインスタンスをワークベンチから起動する。ユーザーのサーバーを起動するにはさまざまな方法がありますが、以下の手順では、ワークベンチからどのように起動するかを説明します。
   a. 「サーバー」ビューを開く。 Java パースペクティブに「サーバー」ビューを追加するには、「**ウィンドウ**」 → 「**ビューの表示**」 → 「**その他**」 → 「**サーバー**」 → 「**サーバー**」と選択します。
   b. インストール済みかつ設定済みのサーバーが「サーバー」ビューにリストされる。
   c. サーバーを右クリックし、「**始動**」を選択する。「サーバー」ビューがそのサーバーの状況を「**始動済み**」と表示した場合、またはコンソールが「e-business のためにサーバー server1 がオープンされました」と表示した場合、サーバーは正常に開始しています。これで、管理コンソールを実行できるようになります。

   注: サーバー・インスタンスが「サーバー」ビューにない場合、次のようにして新規のサーバーを作成します。
   a. 「サーバー」ビュー内で右クリックし、「**新規**」 → 「**サーバー**」を選択する。
   b. 「新規サーバー」ウィザードを使用して、WebSphere Application Server v6.1 を追加する。
2. WebSphere 管理コンソールを実行する。管理コンソールを実行するための別の方法がありますが、以下の手順では管理コンソールをワークベンチから実行する方法を説明します。
   a. 「サーバー」ビューで、先程開始したばかりのサーバーを選択し、「**管理コンソールの実行**」を選択する。 WebSphere 管理コンソールがブラウザー・ウィンドウに開きます。
   b. ユーザー ID を入力し、「**ログイン**」をクリックする。管理コンソールの「ウェルカム」ページが開きます。入力したユーザー ID は、サーバーの構成データに加えたユーザー固有の変更をトラッキングするためだけに使用されます。
3. 管理コンソールを使用して、MyDirectory プロジェクト内にある EmployeeDirectory.ear エンタープライズ・アプリケーションをインストールする。管理コンソールは、ウィザードの方法を使用して、ユーザーがアプリケーションをインストールするのを支援します。このウィザードでは、すべてのオプションが設定されるまで「**次へ**」をクリックしてページからページへ移動します。このチュートリアル用の Web サービスが入っているサンプルのエンタープライズ・アプリケーションをインストールするには、次のようにします。
   a. 管理コンソールの左側で、「**アプリケーション**」メニュー・オプションを展開し、「**新規アプリケーションのインストール**」をクリックする。
   b. 「**ローカル・ファイル・システム**」を選択し、「**パスの指定**」フィールドで、MyDirectory プロジェクトにある EmployeeDirectory.ear ファイルまでの絶対パスを入力する。ヒント: 絶対パスを取得するには、パッケージ・エクスプローラー内の EmployeeDirectory.ear ファイルを右クリックし、「**プロパティー**」を選択します。「プロパティー」ページには、このファイルのロケーションがリストされ、そのファイルは「**パスの指定**」フィールドにコピーおよび貼り付けすることができます。
   c. 「**インストール・オプションの選択**」ページが表示されるまで、「**次へ**」をクリックする。
   d. 「**Web サービスのデプロイ**」を選択する。
   e. 「**要約**」ページが表示されるまで「**次へ**」をクリックし、「**終了**」をクリックする。
   f. ローカル構成に加えた変更を適用するよう指示するプロンプトが出されたならば、「**マスター構成に保管**」リンクをクリックする。変更を確認し、「**保管**」ボタンをクリックします。
4. 管理コンソールを使用して EmployeeDirectory アプリケーションを以下のようにして開始する。
   a. 「**アプリケーション**」 → 「**エンタープライズ・アプリケーション**」をクリックする。
   EmployeeDirectory アプリケーションがサーバー上にインストール済みアプリケーションとしてリストされますが、その状況は「停止」となっています。



b. EmployeeDirectory の隣にあるチェック・ボックスを選択し、「**始動**」をクリックする。EmployeeDirectory アプリケーションが正常に開始されたことを示すメッセージが出され、「状況」アイコンが緑の矢印に変わります。

これで、EmployeeDirectory アプリケーションが localhost のポート 9080 で実行され、Web サービスにアクセスできる状態になりました。このチュートリアルの完了後、管理コンソールに戻って、EmployeeDirectory アプリケーションを停止してから、このアプリケーションをアンインストールできます。

対象の MyDirectory プロジェクトにある EmployeeDirectory.wsdl ファイル (デフォルトでは、グラフィカル WSDL エディター内で開く) を開くと、デプロイしたばかりの Web サービスを調べることができます。 WSDL ファイルが WSDL エディターで開かない場合は、Web サービス・デベロッパー機能がワークベンチ内で有効になっていない可能性があります。「設定」でワークベンチ機能を指定することができます (**「ウィンドウ」** → **「設定」** → **「一般」** → **「機能」**)。

次の WSDL エディターのイメージは、EmployeeDirectory サービス内で使用可能な操作を示しています。

WSDL エディターを使用して、各操作およびその対応する要求メッセージおよび戻りメッセージを調べることができます。これは、Web サービスを理解すること、および残りの課題で Web サービスが使用される方法を理解することに役立ちます。

演習 2.2: Web サービス・データ・ソースに従業員テーブルをバインドする

My Company Directory アプリケーションは、登録簿にある現在の全従業員のレコードをリスト表示します。このレコードはソート可能な列を持つ JTable (employeesTable) に表示されます。ソート可能な列には、ラストネーム、ファーストネーム、E メール、従業員 ID が含まれます。テーブルのレコードを取得するには、サンプル Web サービスのデータ・ソース から戻されたデータ・オブジェクト に、employeesTable をバインド する必要があります。

デモを見る

データ・オブジェクト、データ・ソース、バインダーの概要

作業対象の employeesTable に関するローカルのデータ・オブジェクトを取得するには、ビジュアル・エディターを使用し、データ・ソース をアプリケーションに追加します。データ・ソースは、サンプルの Web サービス・プロキシーに接続し、アプリケーションに使用可能なサービス・メソッドを見つけます。次に、このデータ・ソースから使用可能になる getLightEmployeeRecord サービスを選択します。最後に、行データ・オブジェクト (lightEmployeeRecordRows) に戻されたフィールドに、アプリケーションの employeesTable をバインド します。

Java ビジュアル・エディターに組み込まれたバインダー・クラスを使用して、これらデータ・ソースおよびデータ・オブジェクトのすべてを素早く作成することができます。ビジュアル・エディターは、ユーザーがビジュアル・コンポーネントをデータ・ファクトリーにバインドする際に、ユーザーのプロジェクトに生成される汎用インターフェースおよびクラスのセットを提供します。バインダー・クラスは、デフォルトで jve.generated という名前のパッケージに生成されます。ビジュアル・エディターは、ユーザーがユーザーの必要に合うようにさらにカスタマイズおよび拡張することができる汎用インプリメンテーションとしてのバインダー・クラスを提供します。このチュートリアルは、さらに基本的で単純なデフォルトのバインダー・クラスの能力と柔軟性を明示します。

重要: この演習を始める前に、是非以下のヘルプ・トピックをお読みになるようお勧めします。これらのトピックは、Java ビジュアル・エディターが提供するデータ・オブジェクト、データ・ソース、およびバインダーの背後にある機能およびロジックについて、さらに学習することに役立ちます。

- データ・バインダーの概要
- バインダー API 参照

このチュートリアルでは、Web サービス・データ・ソース、数種類のデータ・オブジェクト、数種類のバインダーをユーザーのアプリケーションで使用します。このアプリケーションにこれらのオブジェクトのインスタンスを追加すると、ビジュアル・エディターが必要なクラスをプロジェクト内の jve.generated パッケージに追加します。このプロジェクトで、データ・バインダーのロジックの拡張、交換、または書き換えができます。 Java ビジュアル・エディターは、このアプリケーションが使用しているデータ・オブジェクト、データ・ソース、バインダーをデザイン・ビューのフリー・フォーム域に表示して、バインディング・オブジェクトの視覚的なサポートを提供します。ビジュアル・エディターは、複数のビジュアル・コンポーネント、データ・オブジェクト、およびデータ・ソースの間に線を引いて、任意の選択したオブジェクトの現行バインディングを表示します。

次の図は、ビジュアル・コンポーネント、バインダー、データ・オブジェクト、およびデータ・ソースがどのように相互作用するかを単純化した概要として示しています。このチュートリアルでビルドするアプリケーションは、バインダーのやや複雑で、かつ創造的な用法を説明しています。この図は、ユーザーがビルドしているサンプル・アプリケーション内のバインダー、データ・オブジェクト、およびデータ・ソースを正確に示しているわけではありません。

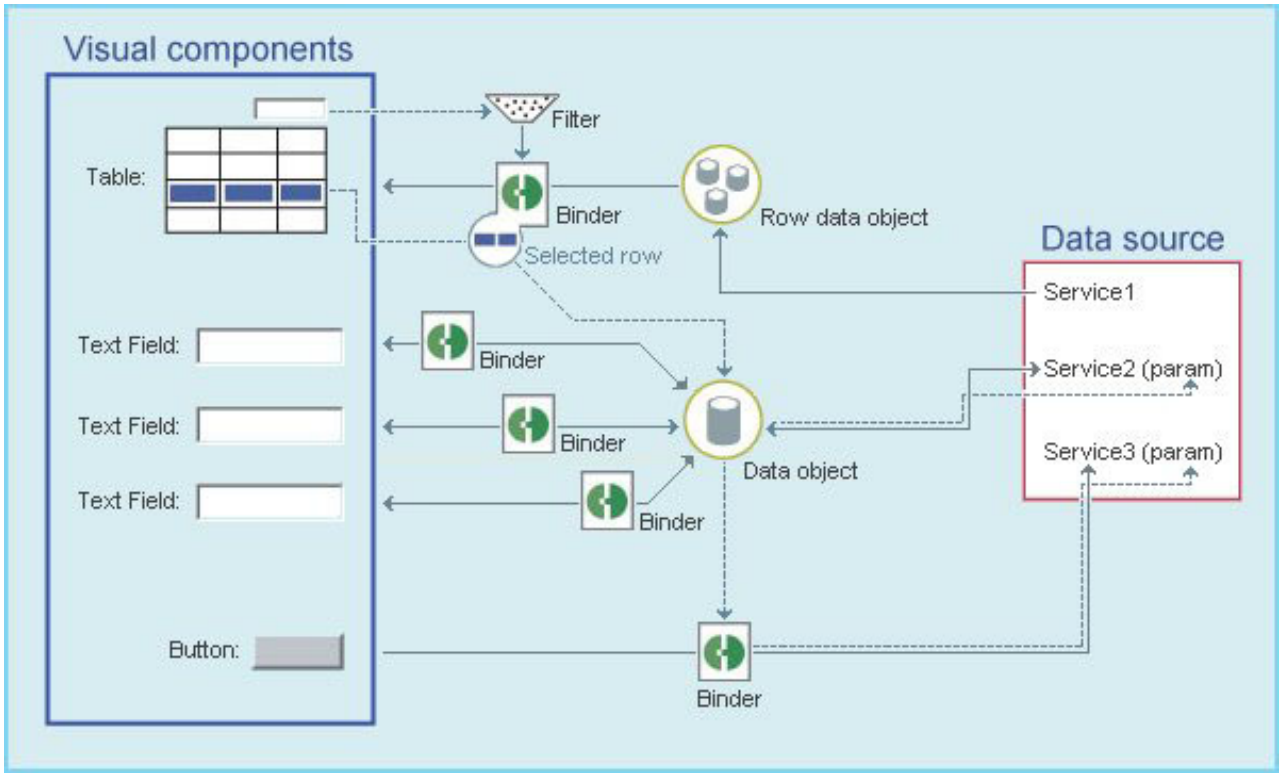


図 1. この図は、ビジュアル・コンポーネント、バインダー、データ・オブジェクト、およびデータ・ソースの関係の一例を示しています。

図 1 では、各ビジュアル・コンポーネントは、バインダーそのものをデータ・オブジェクトと関連付けるか、ボタンの場合では、データ・ソースと関連付ける独自のバインダーを持っています。テキスト・フィールドのバインダーは、フィールドを、データ・オブジェクトの特定のプロパティーにバインドします。この図内の行データ・オブジェクトとデータ・オブジェクトは、両方とも、それらのデータを、データ・ソース上のサービスへの直接呼び出しから取得します。テキスト・フィールドのデータ・オブジェクトは、テーブルで Service2 を呼び出すための引数として選択した行のキー値を使用します。これにより、テーブル内で選択した行について、おそらくさらに多くの情報を含むフル・レコードが戻されます。次に、このフル・レコードは、Service3 を呼び出す際に、ボタンのアクション・バインダーに対して引数として使用されます。これはフィールドに入力した値を更新するメソッドとすることもできます。データ・オブジェクト、データ・バインダー、およびデータ・ソースの詳細な説明は、前述のリンクからアクセスできます。

提供された WSDL ファイルを使用して、Web サービス Java プロキシーをプロジェクトに生成する

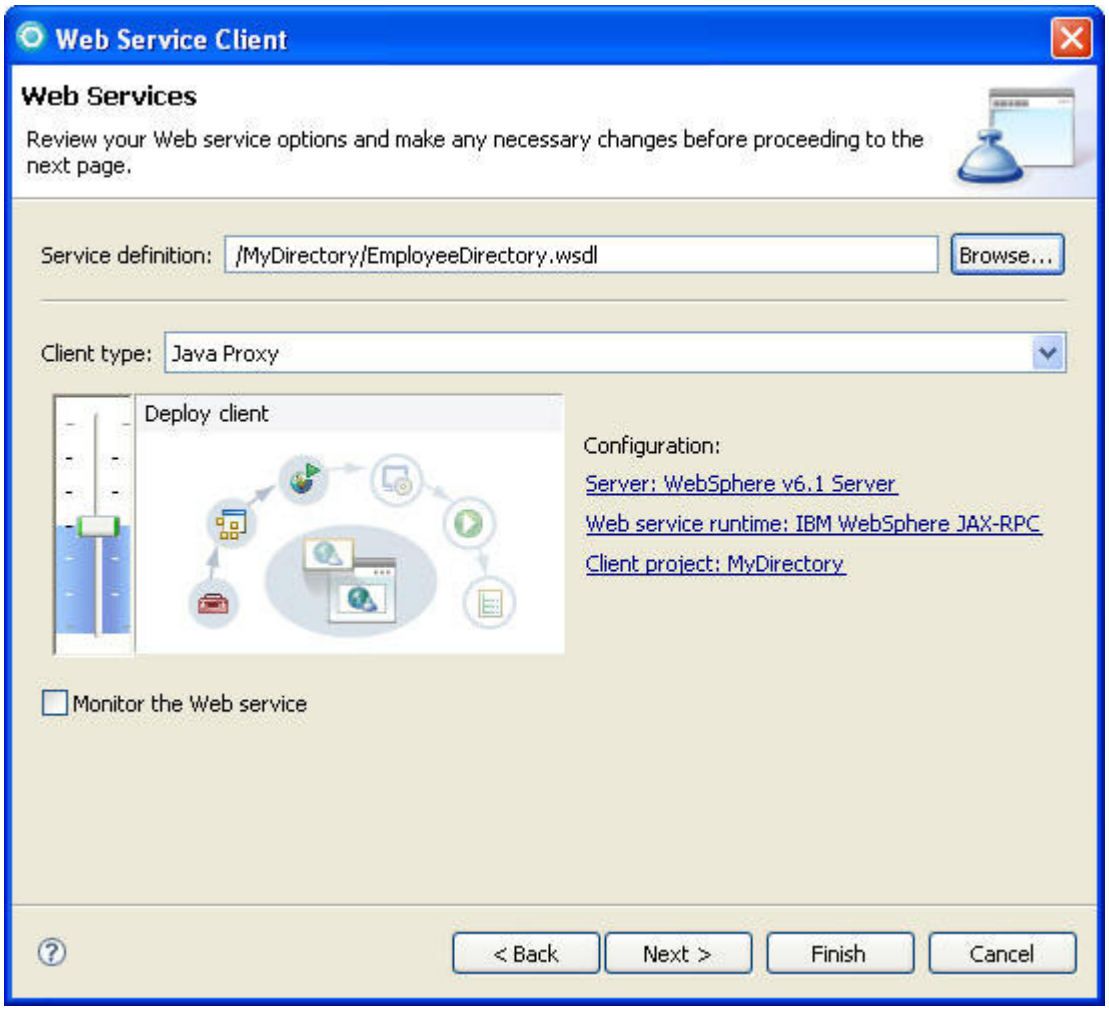
サーバー上で実行中の Web サービスを対象に作業をするには、Web サービスと対話するために、Java アプリケーションに Java プロキシー、またはクライアントが必要です。 WSDL ファイルを使用すると、「Web サービス・クライアント」ウィザードを使用して、 Java プロキシーをこの Java プロジェクトに生成することができます。 MyDirectory プロジェクトには、このプロキシーを生成するために使用する EmployeeDirectory.wsdl ファイルが組み込まれています。 Java プロキシーを生成したならば、Web サービスを表すデータ・ソースを作成し、ビジュアル・コンポーネントのバインディングを開始することができます。

重要: この演習で使用する WSDL ファイルは、ローカルにインストールした WebSphere Application Server 上に Web サービスがデプロイされており、 localhost (http://localhost:9080) のデフォルト・ポート

を使用していることを前提としています。これとは違う方法で EAR ファイルをデプロイする場合は、課題を進める前に、それに従って WSDL ファイルを編集してください。

Web サービス Java プロキシーをプロジェクトで生成するには、次のようにします。

1. メインメニューで、「**ファイル**」 → 「**新規**」 → 「**その他**」」の順にクリックし、「**Web サービス**」 → 「**Web サービス・クライアント**」ウィザードの順に選択する。 Web サービスのカテゴリーが表示されていない場合は、「**すべてのウィザードを表示**」を選択します。
2. ウィザードを使用して、Web サービス・クライアントを次のように定義する。
   a. 「**サービス定義**」に、MyDirectory プロジェクトで提供された WSDL ファイルを次のように入力する: `/MyDirectory/EmployeeDirectory.wsdl`
   b. 「**クライアント・タイプ**」フィールドで、「**Java プロキシー**」を選択する。
   c. スライダー・バーを「**クライアントのデプロイ (Deploy client)**」に設定する。
   d. サーバーと Web サービス・ランタイムが、実行中のサーバーに対して正しく設定されていることを確認する。このチュートリアルは、WebSphere v6.0 と WebSphere v6.1、および IBM WebSphere JAX RPC ランタイムに対してテスト済みです。
   e. Java プロキシー・クライアントが MyDirectory プロジェクトに出力されたことを確認する。



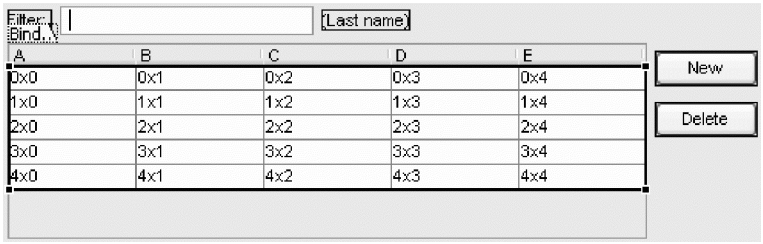
3. 「**終了**」をクリックする。「Web サービス・クライアント」ウィザードは Java プロキシーをプロジェクト内の新規パッケージ (directory.service) に生成します。

Web サービスが戻す行データ・オブジェクトに employeesTable をバインドする

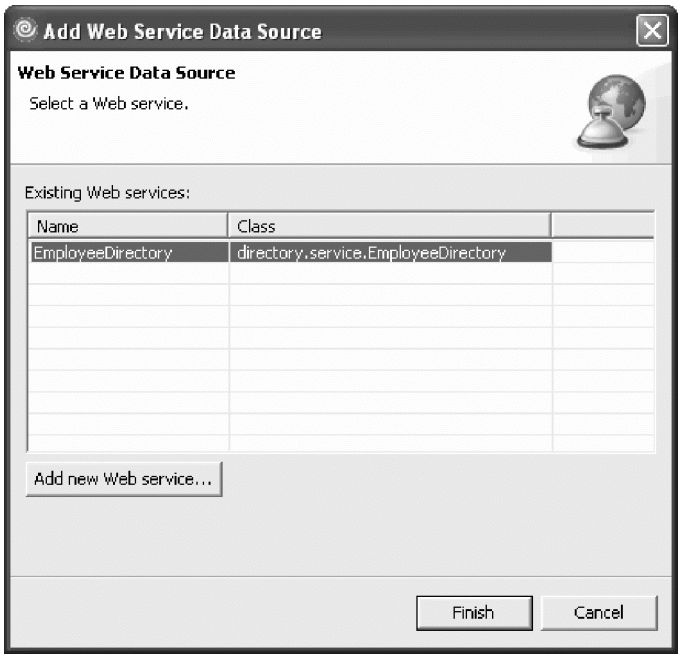
employeesTable は、このアプリケーションでバインディングする最初のビジュアル・コンポーネントであるため、ユーザーがプロジェクトに追加したばかりのサンプル Web サービス・プロキシーを指すデータ・ソースを作成する必要があります。これより後の課題で、その他のビジュアル・コンポーネントをバインドする場合は、このデータ・ソースを再利用します。このステップでは、Web サービス・データ・ソースと lightEmployeeRecordRows データ・オブジェクトを追加します。

従業員テーブルをバインドするには、以下のようにします。

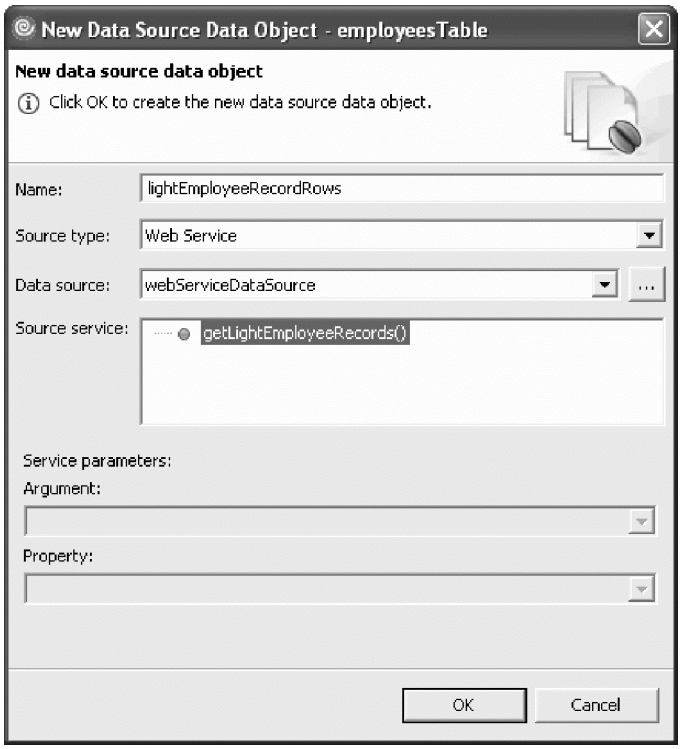
1. Java Bean ビューまたはデザイン・ビューで、employeesTable を選択する。 (その JScrollPane の親を選択していないことを確認します。) デザイン域の employeesTable の上部に、「**バインド**」とラベル付けされた小さなタブが表示されます。



2. employeesTable の上にある「**バインド**」タブをクリックする。代わりに、employeesTable を右クリックして、「**バインディング・プロパティー**」を選択することもできます。
3. アプリケーションには、まだデータ・オブジェクトがないので、新しいデータ・オブジェクトを追加する必要がある。「**新規データ・ソース・データ・オブジェクト**」をクリックします。
4. 「**ソース・タイプ**」フィールドで、「**Web サービス**」を選択する。
5. アプリケーションに Web サービス・データ・ソースをまだ追加していないので、ここで追加する必要がある。「**データ・ソース**」フィールドの隣にある「**新規...**」ボタンをクリックして、「Web サービス・データ・ソースの追加」ダイアログ・ボックスを開きます。これはプロジェクトにある使用可能な Web サービス・クライアント、またはプロキシーを探します。
6. EmployeeDirectory Web サービスを選択して、「終了」をクリックする。新規のデータ・ソースが DirectoryApp.java ファイルに追加されます。

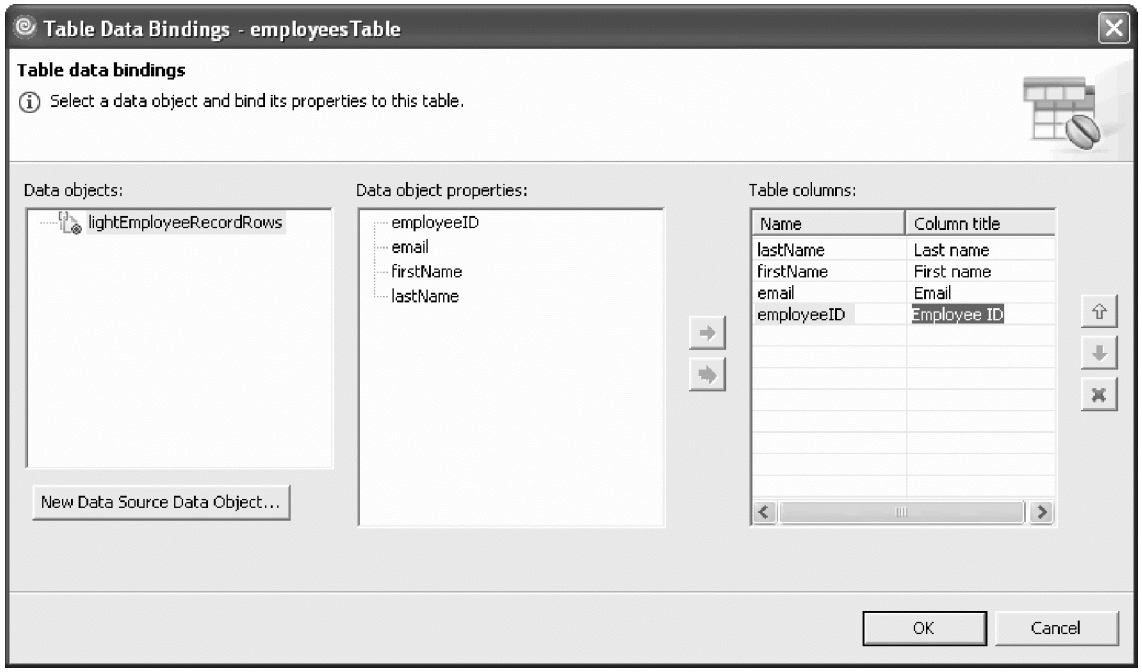


7. 「新規データ・ソース・データ・オブジェクト」ダイアログ・ボックスで、「ソース・サービス」フィールド内で getLightEmployeeRecords() を選択し、新規データ・オブジェクトのデフォルト名 (lightEmployeeRecordRows) を受け入れる。このサービス・メソッドにパラメーターは必要ありません。「OK」をクリックする。新しいデータ・オブジェクトが作成され、デザイン・ビューのフリー・フォーム域に表示されます。



ヒント: テーブルをバインディングしているので、「新規データ・ソース・データ・オブジェクト」ダイアログ・ボックスには、行データ・オブジェクトを戻すサービスだけが表示されます。この場合、getLightEmployeeRecords() メソッドは、オブジェクトの配列を戻す唯一の使用可能なサービスです。

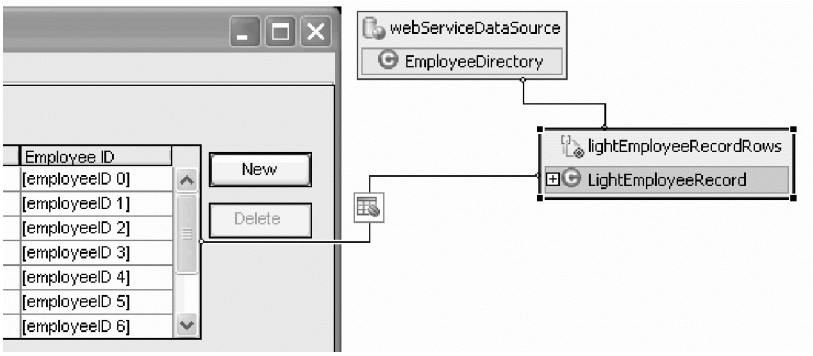
8. 「テーブル・データのバインディング」ダイアログ・ボックスで、lightEmployeeRecordRows データ・オブジェクトを選択する。
9. ここで、employeesTable に表示したい lightEmployeeRecordRows データ・オブジェクトのプロパティーを選択する必要があります。



   a. 二重矢印⇨ ボタンをクリックして、すべてのオブジェクト・プロパティーを「**テーブル列**」リストに追加する。
   b. 上矢印と下矢印を使用して、上から下に次の順序で列を整理する: lastName, firstName, email, employeeID
   c. 列のタイトルを次のように名前変更する: `Last name`, `First name`, `Email`, `Employee ID`

      ヒント: テーブルのバインディングが終了したならば、いつでもバインディング・プロパティーに戻って、列を名前変更したり、再配列したりできます。
   d. 「**OK**」をクリックする。

これで、JRowTableBinder を使用して、employeesTable が lightEmployeeRecordRows データ・オブジェクトにバインドされました。フリー・フォーム域の lightEmployeeRecordRows データ・オブジェクトをクリックすると、ビジュアル・エディターは、データ・オブジェクトからテーブルまで線を引きます。この線上に、テーブル・バインダー ・アイコンによって、JRowTableBinder が示されます。もう 1 本の線は、データ・オブジェクトが webServiceDataSource をそのデータ・ソースとして使用していることを示します。



演習のチェックポイント

プロジェクトおよびアプリケーションに加えられた変更に、注意してください。この課題では、Web サービス・データ・ソースと行データ・オブジェクトを追加し、行データ・オブジェクトに employeesTable をバインドするためのバインダーを追加しました。

Java ビジュアル・エディターによって生成されたすべてのバインダー・クラスを保持するために、ユーザーのプロジェクト内に作成された、新規パッケージ (jve.generated) を調べてください。また Web サービス用の Java プロキシーを保持する新規パッケージ (directory.service) にも注意してください。この演習で学習したことを、記述または要約してください。

これで、My Company Directory アプリケーションを実行すると、Web サービスによって既存の従業員レコードが従業員テーブルに取り込まれるようになります。

演習 2.3: 詳細フィールドをテーブル選択にバインドする

直前の演習では、Web サービスで getLightEmployeeRecords() サービスが戻した lightEmployeeRecordRows データ・オブジェクトに employeesTable をバインドしました。ここでは、テーブルで選択した従業員に基づいて、詳細フィールドに取り込みを行う必要があります。

選択したそれぞれの従業員の追加の詳細を得るには、別のデータ・オブジェクトを使用します。追加する selectedEmployeeRecord データ・オブジェクトは、getFullEmployeeRecord() サービスによって戻されます。このサービスは、テーブル上で選択された従業員の ID をパラメーターとして使用し、電話番号および就労場所を含む従業員のさらなる詳細データを取得します。

行データ・オブジェクトにテーブルをバインドしたときに使用した JRowTableBinder を使用すると、このステップは簡単になります。 JRowTableBinder を使うと、テーブル内の選択した要素を独立したデータ・オブジェクトとして取り出すことができるので、これを getFullEmployeeRecord(java.lang.Integer) メソッドのパラメーターとして使用できます。これによって、それぞれのテキスト・フィールドを selectedEmployeeRecord データ・オブジェクト内の対応するプロパティーに、簡単にバインドできます。

この Web サービスについて、さらに学習したい方に: Web サービスには、それぞれの従業員のすべての詳細データを取得するための 2 つのサービスが組み込まれています。テーブルには全従業員がリストされますが、テーブルには、データのサブセットだけが表示されます。次に 1 人の従業員を選択したときに、

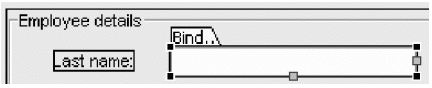
選択されたその従業員に限り、この従業員の残りの情報を取り出すことができます。 Web サービスが、テーブルが要求する各従業員すべてのデータを送ると、 Web トラフィックが濃密になり、アプリケーションのパフォーマンスが低下することがあります。

たとえば、従業員のレコードに写真または添付資料が含まれている場合、従業員全員のリストを作成するだけのために、全員の写真を取得することは望ましくないでしょう。したがって、まず getLightEmployeeRecord サービスを使用してテーブルを作成し、テーブルで選択された従業員のフル・レコードを getFullEmployeeRecord を使って取り込みます。

「Last name」フィールドをバインドする

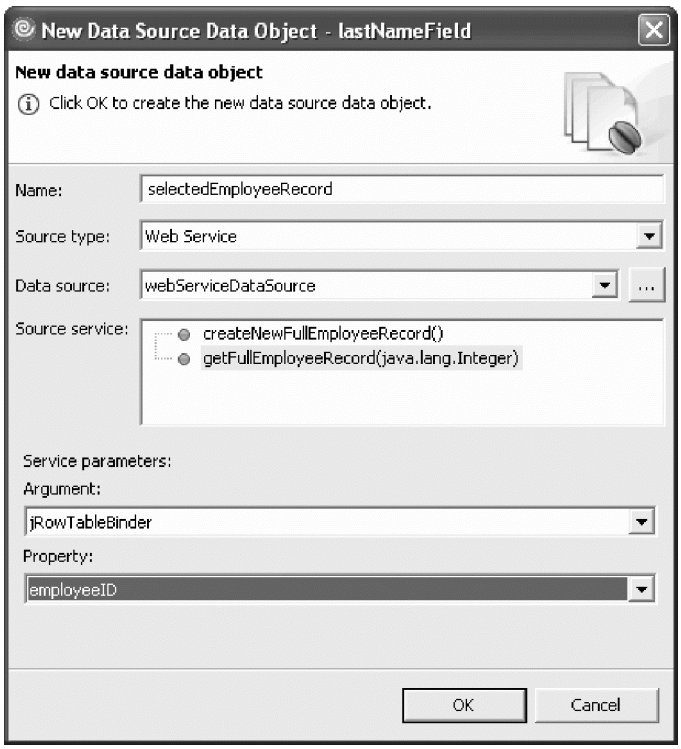
このステップでは、「**Last name**」フィールドを selectedEmployeeRecord データ・オブジェクトの lastName プロパティーにバインドします。

1. 「Java Bean」ビューまたはデザイン・ビューで、ラストネーム (lastNameField) のための JTextField を選択する。このデザイン域では、テキスト・フィールドに「**バインド**」タブが表示されます。

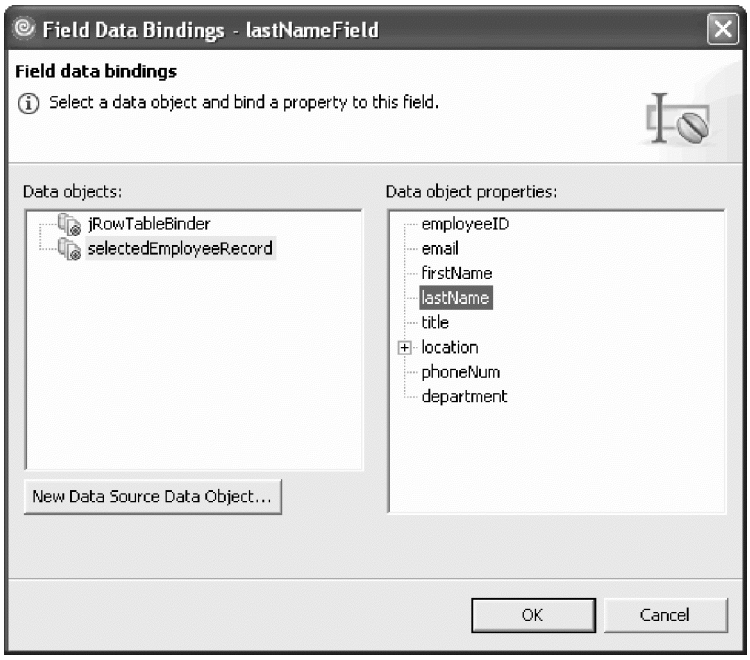


2. 「**バインド**」タブをクリックして、「フィールド・データのバインディング」ダイアログ・ボックスを開く。
3. 「**新規データ・ソース・データ・オブジェクト**」をクリックする。既存の jRowTableBinder データ・オブジェクトが正しいラストネームを戻しますが、これには従業員のフル・レコードは含まれていません。そこで、従業員のフル・レコードを表す新しいデータ・オブジェクトを作成する必要があります。
4. 「**ソース・タイプ**」フィールドで、「**Web サービス**」が選択され、「**データ・ソース**」には、「**webServiceDataSource**」が選択されていることを確認する。
5. 「**ソース・サービス**」リストから、「getFullEmployeeRecord(java.lang.Integer)」を選択する。「新規データ・ソース・データ・オブジェクト」ダイアログ・ボックスには、テキスト・フィールドと互換性のあるデータ・オブジェクトを戻すサービスがリストされます。
6. 「**名前**」フィールドに、`selectedEmployeeRecord` と入力する。
7. 「**引数**」フィールドで「jRowTableBinder」を選択し、「**プロパティー**」フィールドで「employeeID」を選択する。これで、選択した行の従業員 ID が getFullEmployeeRecord() サービス・メソッドの引数に設定されます。

   注: getFullEmployeeRecord(java.lang.Integer) は引数として整数を必要とします。フル・レコードを取り出すために、従業員テーブルで現在選択されている従業員 ID が使用できるとよいでしょう。テーブルをバインドすると、ビジュアル・エディターは jRowTableBinder を自動的に生成します。jRowTableBinder は、従業員テーブルでの現在の選択を listen します。これにより、整数パラメーターには、jRowTableBinder で選択した行の employeeID が使用されます。



8. 「**OK**」をクリックする。
9. 「フィールド・データのバインディング」ダイアログ・ボックスで、「**データ・オブジェクト**」リストに、selectedEmployeeRecord が選択されていることを確認する。 selectedEmployeeRecord データ・オブジェクトには、 jRowTableBinder データ・オブジェクトより多くの使用可能なプロパティーがあることに注目してください。
10. 「**データ・オブジェクトのプロパティー**」リストから、「lastName」プロパティーを選択する。



11. 「**OK**」をクリックする。これでアプリケーションの「Last name」フィールドは、selectedEmployeeRecord データ・オブジェクトの「lastName」プロパティーにバインドされました。これは、getFullEmployeeRecord() によって戻されます。

    selectedEmployeeRecord という名前の新しいデータ・オブジェクトが作成されて、アプリケーションに追加されます。データ・オブジェクトのビジュアル表示が次のイメージに示されているようにデザイン・ビューのフリー・フォーム域に追加されます。

    これで、デザイン域で「lastName」フィールドを選択すると、selectedEmployeeRecord にバインドされていることを線が示します。線の中央にあるテキスト・バインダー・アイコンは、このバインディングに使用されている SwingTextComponentBinder を表しています。デザイン域で線またはバインダーを示すアイコンを選択すると、「プロパティー」ビューでそのバインダーのプロパティーを調べることができます。

残りの詳細フィールドをバインドする

従業員の残りの各詳細フィールドをバインドするには、「Last name」フィールドと同様のプロセスを行いますが、データ・オブジェクトを追加する必要はありません。すでに selectedEmployeeRecord データ・オブジェクトを追加済みのため、それぞれのフィールドを selectedEmployeeRecord データ・オブジェクトの対応するプロパティーに単純にバインドすることができます。

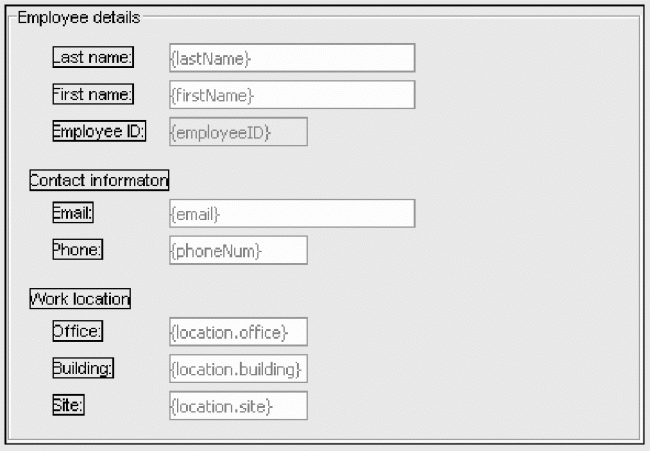
フィールドをバインドするには、このアプリケーションの「従業員の詳細」セクションにあるフィールドのそれぞれに対して、次の手順を完了します。

1. デザイン・ビューでフィールドを選択し、「**バインド**」タブをクリックする。
2. 「フィールド・データのバインディング」ダイアログ・ボックスで、「**データ・オブジェクト**」リストから、「selectedEmployeeRecord」を選択する。
3. 「**データ・オブジェクトのプロパティー**」リストで、バインドしようとするフィールドの該当するプロパティーを選択する。次の図表は、各テキスト・フィールドにバインドする必要があるプロパティーを示しています。

フィールド	**selectedEmployeeRecord データ・オブジェクトのプロパティー**
lastNameField	lastName
firstNameField	firstName
idField	employeeID
emailField	email
phoneField	phoneNum
officeField	location.office
buildingField	location.building
siteField	location.site

4. 「**OK**」をクリックする。

テキスト・フィールドのバインディングが終了したならば、デザイン域は次のイメージのようになるはずです。



「Employee ID」フィールドを読み取り専用にする

「Employee ID」フィールドは、そのフィールド上で編集可能プロパティーが「false」に設定されているため、使用不可になっています。しかし、テキスト・フィールド・バインダーのデフォルト動作は、データ・オブジェクトに値が含まれていると、使用可能の状態を変更します。フィールドが最初の読み取り専用状態のままになるように、このバインダー動作をオフにすることができます。

バインダーが編集可能プロパティーを自動的に切り替えるのを防止するには、以下のようにします。

1. 「Employee ID」フィールドを選択する。そのフィールドのバインダーを表す線が、アイコン とともにデザイン域に表示されます。
2. 「Employee ID」フィールドのバインダー アイコンをクリックする。
3. 「プロパティー」ビューで、autoEditable プロパティーを **false** に変更する。 **Enter** キーを押す。

演習のチェックポイント

これで、アプリケーションを実行し、テーブルから従業員を選択したときに、その従業員のレコードの詳細が詳細フィールドに表示されます。

演習 2.4: アクション・バインダーに「Update」ボタンをバインドする

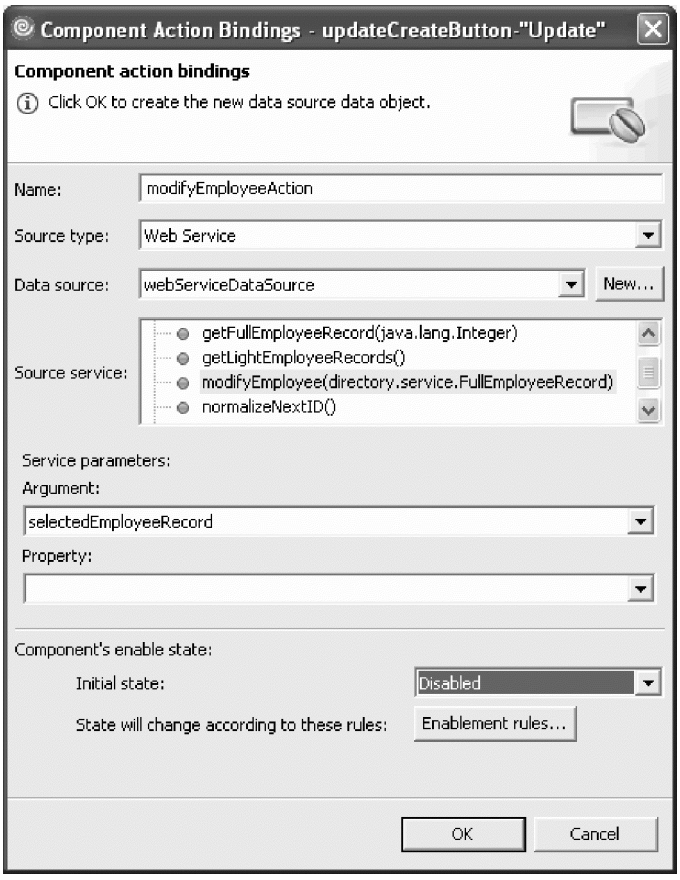
Java ビジュアル・エディターは、ボタンをクリックするとデータ・ソース上のサービスを呼び出すためのアクション・バインダーを提供しています。たとえば、「Update」ボタンをクリックした場合、このアプリケーションは、詳細フィールドに入力された変更を元に、Web サービスの modifyEmployee() メソッドを実行する必要があります。この演習では、アクション・バインダーに「Update」ボタンをバインドします。

「Update」ボタンをバインドするには、以下のようにします。

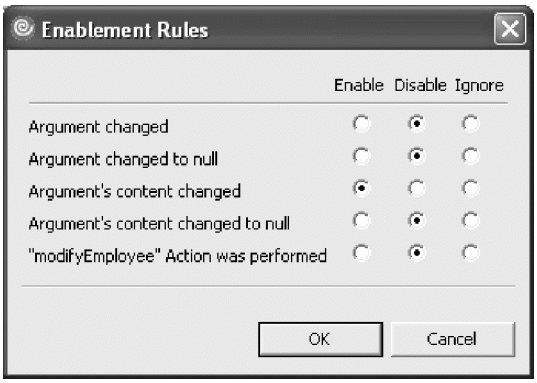
1. デザイン域で「**Update**」ボタンを選択し、「**バインド**」タブをクリックして「コンポーネント・アクションのバインディング」ダイアログ・ボックスを開く。

2. 「**ソース・タイプ**」フィールドで、「**Web サービス**」を選択する。
3. 「**データ・ソース**」フィールドで、「**webServiceDataSource**」を選択する。

4. 「**ソース・サービス**」リストから、「**modifyEmployee(directory.service.FullEmployeeRecord)**」を選択する。
5. 「**名前**」フィールドが自動的に **modifyEmployeeAction** に変わる。このデフォルトを受け入れます。
6. 「**引数**」フィールドで、「**selectedEmployeeRecord**」を選択する。
7. modifyEmployee() メソッドは、従業員のフル・レコードを引数として取るため、「**プロパティー**」フィールドは空白にしておく必要があります。
8. ボタンの「**初期状態**」を「**無効**」に設定する。



9. ボタンの状態をどのように変化させるのかを定義するために、「**有効化の規則**」をクリックする。引数の内容が変更されたときにのみボタンが使用可能になり、それ以外の場合には使用不可になるように指定します。「**OK**」をクリックします。



つまり、「**Update**」ボタンは、selectedEmployeeRecord の内容が変更されるまで使用不可になります。言い換えれば、selectedEmployeeRecord にバインドされているいずれかの詳細テキスト・フィールドの 1 つに新しい値を入力すると、バインダーはボタンをすぐに使用可能にします。新規レコードを選択するか、「**Update**」をクリックすると、ボタンは再び使用不可になります。

10. 「**OK**」をクリックする。

新規の SwingDataServiceAction バインダーが「**Update**」ボタンに対して追加されます。このボタンをデザイン域で選択すると、ビジュアル・エディターは、このボタンが Web サービス・データ・ソースにバインドされていることを示す線を引きます。ピンクの点線の矢印が selectedEmployeeRecord オブジェクトからこの線を指します。この矢印は、selectedEmployeeRecord がこのサービスに対する呼び出しの引数であることを示しています。

演習のチェックポイント

これで、アプリケーションを実行すると、従業員レコードを更新できます。

テーブルにある従業員を選択し、ラストネームを変更します。ラストネームを変更すると、「**Update**」ボタンがすぐに使用可能になります。「**Update**」をクリックすると、modifyEmployee サービスが呼び出され、その従業員が更新されます。新規ラストネームが従業員テーブルに反映されます。

演習 2.5:「Delete」ボタンと確認ダイアログ・ボックスを使用可能にする

この演習では、従業員レコードを削除するように My Company Directory アプリケーションをプログラムします。

次のリストは、アプリケーションに使用させたい動作を説明しています。

- テーブルから 1 名の従業員を選択すると、「**Delete**」ボタンが使用可能になる。
- 「**Delete**」ボタンをクリックすると、「Confirm Delete」ダイアログ・ボックスが開き、削除の確認を求められる。
- 「Confirm Delete」ダイアログ・ボックスの「**Yes**」ボタンをクリックすると、従業員レコードが削除されて、「Confirm Delete」ダイアログ・ボックスが閉じ、従業員レコードのリストが更新される。
- 「**No**」をクリックすると、削除がキャンセルされて、「Confirm Delete」ダイアログ・ボックスが閉じる。

テーブルの行が選択されているかいないかに基づいて、「Delete」ボタンが使用可能または使用不可になるようにプログラムする。

「Delete」ボタンを使用可能または使用不可になるようにプログラムするには、行が選択されたときにボタンを使用可能にするリスナーをテーブルに追加します。

1. 「Java Bean」ビューで、「employeesTable」を選択する。「ソース」ビューで、次の行が強調表示されます。

```
employeesTable = new JTable();
```

2. この行の直後に、employeesTable で、新しい ListSelectionListener、および valueChanged イベントを追加する。

```
employeesTable.getSelectionModel().addListSelectionListener(new ListSelectionListener() {
    public void valueChanged(ListSelectionEvent e) {
       getDeleteButton().setEnabled(getEmployeesTable().getSelectedRowCount() != 0);
   }
});
```

3. これらのコード行を追加すると、ソース・エディターは、ListSelectListener と ListSelectionEvent をインポートするまで、それらにエラーのマークを付ける。必要なインポートを追加するには、メインメニューで「**ソース**」 → 「**インポートの編成**」とクリックします。クラスのインポート・セクションに、以下の行が追加されます。

```
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
```

これで、テーブル内の行を選択すると、「**Delete**」ボタンが使用可能になります。

「Delete」をクリックしたときに、「Confirm Delete」ダイアログ・ボックスが開くようにプログラムする。

actionPerformed イベントを「Delete」ボタンに追加して、「Confirm Delete」ダイアログ・ボックスが開くように、イベントをプログラムします。

1. 「**Delete**」ボタンを右クリックして、「**イベント**」 → 「**actionPerformed**」」と選択する。次のイベント・スタブが、getDeleteButton() メソッドに追加されます。

```
deleteButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e) {
     System.out.println("actionPerformed()");
     // TODO Auto-generated Event stub actionPerformed()
   }
});
```

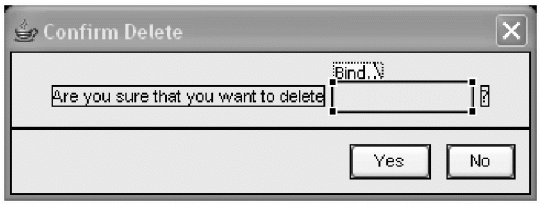
2. この生成されたスタブを、次のコードで置き換える。このコードによって、ボタンがクリックされたときに「Confirm Delete」ダイアログ・ボックスが表示されるように設定します。

```
deleteButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent e) {
     getConfirmDialog().setVisible(true);
   }
});
```

「Confirm Delete」ダイアログ・ボックスのテキスト・フィールドをバインドする。

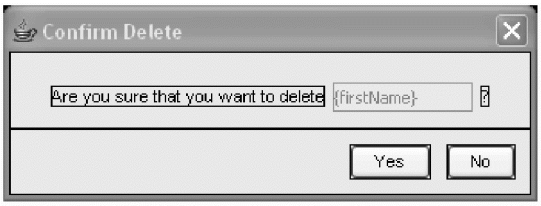
「Confirm Delete」ダイアログのテキスト・フィールドをバインドして、削除する従業員のファーストネームを表示します。

1. Java Bean ビューまたはデザイン域で、「employeeToDeleteField」テキスト・フィールドを選択して、「**バインド**」タブをクリックする。



2. 「フィールド・データのバインディング」ダイアログ・ボックスで、 selectedEmployeeRecord データ・オブジェクトと「firstName」フィールドを選択して、「**OK**」をクリックする。

   これで、employeesTable の選択された行にある firstName 列にテキスト・フィールドがバインドされます。



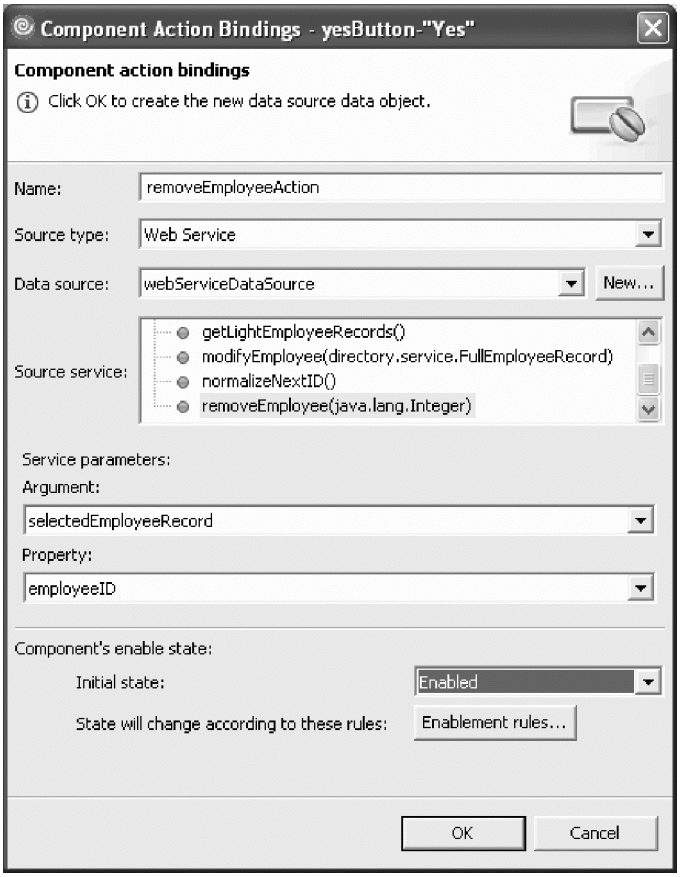
3. このフィールドが読み取り専用となることを保証するために、このフィールドのバインダーの「**autoEditable**」プロパティーを「**false**」に設定する。

削除を実行するように「Yes」ボタンをバインドする。

Web サービス上で removeEmployee(java.lang.Integer) メソッドを呼び出すように、「**Yes**」ボタンをバインドします。

1. 「**Yes**」ボタンを選択し、「**バインド**」タブをクリックして、「コンポーネント・アクションのバインディング」ダイアログを開く。
2. 「**ソース・タイプ**」フィールドで、「**Web サービス**」を選択する。
3. 「**データ・ソース**」フィールドで、「**webServiceDataSource**」を選択する。
4. 「**ソース・サービス**」リストから、「**removeEmployee(java.lang.Integer)**」を選択する。
5. 「**名前**」フィールドが自動的に **removeEmployeeAction** に変わる。このデフォルトを受け入れます。
6. 「**引数**」フィールドで、「**selectedEmployeeRecord**」を選択する。
7. 「**プロパティー**」フィールドで、「**employeeID**」を選択する。 removeEmployee() メソッドはその引数として整数を取るため、selectedEmployeeRecord の従業員 ID を使用します。
8. ボタンの「**初期状態**」を「**有効**」に設定する。
9. 「**有効化の規則**」で、それぞれの条件について「 **無視**」を選択する。

   このコンポーネント状態は、その状態を変更する必要がないため、「Yes」ボタンが常時使用可能であることを意味します。



10. 「**OK**」をクリックする。

従業員が削除された後、「Confirm Delete」ダイアログを非表示にするイベントを追加する

このステップでは、イベントを (「**Yes**」ボタンそのものにではなく) 「**Yes**」ボタンのバインダー に追加します。従業員が除去された後、「Confirm Delete」ダイアログ・ボックスが閉じるようにしたいということは、バインダーが正常にサービスをデータ・ソースに呼び出した後で、それを行いたいということを意味します。

次のコードを、getRemoveEmployeeAction() メソッドに追加する。

```
removeEmployeeAction.addActionBinderListener(new jve.generated.IActionBinder.ActionBinderListener() {
   public void afterActionPerformed(jve.generated.IActionBinder.ActionBinderEvent e) {
     getConfirmDialog().setVisible(false);
   }
   public void beforeActionPerformed(jve.generated.IActionBinder.ActionBinderEvent e) {}
});
```

このイベント・コードにより、バインダーのアクションが実行されると「Confirm Delete」ダイアログ・ボックスが非表示になります。

演習のチェックポイント

これで、My Company Directory アプリケーションを実行したときに、テーブル内で従業員を選択して、「**Delete**」ボタンをクリックし、「**Yes**」をクリックして、削除を確認できるようになりました。従業員レコードは、この登録簿から除去され、従業員のリストは除去を反映したものになります。

演習 2.6: 新規の従業員追加用のアクションとバインディングをセットアップする

この演習では、My Company Directory アプリケーションで、新規従業員レコードを追加できるようにします。

新規従業員を追加する場合、アプリケーションの動作はより複雑かつダイナミックであるため、この演習は本質的に少し複雑であり、ユーザーはソース・コードの一部を手操作で変更する必要があります。さらに、この演習では、データ・オブジェクトの高度な機能の一部を示して、必要に応じてバインダーおよびデータ・オブジェクトおよびバインダーを使用できる方法の創造的な例を紹介します。

以下の項目は、アプリケーションで必須の動作の説明です。

- 「**New**」ボタンをクリックしたときに、次の動作が発生します。
  - 従業員テーブルの選択が解除され、テーブルが使用不可になる。
  - テーブル選択をクリアすると、「**Delete**」ボタンが使用不可になる。
  - 「**Filter**」フィールドが使用不可になる。
  - 新規従業員 ID を除いて、詳細フィールドのすべての値が消去される。
  - 「**Update**」ボタンのテキストが「**Add**」に変わる。
- 「**Add**」ボタンをクリックしたときに、次の動作が発生します。
  - 詳細フィールドに入力した値は、新規従業員レコードとして登録簿に追加される。
  - テーブルが使用可能になり、値が更新される。
  - 「**Filter**」フィールドが使用可能になる。
  - 「**Add**」ボタンのテキストが「**Update**」に戻る。

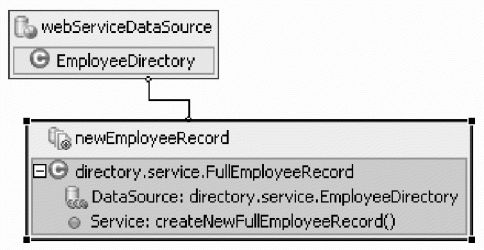
createNewFullEmployeeRecord() を呼び出す新規データ・ソース・データ・オブジェクトを追加する。

サンプル Web サービスは、次に利用可能な従業員 ID 番号を持つ、新しい空の従業員レコードを提供する createNewFullEmployeeRecord サービスを提供します。この空のレコードは、新しい従業員の情報を取り込んで、Web サービスに再度サブミットされます。

1. Java ビジュアル・エディターのパレット上で、データ・オブジェクト・ドロワーを展開して、「**データ・ソース・データ・オブジェクト**」を選択する。
2. デザイン・ビューまたはフリー・フォーム域にマウス・ポインターを移動して、左クリックし、データ・ソース・データ・オブジェクトをドロップする。新規データ・ソース・データ・オブジェクトが追加されて、フリー・フォーム域に表示されます。

3. データ・ソース・ データ・オブジェクトを右クリックして、「**フィールド名の変更**」を選択する。データ・オブジェクトの名前を `newEmployeeRecord` に変更します。
4. newEmployeeRecord データ・オブジェクトを右クリックして、「**バインディング・プロパティー**」を選択する。「データ・バインディング」ダイアログ・ボックスが開きます。
5. 「**データ・ソース**」フィールドで、「webServiceDataSource」を選択する。
6. 「**サービス**」フィールドで、「createNewFullEmployeeRecord」を選択する。
7. 「**OK**」をクリックする。

フリー・フォーム域で、newEmployeeRecord データ・ソース・データ・オブジェクトが Web サービスにバインドされるのを見ることができます。



データ・オブジェクトの切り替えを容易にするために基本データ・オブジェクトを追加する。

詳細フィールドおよび「Update」ボタンは、(新規従業員の更新および作成の両方の実行のため) モードを切り替える必要があるため、これらはそれぞれ異なる時点で 2 つの異なるデータ・オブジェクトにバインドされる必要があります。このステップを容易にするために、ユーザーは switchingDataObject という名前の基本データ・オブジェクトを追加します。この基本データ・オブジェクトを使用して、テキスト・フィールドのバインディングを selectedEmployeeRecord と newEmployeeRecord との間で切り替えます。

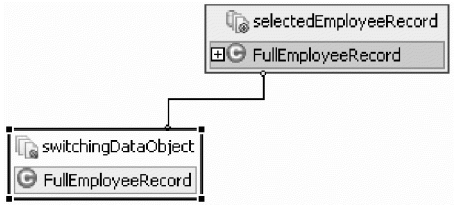
新規基本データ・オブジェクトは、単に、以前の演習でユーザーが定義した別のデータ・オブジェクト (selectedEmployeeRecord) を指すだけです。この新しいデータ・オブジェクトは、この課題の始めに作成した newEmployeeRecord を使用するよう、この基本データ・オブジェクトに指示するメソッドを作成すると、役立つようになります。すなわち、この基本データ・オブジェクトは、中間データ・オブジェクトとして機能します。この中間データ・オブジェクトは selectedEmployeeRecord データ・オブジェクトと newEmployeeRecord データ・オブジェクトとの間で切り替えを行い、対象のアプリケーション内のビジュアル・コンポーネントが 2 つの異なるデータ・オブジェクトと連動することを可能にします。

1. ビジュアル・エディターのパレット上で、「**基本データ・オブジェクト**」を選択し、フリー・フォーム域にドロップする。 basicDataObject が追加されます。

2. データ・オブジェクトを `switchingDataObject` に名前変更する。
3. switchingDataObject の「プロパティー」ビューで、 **sourceObject** プロパティーを **selectedEmployeeRecord** に設定する。 selectedEmployeeRecord は、プロパティーの「Value」列のドロップダウン・メニューから選択できます。

   これで switchingDataObject は、selectedEmployeeRecord を参照し、同じ値を反映するようになります。

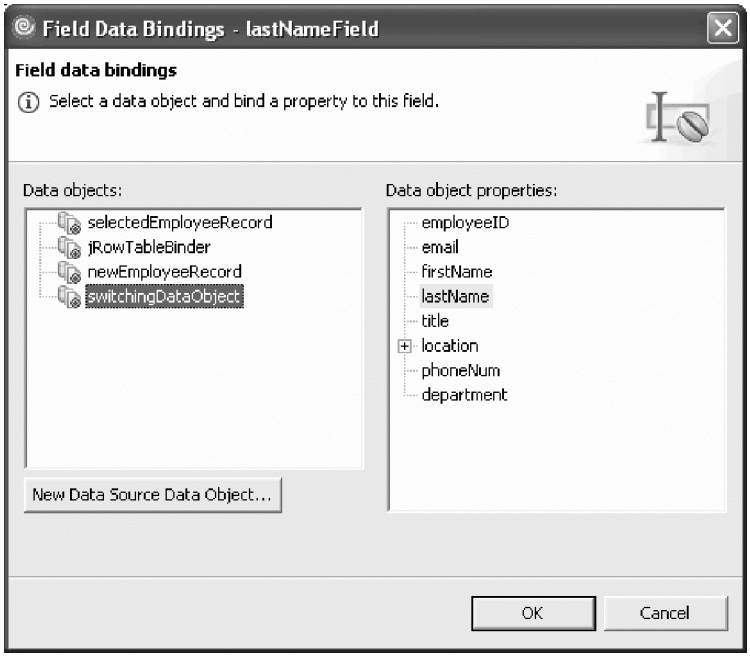
   

それぞれの従業員フィールドを switchingDataObject に再バインドする。

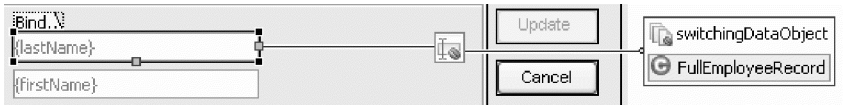
各従業員詳細フィールドが既に selectedEmployeeRecord にバインドされていても、ここでは、それらを switchingDataObject にバインドします。これらのフィールドをバインドしたならば、既存の従業員レコードを変更する場合か、または新規従業員レコードを追加するかによって、テキスト・フィールドのデータ・オブジェクト間を動的に切り替えることができます。

従業員詳細セクションのそれぞれのフィールドについて、次の手順を完了します。

1. フィールドを選択して、「**バインド**」タブをクリックする。
2. 「フィールド・データのバインディング」ダイアログ・ボックスで、switchingDataObject を選択する。これらのフィールドは selectedEmployeeRecord に以前バインドされています。



3. フィールドが正しいデータ・オブジェクト・プロパティーにまだバインドされていることを確認して、「**OK**」をクリックする。デザイン・ビューでこのフィールドを選択すると、これで、バインダーの線が switchingDataObject を指しているのを見ることができます。



モードの更新および切り替えのフラグおよびメソッドを定義する。

次の updateMode() メソッドは、モード・フラグが「新規」に設定されているかどうか調べるために検査して、状況に応じて、それに従ってアプリケーションの動作を変更します。デフォルトでは、Boolean のフラグの isNewMode は「false」に設定されています。また updateMode() メソッドは、従業員テーブルと「Filter」フィールドを使用可能にし、「Update」ボタンのテキストを「Update」に設定します。isNewMode を「true」に設定すると、従業員テーブルは使用不可になり、選択項目があると解除され、「Filter」フィールドは使用不可になり、「Update」ボタンのテキストは「Add」に設定されます。

DirectoryApp.java クラスに、次のコードを最後の閉じ中括弧の直前に追加します。

```
private boolean isNewMode = false;
private void updateMode() {
   if (isNewMode) {
      getEmployeesTable().clearSelection();
      getEmployeesTable().setEnabled(false);
      getFilterField().setEditable(false);
      getUpdateCreateButton().setText("Add");
   } else {
      getEmployeesTable().setEnabled(true);
      getFilterField().setEditable(true);
      getUpdateCreateButton().setText("Update");
   }
}
```

actionPerformed イベントを「New」ボタンに追加する。

このステップでは、「**New**」ボタンがクリックされたときのイベント・コードを追加します。イベントは、switchingDataObject に対して newEmployeeRecord データ・オブジェクトを使用するように指示を出し、モード・フラグを「新規」に設定して、直前のステップでユーザーが追加した updateMode() メソッドを実行します。

1. デザイン・ビューで、「**New**」ボタンを右クリックし、「**イベント**」 → 「**actionPerformed**」」と選択する。 getNewButton() メソッド内に、次のコードが生成されます。

```
newButton.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent e) {
      System.out.println("actionPerformed()"); // TODO Auto-generated Event stub actionPerformed()
   }
});
```

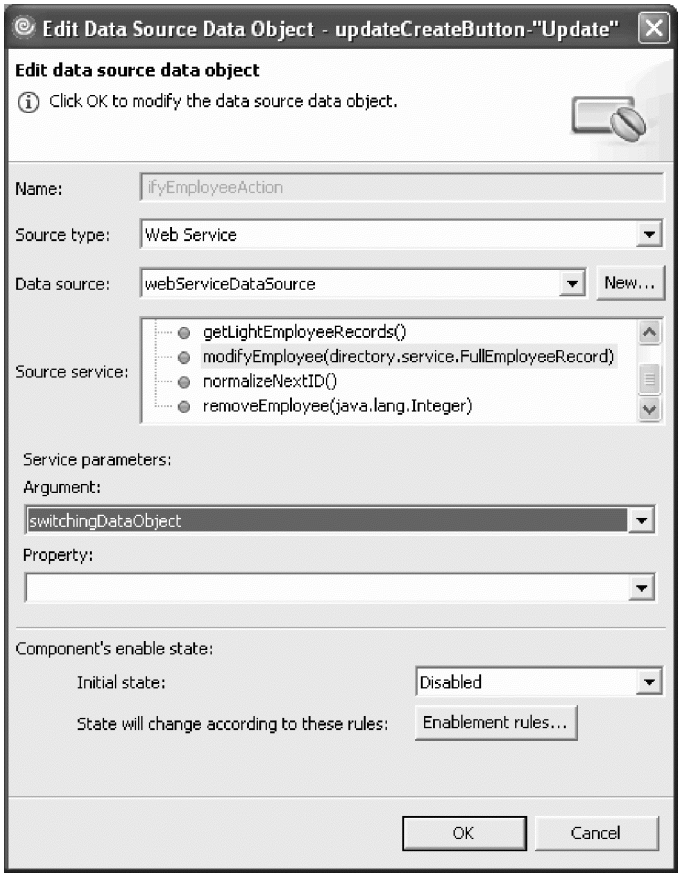
2. 生成されたこのスタブを次のコードで置き換える。

```
newButton.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent e) {
      getSwitchingDataObject().setSourceObject(getNewEmployeeRecord());
      getNewEmployeeRecord().refresh();
      isNewMode = true; //sets application to new mode
      updateMode(); //changes UI according to new mode
      getLastNameField().grabFocus();
   }
});
```

「Update」ボタンを再バインドする。

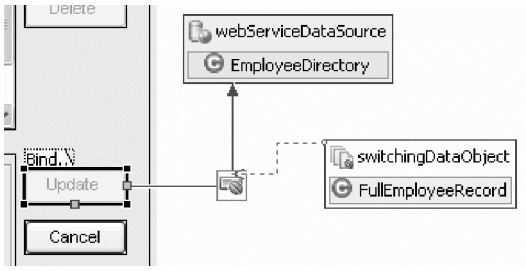
直前の課題では、modifyEmployee メソッドを Web サービスで使用するために、「**Update**」ボタンをプログラムしました。そのアクションは、SwingDataServiceAction としてインプリメントされています。SwingDataServiceAction のプロパティーの 1 つにソース・オブジェクトがあります。これは、サービスの引数としての役割を果たします。変更アクションのソース・オブジェクトは、現在、selectedEmployeeRecord に設定されています。更新と追加の両方を制御するようにボタンをプログラムするためには、 switchingDataObject を modifyEmployee サービスの引数として使用するようにボタンのアクションを再構成します。

1. デザイン・ビューで、「**Update**」ボタンを選択する。ピンクの点線の矢印は、selectedEmployeeRecord がサービス呼び出しの引数であることを示していることに注目してください。
2. 「**Update**」ボタン上にある「**バインド**」タブをクリックする。
3. 「**引数**」フィールドで、switchingDataObject を選択する。



4. 「**OK**」をクリックする。

これで、ボタンのアクションは、switchingDataObject を、modifyEmployee メソッドに対する 引数として使用するように構成されていることに注目してください。



モードをリセットするためにイベントを「Update」ボタンのバインダーに追加する

「**Update**」ボタンをクリックして、Web サービス上のアクションが完了した後で、アプリケーションをデフォルトのモードおよびボタン動作に戻したい場合があります。これを行うには、更新および追加が実行された後にモードを更新し、テーブルを最新表示するアクション・バインダー上のイベント・リスナーを追加します。

次のコードを、「Update」ボタンの getModifyEmployeeAction() メソッドに追加する。

```
modifyEmployeeAction.addActionBinderListener(
  new jve.generated.IActionBinder.ActionBinderListener() {
    public void afterActionPerformed(jve.generated.IActionBinder.ActionBinderEvent e) {
      if (isNewMode) {
        //Go back to using the selectedEmployeeRecord
        getSwitchingDataObject().setSourceObject(getSelectedEmployeeRecord());
```

```
        //Revert out of new mode
        isNewMode = false;
        updateMode();
      }
      // Refresh the table's data object
      getLightEmployeeRecordRows().refresh();
    }
    public void beforeActionPerformed(jve.generated.IActionBinder.ActionBinderEvent e) {}
});
```

演習のチェックポイント

これで、My Company Directory アプリケーションを実行したときに、「**New**」ボタンをクリックして、新規従業員レコードを追加できるようになりました。

演習 2.7:「Cancel」ボタンの動作をプログラムする

アプリケーションの使用中、従業員レコードに対する変更を行いかけたあとで、それを実施しないと決めた場合は、変更を簡単に取り消したいと考えることでしょう。言い換えれば、操作を最初からやり直せるようにするために、フィールドをキャンセルまたは消去できる必要があります。このような機能を追加するには、「**Cancel**」ボタンにいくつかの actionPerformed イベントを設定します。

次のリストは、「**Cancel**」ボタンの必須の動作を説明しています。

- 「新規」モードで「**Cancel**」ボタンをクリックすると、アプリケーションは新規モードから戻ります。
- 従業員レコードの変更中に「**Cancel**」ボタンをクリックすると、それまでに変更した値は元の値に戻ります。

actionPerformed イベントを「**Cancel**」ボタンに追加して、必須の動作を実行するには、次のようにします。

1. デザイン・ビューで、「**Cancel**」ボタンを右クリックし、「**イベント**」 → 「**actionPerformed**」と選択する。 getCancelButton() メソッド内に、次のコードが生成されます。

```
cancelButton.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent e) {
      System.out.println("actionPerformed()"); // TODO Auto-generated Event stub actionPerformed()
   }
});
```

2. 生成されたイベント・スタブを、次のコードで置き換える。

```
cancelButton.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent e) {
      if (isNewMode) {
         getSwitchingDataObject().setSourceObject(getSelectedEmployeeRecord());
         isNewMode = false;
         updateMode();
      } else {
         getSelectedEmployeeRecord().refresh();
      }
   }
});
```

演習のチェックポイント

この演習では、actionPerformed イベントを使用して「**Cancel**」ボタンをプログラムする方法を学習しました。

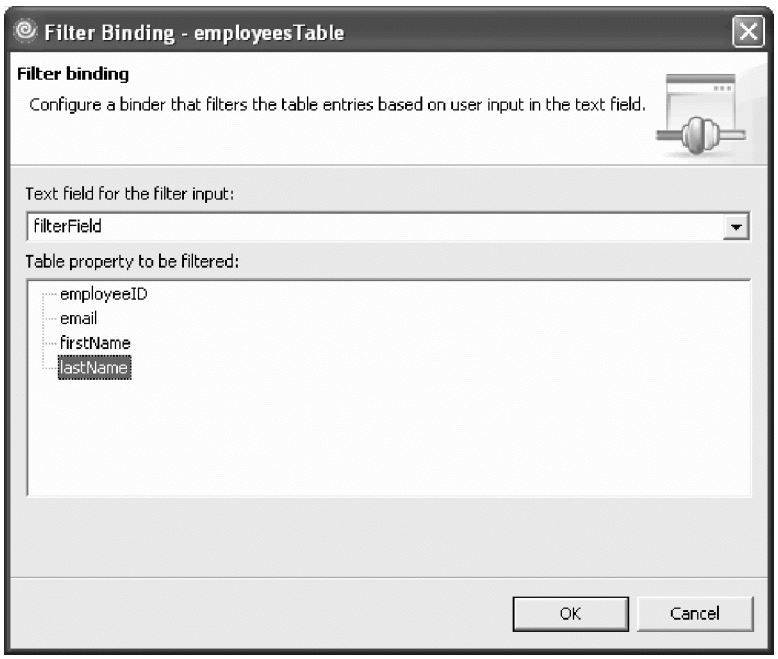
演習 2.8: 従業員テーブルのフィルターをセットアップする

テキスト・フィルター・バインダーを使用して従業員テーブルの内容をフィルタリングすることができます。このフィルターは、テキスト・フィールドから入力を受け取り、テーブル内の特定のプロパティーまたは列に基づいて、テーブルをフィルタリングします。

このアプリケーションでは、「**Filter**」フィールドに入力された文字を使用して、従業員のラストネームでフィルタリングを行います。「**Filter**」フィールドに入力された正確な値が従業員レコードのラストネームにある場合、そのテーブルの該当従業員レコードが表示されます。

このテーブルにフィルターを作成するには、次のようにします。

1. employeesTable のバインダー・アイコンを選択し、「**フィルターのバインディング・プロパティー**」を選択する。「フィルターのバインディング」ダイアログ・ボックスが開きます。
2. 「**フィルター入力のためのテキスト・フィールド**」リストで、「**filterField**」を選択する。
3. 「**フィルタリングするテーブル・プロパティー**」リストで、「**lastName**」を選択する。



4. 「**OK**」をクリックする。

新規の SwingPropertyFilter が生成されます。テーブルのバインダー上の「フィルター」プロパティーがこの新規フィルターを使用するように設定されます。新規フィルターは、その入力用に「**Filter**」フィールドを使用して、テーブルの「lastName」プロパティーを対象にフィルタリングするように構成されます。

演習のチェックポイント

この演習では、テーブル用のフィルターをセットアップする方法を学習しました。

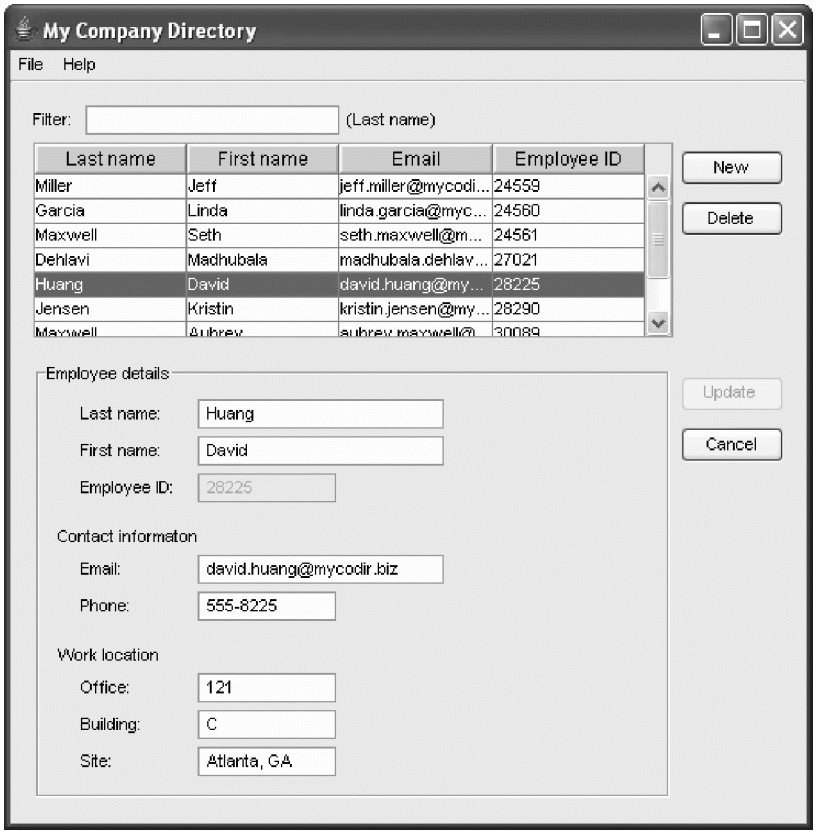
これで、My Company Directory アプリケーションを実行すると、「**Filter**」フィールド内に入力した文字でテーブルをフィルタリングし、入力した文字をラストネームに含む行を表示できるようになりました。

おつかれさまでした。My Company Directory アプリケーションが完成しました。

要約: Web サービスを使用するリッチ Java クライアントをビルドする

おつかれさまでした。Java ビジュアル・エディターを使用して、リッチ Java クライアントである (サンプルの Web サービスに接続して、従業員登録簿を管理する) My Company Directory アプリケーションをビルドする方法を学習しました。

製品の画面図を参照する:



演習での学習事項

ビジュアル・エディターを使用して、従業員テーブルを配置するために GridBagLayout を使用するグラフィカル・ユーザー・インターフェースを完成させました。次にテーブル、フィールド、およびボタンを適切なデータ・オブジェクトとデータ・ソースにバインドして、生成した Web サービスの Java プロキシーとアプリケーションが連動するようにしました。また、アプリケーションが正しく動作するようにして、それが使いやすく、直感的に理解できるものにするために、いくつかの複雑なコーディングも行いました。さらに、エンタープライズ・アプリケーションを WebSphere Application Server v6 上にインストールし、Web サービスをデプロイする方法を学習しました。

もっとも重要なことは、データを処理するために Java ビジュアル・エディターが提供する強力なバインダー・クラスについて学んだことです。これで、ご自身でバインダーを体験し、それを新しい、面白い使い方に応用できるようになる準備ができたのです。

以下の作業を行えるようになっているはずです。

- Java ビジュアル・エディターを使用して GridBagLayout 内にコンポーネントを配置する。
- ビジュアル・クラスを Java Bean として実行する。
- Java アプリケーションのビジュアル・コンポーネントを Web サービスが戻したメソッドおよびデータ・オブジェクトにバインドする。
- イベントをビジュアル・コンポーネントに追加する。

追加のリソース

完成した My Directory アプリケーションをインポートします

このプロジェクトには、完成したアプリケーションとして、バインダー・クラスが生成された jve.generated パッケージと、WebSphere Application Server v6.1 向けに構成した Web サービス Java クライアントが含まれています。この完成したプロジェクトを、チュートリアルを使用せずにインポートする場合は、Java ビルド・パスの変数の構成が必要になる場合があります。この変数は、WebSphere v6.1 Web サービス・シン・クライアントの JAR ファイルをポイントしている必要があります。

ヒント: インポートをする際に別のプロジェクト名を指定しない限り、MyDirectory プロジェクト・コンテンツは上書きされます。