

Chapter 1

Creating your first JavaServer Faces Web application

Chapter Contents

- ▶ Introducing Web applications and JavaServer Faces
- ▶ Installing Rational Application Developer
- ▶ Setting up a Web project
- ▶ Creating the Web application
- ▶ Running the Web application

Time Required

- ▶ 2 hours

Files Required

- ▶ none
-

We begin our guided tour by introducing you to Web applications and JavaServer Faces (called JSF or Faces), followed by a simple example of how you can create Faces Web applications in Rational Application Developer. JSF provides a rich set of user interface (UI) components and a solid framework upon which to build Web applications, but as you'll see in this book, the power really lies in Rational Application Developer's tooling, which lets you visually build Web applications.

Introducing Web Applications and JavaServer Faces

Web applications reside on application servers such as WebSphere® Application Server. These applications consist of one or more Web pages that users access through Web browsers, as shown in Figure 1–1.

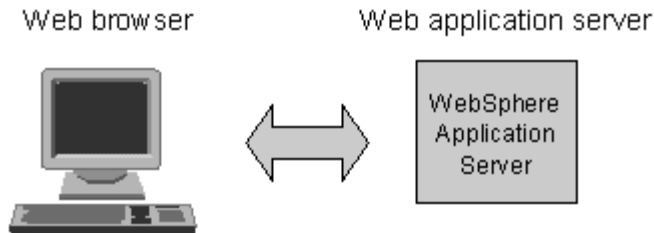


Figure 1–1: User requests from a Web browser

Web applications typically contain a combination of static and dynamic content, such as text and images, and programs that run on the server or in the user’s Web browser. When a user clicks a link or a button on a Web page, the request is sent to the server, which processes it and returns the response (which could be on the page already loaded or on a new page) to the user’s Web browser. Web pages returned to the Web browser consist of tags, drawn from languages such as Hypertext Markup Language (HTML) or JavaScript™, which the browser interprets into user elements on-screen.

As far as the user is concerned, what happens on the Web application server to process a request doesn’t really matter so long as it’s done quickly enough. From your perspective, however, what happens on the server—and the steps needed to make it happen, including what needs to be done to develop Web pages, deploy them to servers, and maintain Web applications—matters quite a bit, because it determines how easy your job is.

One technology frequently used to develop Web applications is a JavaServer Page™ (JSP). A JSP is a dynamic Web page that combines static HTML elements with Java code. The application server compiles and runs the JSP when a user requests the page via a Web browser, and it returns the resulting output to them. Because one of the problems with JSPs is their combination of presentation logic (HTML tags controlling the display of visual elements) with control logic (Java code performing dynamic processing), they can be difficult to maintain for large Web applications.

Faces Web pages are similar to JavaServer Pages, in that they combine HTML tags with Java code and are compiled and run on servers when users request Web pages. The chief distinction of JSF is that Web developers work with Faces UI components on the Web page—rather than with HTML elements. This technique lets you create Web applications using Model-View-Controller (MVC) architecture, as is typical with client-server applications. The JSF framework acts as the controller managing the steps for processing the Web page, including validating, converting, and acting on data that users enter via the Web page. You define your Web application's data in an object model, bind the object model to Faces UI components on the Web page (creating viewable elements), and link actions with particular events, such as selection of links or buttons on the Web page.

The Faces UI components include common elements, such as input and output fields, check boxes, and buttons; but they also include more complex components, such as table grids that you can bind to a collection of data items or a database table. The components provide many useful built-in features: For example, you can customize components both to verify that users supply data in particular input fields and to automatically convert user data into various formats. JSF is readily extensible; you can build your own components that operate within the JSF framework. Other features let you easily control navigation from one Web page to another by specifying which page the user should be directed to when an action is successful—or which other page when it fails. Each component also includes standard event listeners, such as an action listener for button clicks, that apply the processing logic associated with a particular event.

You define the object model for your Web application in JavaBeans, called *managed beans*, which the JSF framework manages automatically, deciding when to create beans, maintaining references to them, and removing references when beans are no longer needed. When a bean property is bound to a particular UI component, JSF automatically displays the property's current value in the Web page and sets the

Model-View-Controller architecture

The Model-View-Controller (MVC) architecture is a well-known design pattern for structuring applications. The model contains state data for the application, such as values entered by users. View refers to the appearance of data on-screen, controlling what users see. The controller determines the application's reactions to events, such as mouse clicks and keyboard input.

property with any newer user-entered value. Specifying the scope for a managed bean determines how long the bean will be available. For example, when you set the scope to “request,” the bean is only available to the current Web page request, but when you set the scope to “session,” the bean is available to all Web pages in the user’s current session, letting you easily pass data from one Web page to another. Scope also tells the JSF framework what a bean’s lifecycle should be.

As you’ll see when you build the Faces Web applications in this book, Rational Application Developer handles most of your work for you. For each Faces Web page you create, the Page Designer tool creates a special Java class, called a PageCode class, containing that page’s processing logic. The PageCode class sets up the context for the current request and gives you the appropriate points to insert the logic for your event listeners. You typically won’t

edit the PageCode class directly; instead, you’ll use the Page Data view to identify the data objects associated with the Web page (such as the JavaBeans that define the application’s object model), the Page Designer to create the view for your Web page, and the Quick Edit view to create event listeners.

In the rest of this chapter, we’ll take you step-by-step through developing a simple Faces Web application, introducing some of the basic Faces UI components (text fields, radio buttons, check boxes, buttons, and so on) and familiarizing you with the Page Designer. The Web application you’ll build consists of two Web pages: an HTML input form for a simple survey (shown in Figure 1–2) and a second Web page echoing the values entered in the input form. To pass data from one Web page to the other, you’ll use a managed bean with “request” scope, binding the Faces UI components of your Web pages to properties in the managed bean.

What’s a class?

A class is a set, or group, made up of members sharing the same state and behavior. Each Java program is a class, and each class has a superclass defining where the class fits into the object hierarchy. Just as you may have inherited certain traits from your parents, a Java class can inherit both state and behavior from its superclass.

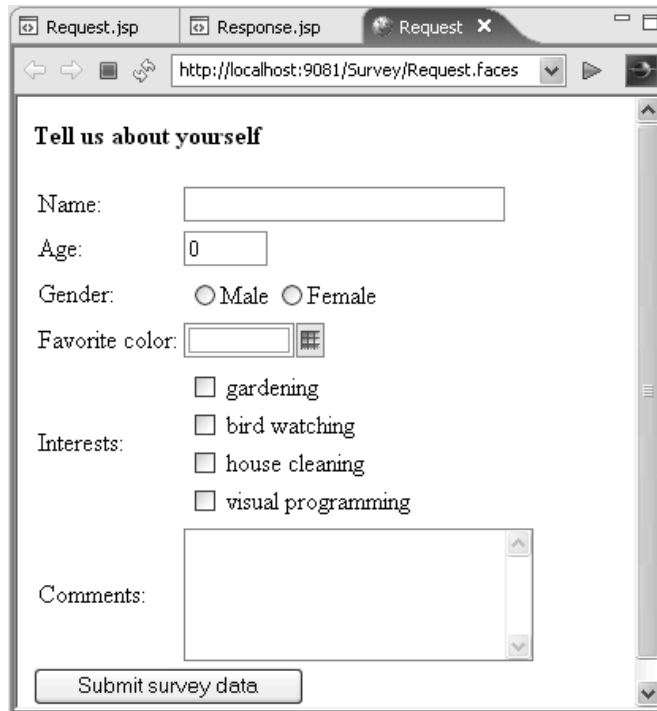
A screenshot of a web browser window showing a survey form. The browser's address bar displays 'http://localhost:9081/Survey/Request.faces'. The form is titled 'Tell us about yourself' and contains the following fields and controls: a text input for 'Name:', a numeric input for 'Age:' with the value '0', radio buttons for 'Gender:' with 'Male' selected, a color picker for 'Favorite color:', a list of checkboxes for 'Interests:' including 'gardening', 'bird watching', 'house cleaning', and 'visual programming', and a text area for 'Comments:'. A 'Submit survey data' button is located at the bottom of the form.

Figure 1–2: The Survey application's input form.

What's an HTML form?

HTML forms collect information entered by users via Web browsers, passing it to Web applications for processing. HTML forms simplify data entry by incorporating text fields, radio buttons, pull-down menus, or other UI elements. When a user clicks the Submit button contained within the form, the user-input values are passed as parameters to an application server program, such as a JSP or JSF servlet. The Web application processes the input and returns a Web page to the user's Web browser with an appropriate response. You specify the URL for the application server program in the form's Action attribute. Within the form, you specify the various HTML tags controlling the display of the input fields and Submit button. HTML forms generated from Faces Web pages have elements created as Faces UI components, which the JSF framework converts into the appropriate HTML tags.

Installing Rational Application Developer

In order to follow along with the steps in this book, you need to install IBM Rational Application Developer 7.0. If you don't already have a copy, you can download a free trial version at <http://www.ibm.com/developerworks/downloads/r/rad>. Follow the installation instructions bundled with the program, first installing the IBM Installation Manager, which you use to install, update, and uninstall product packages as well as manage product licenses. The instructions in this book assume that Rational Application Developer is installed in the default folders for shared resources and package groups.

After installing Rational Application Developer, make sure you use the IBM Installation Manager to install the latest product updates. You'll need Rational Application Developer 7.0.0.2 or later to properly use the samples in this book.

Setting up a Web Project

Before you can create your first Web application, you need to do some preparatory work in Rational Application Developer to specify where your Web application will be saved.

Rational Application Developer is organized into a variety of perspectives, or different ways of looking at a development project based on your role on the development team. In this book, you'll use the Web perspective to build Faces Web applications, and, later, you'll use the Java perspective to build Java applets and applications. Within the Web perspective, your Web applications are organized into projects and folders according to their content, such as package folders for Java classes. You can switch perspectives whenever you need to. Within a particular perspective, you'll also see a variety of different editors and views applicable to the work done within that perspective. Don't worry—we'll explain how each of these works as we go along.

Starting Rational Application Developer

1. To start Rational Application Developer, select **Start → Programs → IBM Software Development Platform → IBM Rational Application Developer → IBM Rational Application Developer**. When you start Rational

Application Developer, a window appears asking which folder you want to use for your workspace (the place where Rational Application Developer saves your work), as shown in Figure 1–3.

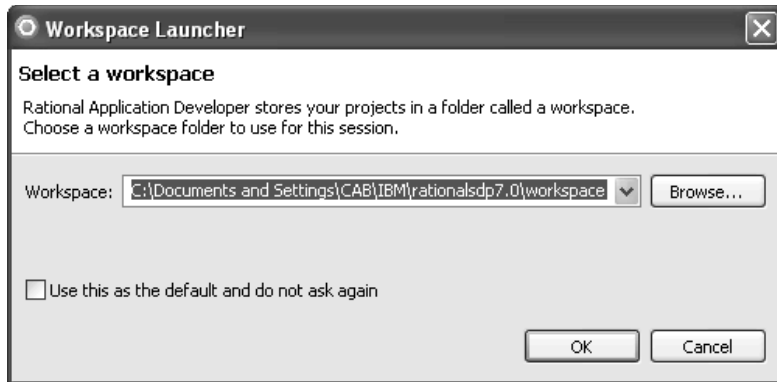



Figure 1–3: The Workspace Launcher window.

Tip: You can use workspaces to organize your work projects by using a separate workspace for each development or customer project that you work with.

2. Leave the workspace name set to the default, and click **OK**. After a few seconds, Rational Application Developer's Welcome page appears. On the Welcome page, you can click the various icons to see an overview of the Rational Software Development Platform, what's new in the current version, or tutorials and samples that will teach you how to use the tool. We'll skip the Welcome information for now, but you can always come back to it by selecting **Help → Welcome** from the Workbench menu.

3. Click the **X** on the Welcome pane tab to close it. The Workbench opens in the J2EE™ perspective. We'll use the Web perspective to work with Faces Web applications, so click the **Open Perspective** icon () in the toolbar tab along the top right of the Workbench, and select **Web** from the list of perspectives. If you don't see Web in the list, select **Other...**, select **Web** in the Open Perspective window, and click **OK**. The Workbench should now look like Figure 1–4.

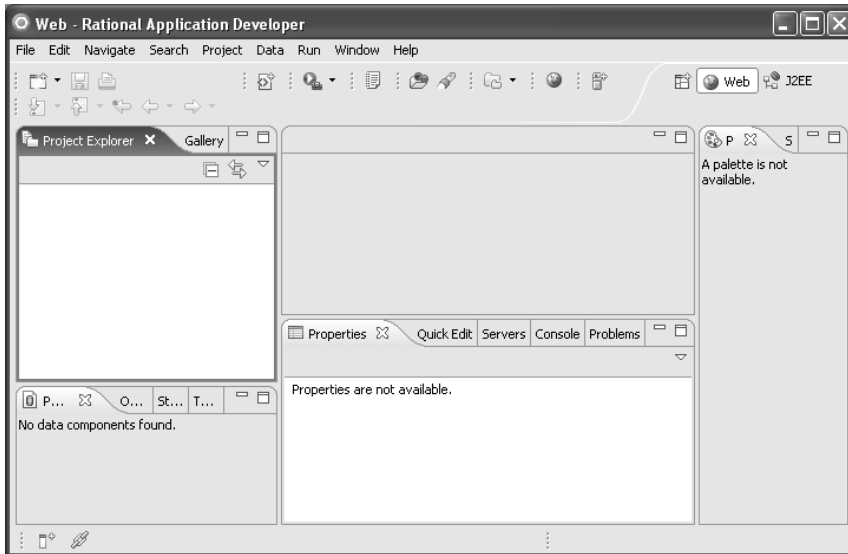



Figure 1–4: The Rational Application Developer Workbench.

Notice that the Workbench is divided into several toolbars and work areas. Let's take a minute to get familiar with the layout before we continue.

The toolbar along the top right tab shows an icon for each perspective that's currently open, such as the Web perspective ( Web) currently appearing in the Workbench. You can have multiple perspectives open at any given time—to switch from one to another, just select its icon. You can also customize this portion of the toolbar by moving it to the left edge of the window or removing the text descriptions now accompanying each icon. To customize, right-click anywhere in the toolbar tab and select the option you want from the context menu.

Along the top of the Workbench are a series of pull-down menus and a toolbar of common commands used to work within the particular perspective you are using. Because the menus and toolbar match the functions available in a given perspective and the editors being used, they will change as you switch from one perspective to another and work with different editors. The toolbar contains a subset of the actions available from the pull-down menus; you can customize it to add, reorganize, or remove toolbar actions according to your preferences.

The work areas within the Workbench are also tailored to the functions available in a given perspective and the editors and views being used. In the Web perspective, you will use the Project Explorer to navigate the various folders and files attached to your Web projects. When you open a file, the editor for its file type appears in the center of the Workbench. For example, the Java Editor appears when you are working with Java source files, and the Page Designer appears when you are working with Web pages. Any associated views appear in the other work areas, such as the Properties view and the Palette (which you'll use to build Faces Web pages). Selection tabs in each work area let you switch from one view to another—for example, to select the Properties view to work with the attributes of the various Faces components within a Web page. You can move views from one work area to another if you prefer a different placement. If you ever find that you've customized a perspective more than you intended, you can always select **Window → Reset Perspective** to restore the appearance of the current perspective to its original layout.

Creating the Project

Next, you'll use the Web perspective to create a new Web project to contain your Web application. You'll create the project as a Dynamic Web project—that is, one that contains dynamic content, such as the Java code for your Web application. You'll also specify an EAR project (called an *Enterprise Application*) used for deploying and testing the Web project on an application server.

1. To create the project, select **File → New → Dynamic Web Project** in the Workbench menu.
2. Enter a project name of **Survey**. Notice that the project location is set to the default for your workspace (the directory specified when you first started Rational Application Developer).

3. Make sure the **Add project to an EAR** check box is checked, and enter **SurveyEAR** for the EAR project name.
4. Select **WebSphere Application Server v6.1** for the target runtime, and select **Faces Project** for the configuration. These options specify that the Web pages in your project will run on WebSphere Application Server v6.1 (the Rational Application Developer Test Environment) and that the project will include the necessary class libraries to create Faces Web pages. The window should look like Figure 1–5.

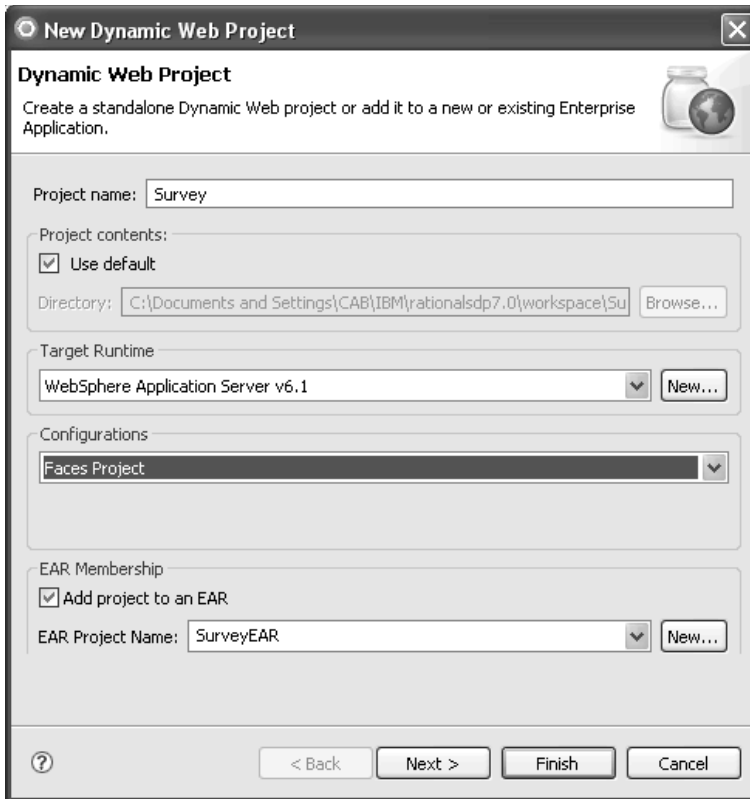


Figure 1–5: Creating a Dynamic Web project.

5. Click **Finish**.

In the Project Explorer, click the plus (+) signs to expand the content of folders in your Survey and SurveyEAR projects, as shown in Figure 1–6.

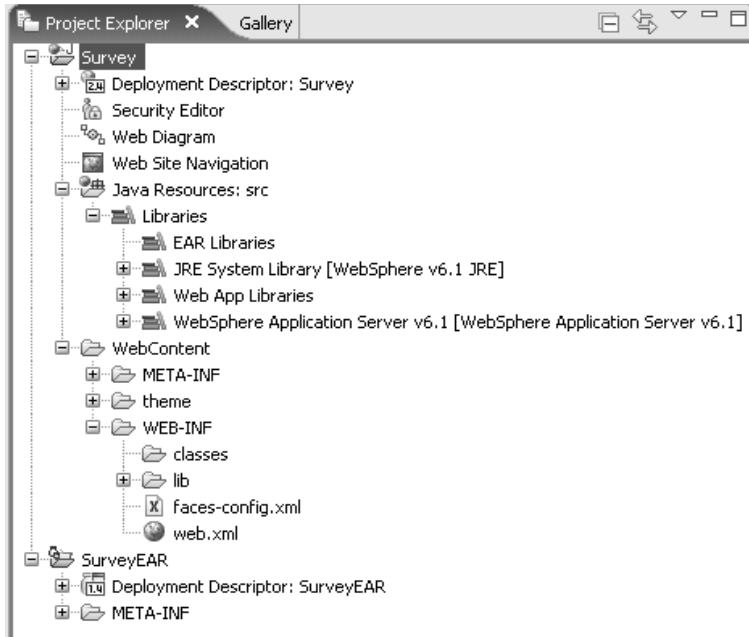


Figure 1–6: The Project Explorer.

Rational Application Developer automatically created a default folder structure for your projects and placed several control files in the folders. The project structure is organized according to the specifications for packaging J2EE applications. A Web project in which you place the actual content for your Web applications uses the standard structure for Web Archive (WAR) files, also called a *Web module*. An Enterprise project uses the standard structure for Enterprise Archive (EAR) files. When you create a Dynamic Web project, Rational Application Developer associates the Web project with an Enterprise project and places the Web project's WAR file in the Enterprise project's EAR file, which you can deploy to an application server.

You can associate multiple Web projects with a particular Enterprise project, combining related Web modules into one large J2EE application. Luckily, you don't have to remember all the details for WAR and EAR files, because Rational

Application Developer does the packaging for you. Just place your Java source files in Java packages within the **Java Resources: src** folder, and put the content for your Web pages in the **WebContent** folder. Rational Application Developer automatically updates the necessary configuration and deployment information for you.

Note: When you create a Dynamic Web project, the Web diagram editor automatically opens in the Workbench. We'll show you how to use the Web diagram editor in the next chapter. For now, just close the editor.

Creating the Web Application

The Survey application you'll build in this chapter consists of two Web pages: an input form, where the user enters their survey information, and a second Web page that echoes the values users enter. To pass data from one Web page to the other, you'll use a Faces managed bean with a "request" scope, binding the bean's properties to Faces UI components in the Web pages. This JavaBean will serve as the object model for your Web application. (In a real application, you'd save the survey information to a database of some sort, but we've saved database work until later in the book.)

Creating the JavaBean

Follow these steps to create the JavaBean for your Web application:

1. To create the Java package and class for your JavaBean, right-click the **Java Resources: src** folder in the Survey project, and select **New → Class** from the pop-up menu.
2. Enter **SurveyData** for the class name and **samples** for the package name so that the window looks like Figure 1–7.

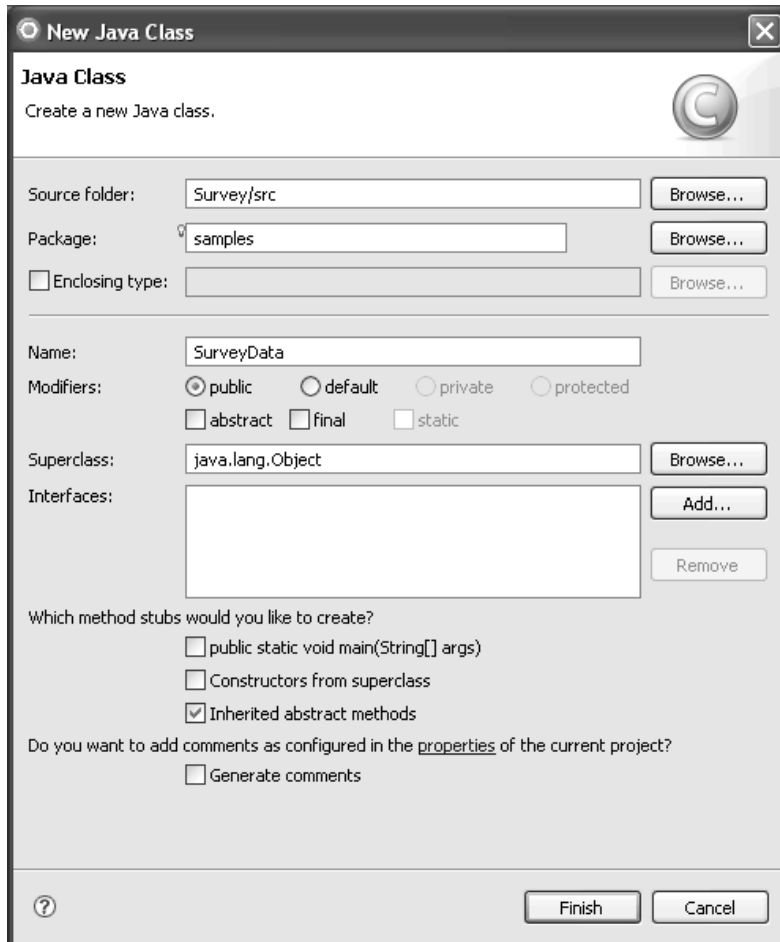


Figure 1–7: Creating the JavaBean for the Survey application.

Tip: Managed beans that will be stored as either “session” or “application” in scope also need to implement the `java.io.Serializable` interface so that they can persist on the server across user requests. An *interface* is a template defining how particular functions should be implemented. Interfaces don’t contain any executable code—they define patterns for other programs to follow when implementing a function. The `java.io.Serializable` interface doesn’t actually define any methods classes need to implement; it just identifies objects that can be serialized (saved in persistent storage on the server) across user requests.

3. Click **Finish**.
4. The new class opens in the Java Editor. Your JavaBean needs six properties—one for each of the input values in the survey Web page. To define the properties, enter the class variable declarations shown in bold in Listing 1–1.

```
package samples;

public class SurveyData {
    private String name;
    private int age;
    private String gender;
    private String favoriteColor;
    private String[] interests;
    private String comments;
}
```

Listing 1–1: Properties for the SurveyData JavaBean.

Tip: You'll find that you'll have to enter very little code by hand, because Rational Application Developer generates so much of it for you. If you're new to Java programming or just need a refresher on basics of the Java programming language, look in the appendix.

5. To add getter and setter methods for the properties, position the cursor after the last class variable (this is where the new methods will be added), right-click, and select **Source → Generate Getters and Setters...** from the pop-up menu.
6. In the Generate Getters and Setters window, click **Select All**, causing the window to look like Figure 1–8.

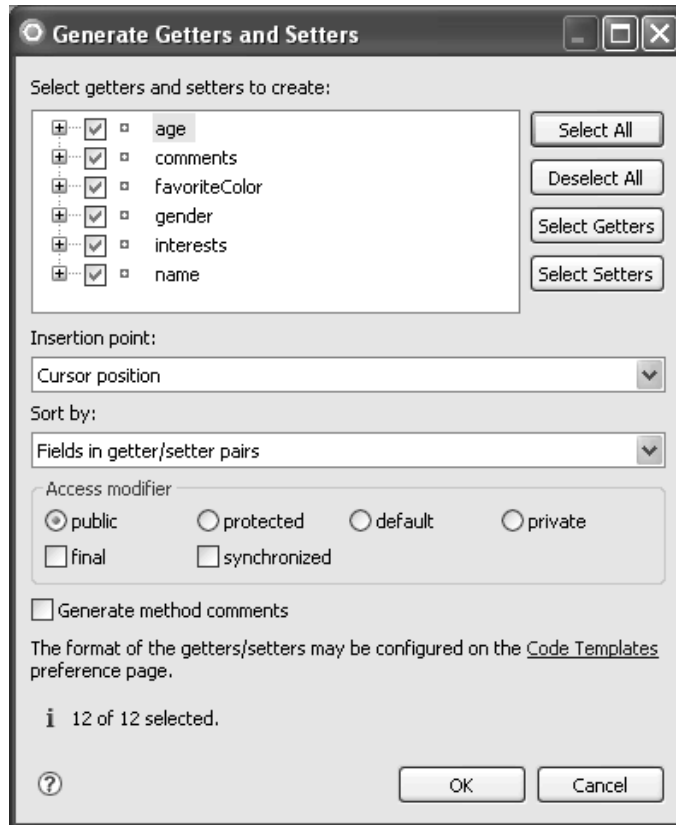


Figure 1–8: Generating getter and setter methods for the JavaBean.

7. Click **OK**.
8. To save your changes, right-click anywhere in the Java Editor and select **Save** from the pop-up menu.
9. You're finished working with this class, so close the Java Editor.

Creating the Input Form Web Page

Next, you'll create an input form where users can enter information. In the following sections, you'll add Faces UI components to the Web page and bind the components to properties in the JavaBean you just created.

1. In the Project Explorer, right-click the **WebContent** folder for the Survey project and select **New** → **Web Page** from the pop-up menu.
2. Enter **Request.jsp** for the file name, causing the window to look like Figure 1–9.

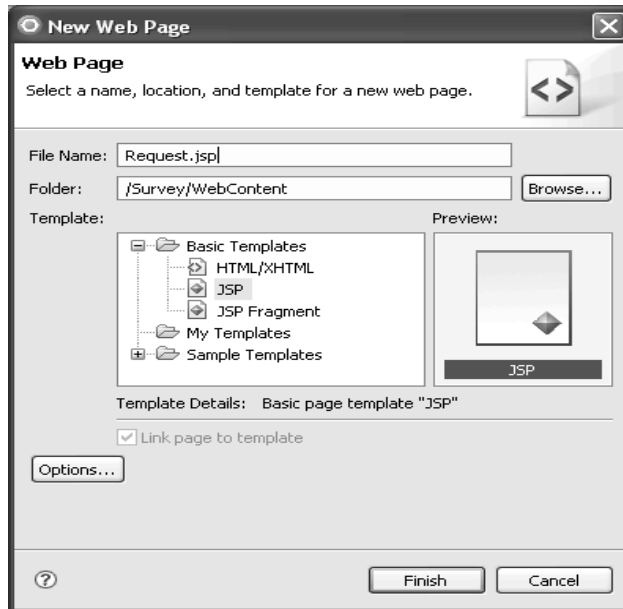


Figure 1–9: Creating a Faces Web page.

3. Click **Finish**.

The Request.jsp file opens in the Page Designer, as shown in Figure 1–10. (If you see HTML source code instead of the design surface, click the **Design** tab.)

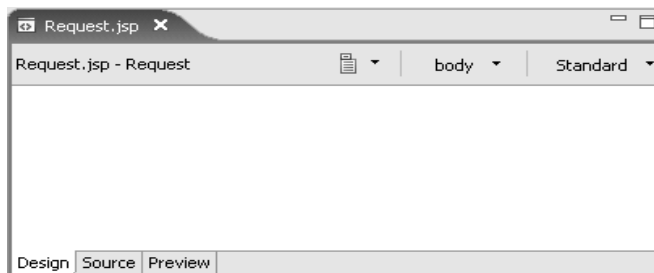


Figure 1–10: A new Faces Web page in the Page Designer.

Tip: To view the HTML and Faces JSP source for your Web page, click the **Source** tab in the Page Designer. To see a preview of what the HTML elements in your Web page will look like in a Web browser, click the **Preview** tab. Note, however, that the Preview tab won't show you what the final Faces UI elements will look like, because it's not actually running in a real application server.

Using the Page Designer

The Page Designer works much like any other graphical editor. To add or change text in the Web page, just type on the design surface. To change the attributes of a particular text string, select the text on the design surface and assign the value you want in the Properties view, which appears at the bottom of the Workbench. Its layout changes based on the element currently selected on the design surface.

To add complex elements to the Web page, such as Faces UI components, select the element you want in the Palette (Figure 1–11), drop it on the design surface, and set its attributes in the Properties view. The Palette contains all common elements of Web pages and organizes them in related groups called *drawers*. You click the name of a drawer to open it.

Tip: The Properties view (click the **Properties** tab to see it) and the Palette should automatically appear in the Workbench when you open a file with the Page Designer. To see a view that isn't open, use **Window → Show View**.

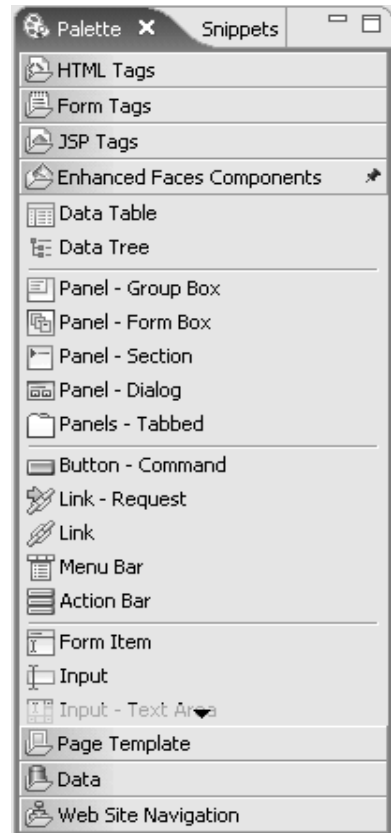



Figure 1–11: The Enhanced Faces Components drawer in the Palette.

Also notice the Page Data view shown in Figure 1–12. You’ll use this view to add managed beans to your Faces Web pages.

Now, let’s use the Page Designer to create your Web page. For your first Web page, you’ll build the Faces UI components manually. We’re taking a slow, step-by-step approach to

building this Web page, allowing us to examine each component in detail, making sure you understand its use. Later in this chapter, you’ll see how Page Designer’s shortcuts simplify work when you build the response Web page.

1. In the design surface, enter **Tell us about yourself**.
2. In the Properties view, select **Heading 4** as the Paragraph value for the text.
3. Click **HTML Tags** in the Palette to open the drawer for HTML elements.
4. Select a **Table** ( **Table**) in the Palette’s HTML drawer and drop it on the design surface below the heading (we’ll cover Web page layouts in more detail later in this book, but for now you’ll place your Faces UI components in an ordinary HTML table to lay out the elements on the Web page). When prompted to enter the number of table rows and columns, set the rows to **6**, the columns to **3**, and the border width to blanks, as shown in Figure 1–13. Setting the border width to blanks means that the user won’t see the table borders on the Web page—it will just be used to lay out the elements within the table. Notice that you can set some table properties here, or, if you prefer, you can set the properties *after* dropping the table on the design surface, by editing the values in the Properties view.

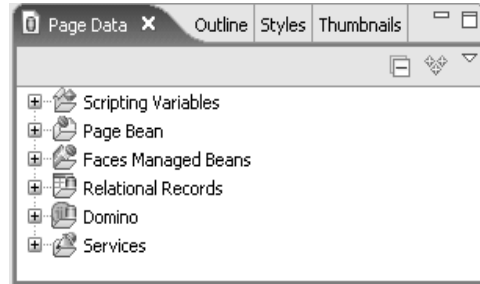


Figure 1–12: The Page Data view.

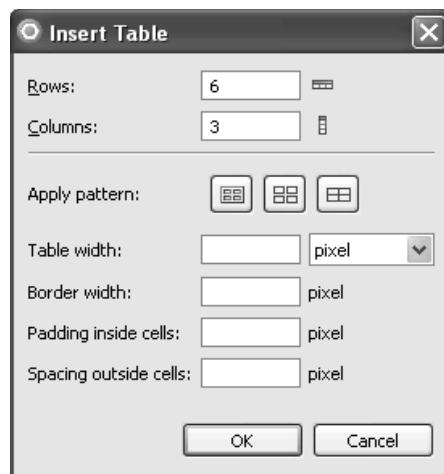


Figure 1–13: The Insert Table window.

5. Click **OK**.
6. Now you'll set the static text values in the table. In the design surface, enter **Name:** for the text in the first table cell (the first row and first column). If the first cell is not already selected, just click in it to select it.
7. Select the next cell in the first column on the design surface (the second row and first column). Enter **Age:** for the text in this cell.
8. Repeat to add the remaining labels for **Gender:**, **Favorite color:**, **Interests:**, and **Comments:**, so that the design surface looks like Figure 1–14.

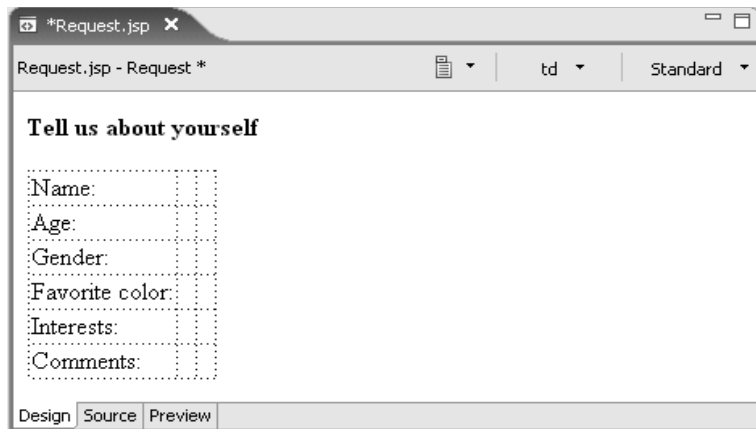
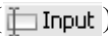


Figure 1–14: Setting the static text for the HTML table.

9. So far you've added only ordinary HTML elements to your Web page. Now you'll add Faces UI components to make things more interesting. Click **Enhanced Faces Components** in the Palette to open the drawer for Faces UI components.
10. Select an **Input** component ( Input) in the Enhanced Faces Components drawer and drop it on the design surface in the second column of the first table row.

11. Notice that tabs appear in the Properties view to set property values for this `inputText` component, including validation options. The default format for an input component is to accept string data, which is how users' names are received. Set the component's Id to **name** (a descriptive label), and enter **30** for the width (the size of the field on the Web page) so that the Properties view looks like Figure 1–15. Notice that the component also has a Value property—as you'll see in a moment, this is how you bind the component to a property in your `JavaBean`.

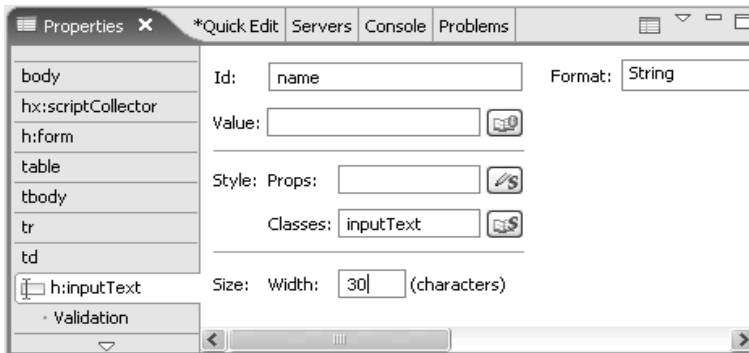



Figure 1–15: Setting properties for the input field.

12. Select another **Input** component in the Enhanced Faces Components drawer and drop it on the design surface in the second column of the second table row.
13. In the Properties view, enter **age** for the Id and **5** for the width.
14. Select **Number** for the format of the age input component. Notice that when you change the format to Number, additional options appear, allowing you to select the type of number the field will contain as well as a mask pattern and input assistants such as spinners or sliders. Leave the type set to **Decimal** and check the **Integer only** check box.
15. Select a **Radio Button Group** component ( Radio Button Group) in the Enhanced Faces Components drawer and drop it on the design surface in the second column of the third table row.
16. In the Properties view, enter **gender** for the Id, and leave the direction set to **<default>**. The default is to show the choices horizontally.

17. To set the first radio button choice, click **Add Choice** and enter **Male** for both label and value. The label appears on the Web page, but the value gets passed to your application logic in the Faces Web page. For this example, we'll use the same strings for both.
18. Click **Add Choice** again and enter **Female** for the label and the value. The design surface should now look like Figure 1–16. Notice that visual cues are included in the two input components to tell you whether they accept string or number data.

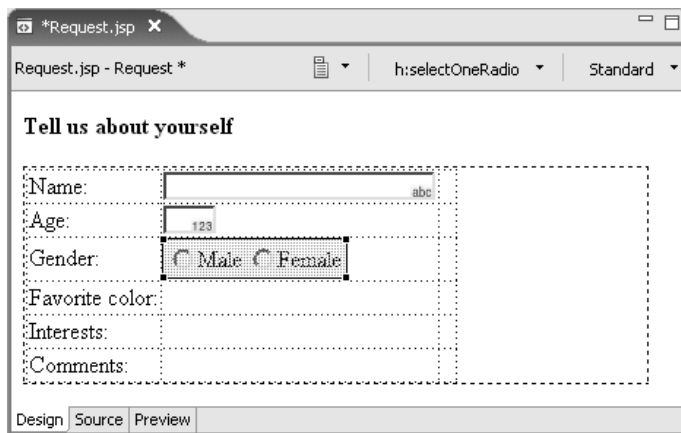






Figure 1–16: Radio button group added to the design surface.

19. For the control where the user selects their favorite color, we'll use a special component that Rational Application Developer provides. Select a **Select–Color** component ( Select - Color) in the Enhanced Faces Components drawer and drop it on the design surface in the second column of the fourth table row.
20. In the Properties view, enter **favoriteColor** for the Id.
21. Check the **Show color names in control** check box and click **Add default colors**. This loads the choices in the control's table of label/value pairs. Notice that, for this control, the value the user sees in the Web page includes the name of the color, but the internal value for the selection is the red-green-blue value used in HTML controls. You'll see in a moment how we use the internal value to display the selected color in the response Web page.

22. To insert a group of checkboxes indicating possible user interests, select a **Check Box Group** component ( Check Box Group) in the Enhanced Faces Components drawer and drop it on the design surface in the second column of the fifth table row.
23. In the Properties view, enter **interests** for the Id, and set the direction to **Vertical**.
24. Use the Add Choice button to add choices for the range of interests shown in the survey: gardening, birdwatching, house cleaning, and visual programming. Set identical labels and values for all choices.
25. To add the input field for survey comments, select an **Input – Text Area** component ( Input - Text Area) in the Enhanced Faces Components drawer and drop it on the design surface in the second column of the sixth table row.
26. In the Properties view, enter **comments** for the Id, **25** for the width, and **5** for the height.
27. The input fields have all been added to your Web page, but you need one more component—a command button allowing users to submit the form. Select a **Button – Command** component ( Button - Command) in the Enhanced Faces Components drawer and drop it on the design surface below the table.
28. In the Properties view, click the **Display options** tab and enter **Submit survey data** for the button label. The design surface should now look like Figure 1–17.

Curious readers: If you click the **Source** tab at this point, you'll see all sorts of special tags like `h:inputText` and `hx:commandExButton`. These tags are the Faces UI components used in your Web page—some from the standard JSF component library and others from an IBM extension library that improves upon the standard components. Luckily, Page Designer handles all this for you, so you don't need to worry about it.

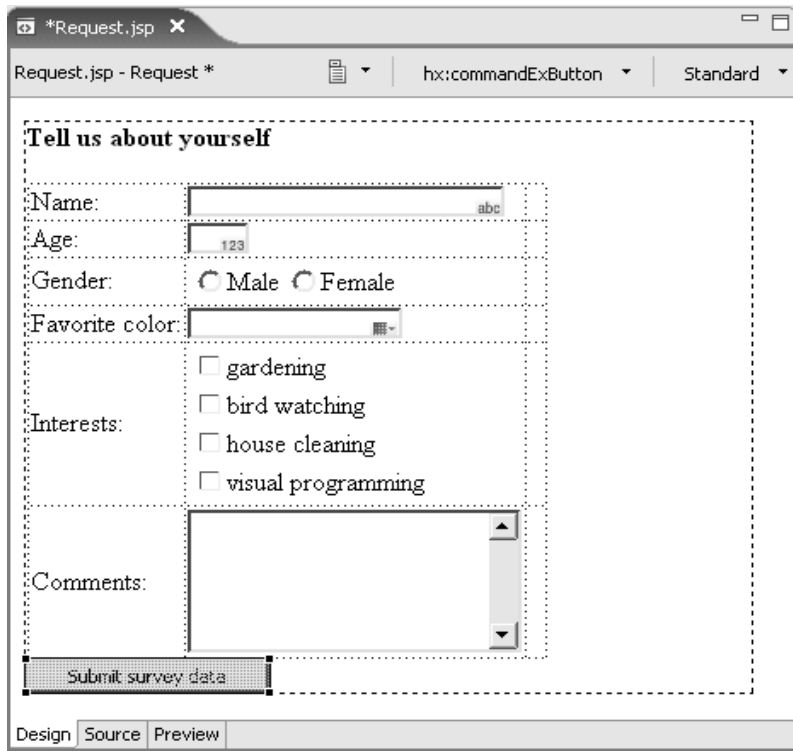


Figure 1-17: All input components added to the Web page.

Binding UI Components to JavaBean Properties

So far you've built the user interface (the view) for your Web application. Now you'll bind the UI components to properties in your JavaBean (the model), allowing you to pass data from the input form to the response Web page.

1. In the Page Data view, right-click **Faces Managed Beans** and select **New → Faces Managed Bean** from the pop-up menu.
2. In the Add JavaBean window, enter **surveyData** for the name and **samples.SurveyData** for the class (you can click the icon next to the class name field to select it from a list if you'd rather not type in the full class name)
3. Check the **Make this JavaBean reusable** check box. This makes your bean a Faces Managed Bean, which lets you use it within your Web application according to the scope you assign it: request, session, or application.

For this example, you'll use the request scope to make the bean available to Web pages attached to the current request (that is, to make it available to the survey input form Web page and its response Web page). Later in this book, you'll use session and application scopes.

4. Enter **Survey application data** for the description and select **request** for the scope, causing the window to look like Figure 1–18. Note that you can also initialize bean properties from this window.



Figure 1–18: Adding a managed bean to the Faces Web page.

5. Click **Finish**.
6. Click the plus sign next to `surveyData` to expand the contents. The Page Data view should now look like Figure 1–19.

Tip: If you don't see the managed bean in the Page Data view, click the Menu dropdown in the upper right corner of the Page Data view and then select **Show Faces Managed Beans – From /WEB-INF/faces-config.xml** from the menu.

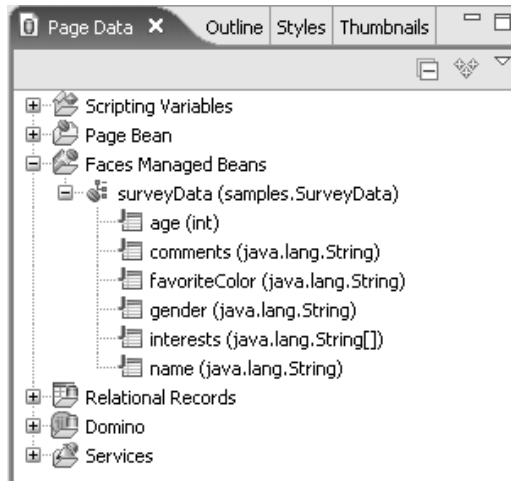


Figure 1–19: Page Data view with `surveyData` JavaBean added.

7. Select **age** in the Page Data view and drag and drop it onto the age input field on the design surface (make sure you drop it on the second input component, associating it with the correct control—you’ll see a prompt that says “Drop here to bind ‘age’ to the control ‘age’ ”). Notice that the control’s value property in the Properties view is now bound to the JavaBean property.

Tip: In the Properties view, make sure that the **age** input field is still set to Number format and that the **Integer only** check box is checked (binding the property to the component may cause it to be reset to String format). Usually, you won’t run into problems with attributes getting reset, because you’ll be letting Page Designer automatically build Web pages for you. This step is necessary for this example because of the “scenic route” we’re taking in setting up the Web page.

8. Repeat to bind `comments`, `favoriteColor`, `gender`, `interests`, and `name` in the Page Data view to their associated UI controls so that the design surface looks like Figure 1–20. Notice that some of the UI components visually cue you about the values they’re bound to. Also notice that, in some cases, the bean property is bound to the value appearing in the Web page (`name`, `age`, and `comments`), but, in other cases, the bean property is bound to the values users select on the Web page (selections for `gender`, `favoriteColor`, and `interests`).

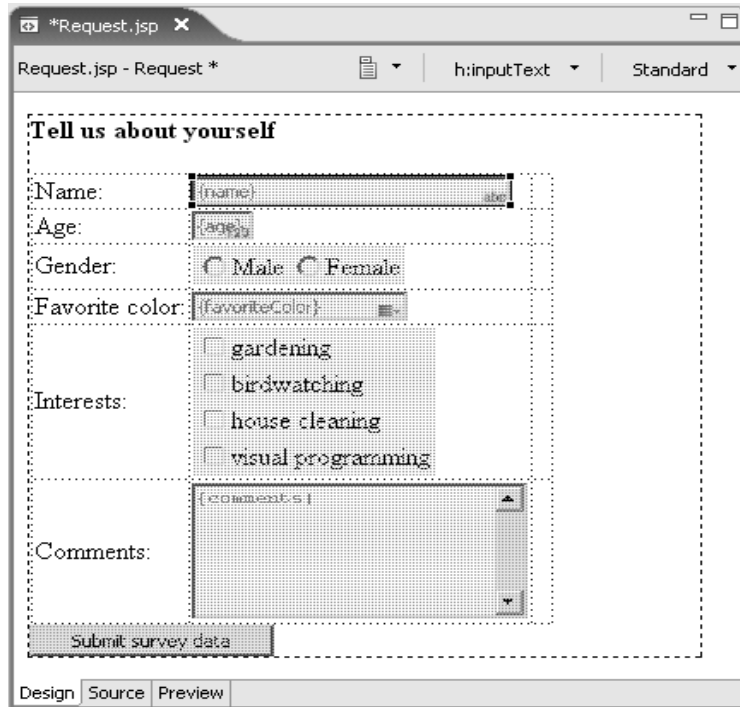


Figure 1–20: Binding UI components to JavaBean properties.

Adding Validation to the Input Form

One of the most common tasks in form-based Web applications, including this survey example, is validation of users' inputs, making sure users enter all required data and do so in correct formats. As you'll see in this section, Faces UI components can do many validation tasks for you.

1. Select the first input field on the design surface (the input field for the user's name) so that its properties appear in the Properties view.
2. Click the **Validation** tab for this control in the Properties view (you may need to scroll down to see the tab) and check the **Value is required** check box. This says that the user must enter a value in the input field—but where, and how, will error messages be issued to notify users of missing values? JSF gives you two options. You can have a single error message component on the Web page to display errors for all Faces UI components on that page, or you can have individual error message components associated with particular

UI components (you can also divide labor among a number of individual error message components and one general one). For this example, we'll use individual error messages for the first three UI components in the Web page and one general error message component for everything else.

3. To have Page Designer automatically add an error message component for the input field, check the **Display validation error messages in an error message control** check box. (This check box is in the Properties view—you may need to scroll to the right to see it.)
4. The error message control is automatically added next to the input field. To tidy up the Web page layout, we'll display all error messages associated with individual Faces UI components in the third column of the HTML table. Select the **Error Message control** on the design surface and move it to the third column of the first table row so that the design surface looks like Figure 1–21. The Page Designer gives you visual cues showing where components will be dropped, so look for a vertical bar in the table cell where you want to move the component.

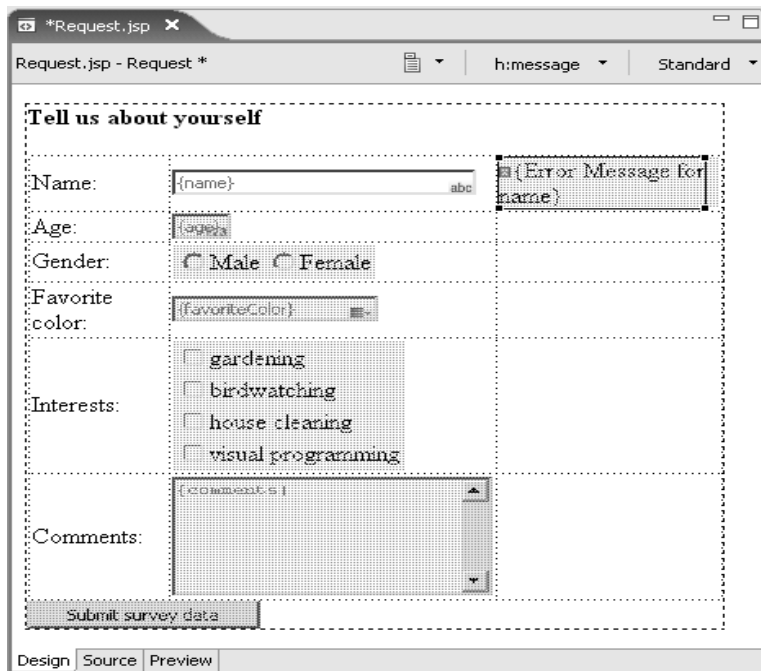





Figure 1–21: Error message control added for the name input field.

5. You'll also want to make any error messages stand out so that users will notice them. In the Properties view, click the **Browse...** icon () next to **Style: Props**.
6. In the New Style window, select **Red** from the Color pull-down menu and click **OK** to close the window.
7. Select the input field for users' ages in the design surface. In the Properties view, check the **Value is required** check box and enter **1** as minimum and **120** as maximum. This says that users will only be allowed to enter ages from 1 to 120.
8. To have Page Designer automatically add an error message control for this input field, check the **Display validation error messages in an error message control** check box.
9. Move the new error control to the third column of the second table row and set its color to red.
10. Another way you can add error message controls to the Web page is by selecting the control in the palette, dropping it on the design surface, and setting its properties to associate it with a particular Faces UI component. Let's try it out to see how it's done. Select a **Display Error** component ( Display Error) in the Enhanced Faces Components drawer and drop it on the design surface in the third column of the third table row (make sure you select the Display Error component—not the Display Errors component; we'll use Display Errors in a moment).
11. In the Properties view, select **gender** in the Id pull-down for the component, associating the error control with the gender UI component.

Tip: There are two Id values in the Properties view—select gender from the Id pull-down and leave the error message control's Id set to message1.

12. Set the error control's color to red.
13. Select the **gender radio button group control** in the design surface. In the Properties view, click the **Validation** tab for this control, and check the **Value is required** check box. Notice that the **Display validation error messages in an error message control** check box is already checked—that's because

you associated the gender component with the error message control you just added to the Web page. Now, if the user doesn't select either of the choices in this radio button group, an error message will automatically be displayed.

14. Finally, we'll add a general error message control to handle any errors issued by the other UI components as well as any application-level messages that might occur (in other words, messages not associated with any specific UI component). Even though your Web page might include individual error controls for each input field, it's a good practice to include a catch-all control to make sure the user is notified in case any other errors occur (this is also helpful while you're testing your Web applications, making sure you don't miss any error conditions). Select a **Display Errors** component ( Display Errors) in the Enhanced Faces Components drawer and drop it on the design surface below the submit button.
15. In the Properties view, select **Show only error messages not associated with a specific component** and set the control's color to red. The design surface should now look like Figure 1–22.

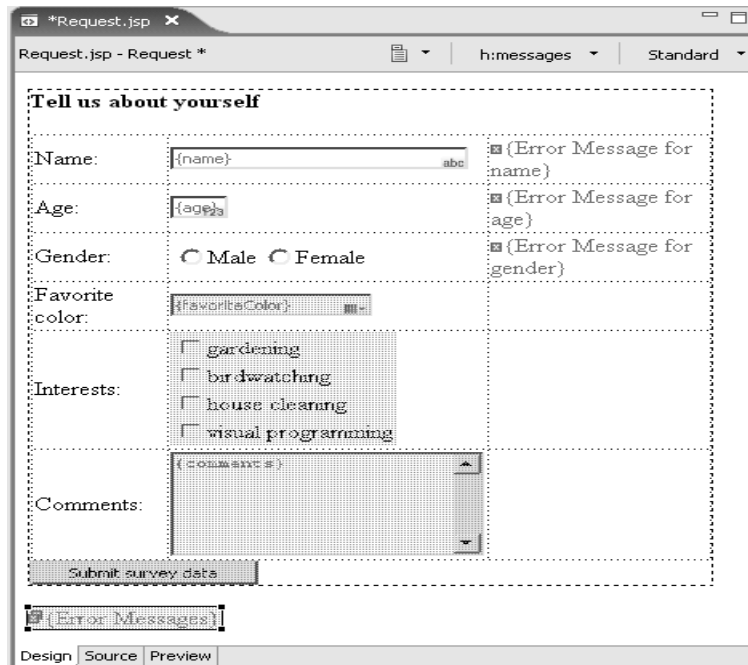




Figure 1–22: Error controls added to the Web page.

You just added validation and error reporting for all the form's input fields without writing a line of code!

Creating the Response Web Page

Now that your input form is ready, you'll create a response Web page to echo the values users enter in the form. In the next section, you'll define the navigation rules to go from one Web page to the other, and you'll save your changes.

1. In the Project Explorer, right-click the **WebContent** folder for the Survey project and select **New → Web Page** from the pop-up menu. Enter **Response.jsp** for the file name and click **Finish**.
2. Your new Web page opens in the Page Designer. In the design surface, enter **This is what you told us**. In the Properties view, select **Heading 4** as the Paragraph value for the text.
3. For the input form Web page, you manually added Faces UI components to the Web page, allowing us to examine the controls in detail. This time, however, you'll let Page Designer handle it for you. The managed bean containing the survey application data is already available to this Web page, because you created it as a reusable JavaBean for the first Web page. In the Page Data view, select **surveyData** and drop it on the design surface below the heading—you'll see a prompt saying "Drop here to insert new controls for "surveyData"".
4. In the Insert JavaBean window, make sure **Displaying data (read-only)** is selected for all controls to be created.
5. To arrange the fields in the same order they appear in the input form Web page, select the field names in the table and use the up () and down () arrow icons to move them up or down in the list until the window looks like Figure 1–23. You can also change the labels appearing on the Web page to make them more readable.

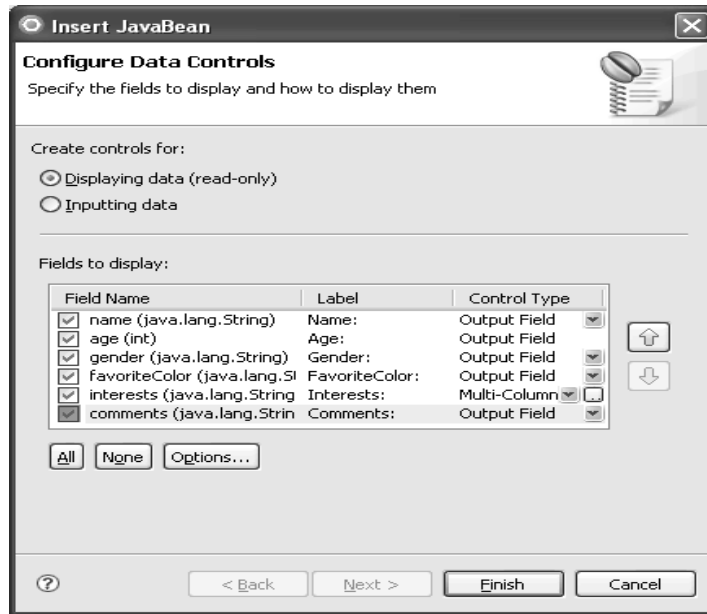



Figure 1–23: Adding controls to the response Web page

6. Click **Finish**. Notice that the generated controls include all the properties in your JavaBean, plus an error messages control to display any error messages associated with this Web page.
7. The generated controls also include an outer table, to format the layout, and an inner table, for the list of selected interests. We don't need the header on the table of selected interests, so select **Interests** (the string appearing in the table header) on the design surface, and, in the Properties view, change the Value to blanks. In the design surface you'll see "outputText" in the table header—this is just the name of the Faces UI component used to display the table header. As you'll see in a moment, that header area won't appear when the Web page displays in a Web browser.
8. To display the user's favorite color, we'll set the associated output field's text and background colors to the selected value. Select the `{favoriteColor}` output field on the design surface. In the Properties view, enter **color: #{surveyData.favoriteColor}; background-color: #{surveyData.favoriteColor}** for the Style: Props value. This specifies that the output field's text and background color values will be taken from whatever color

the user selects on the first Web page. If you've used style sheets to format text in Web pages, you'll recognize the syntax for specifying colors and background colors—"color: *value*; background-color: *value*", where *value* is the name of the color to be used. The special notation we're using to specify the color values is called a *value-binding expression*. These expressions cause the `favoriteColor` property in the `surveyData` managed bean to be used as the color.

9. This Web page will include a link back to the input form, so select a **Link – Request** component ( Link - Request) in the Enhanced Faces Components drawer and drop it on the design surface, below the error messages control.
10. In the Configure Label window, enter **Return to survey** for the label and click **OK**. This assigns a label to the hyperlink—you'll set the actual link in a moment. The design surface should look like Figure 1–24.

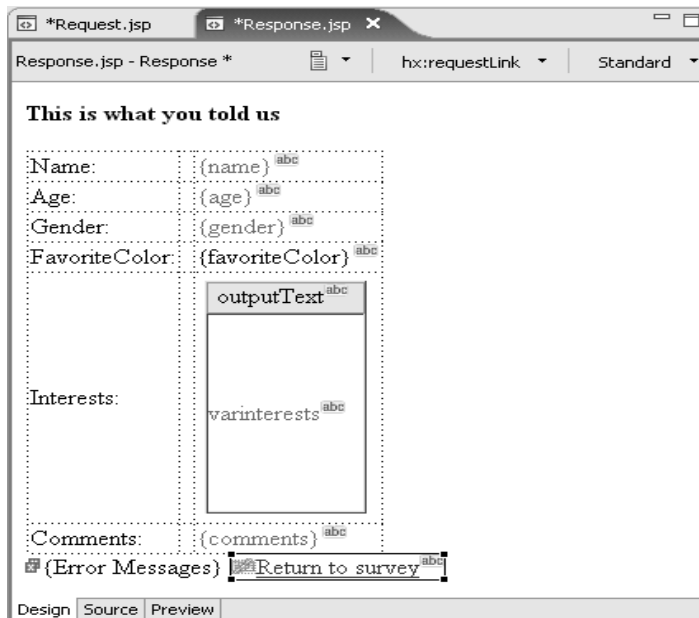



Figure 1–24: Response Web page with all controls added.

Easy, wasn't it!

Setting up the Navigation Rules

By default Faces Web pages remain on the current page when users submit requests (as when users click buttons on Web pages). Navigation from one Web page to another is specified by setting outcomes (or actions) for Web pages and then defining navigation rules telling JSF where to go when particular outcomes occur. You can set outcomes as return values in ActionListener methods (something you'll do later in this book), but, since we try to avoid writing code whenever possible, we'll show you another technique. For this Web application, an outcome of "success" on the input form should take users to the response form, but an outcome of "return" on the response form should return users to the input form. Let's see how this is done.

1. Switch back to **Request.jsp** in the Page Designer and select the **submit button** on the design surface. In the Properties view, click the **All Attributes** icon (). This shows a list of *all* attributes associated with the component, not just the common ones the normal tabs display.
2. Enter **success** for the value of the **action** attribute so that the Properties view looks like Figure 1–25.

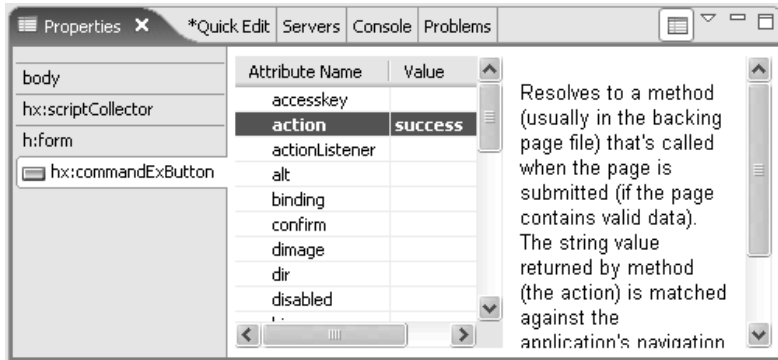



Figure 1–25: Setting the action for the input form Web page.

3. You've set the outcome value for the button—now you'll define the navigation rule. Click the **All Attributes** icon again to return to the normal Properties view.
4. In the Properties view, click the **hx:commandExButton** tab and click **Add Rule...** (You may need to scroll right to see the button.)

5. In the Add Navigation Rule window, select **Response.jsp** for the page to go to, select **The outcome named:** and enter **success** for the outcome name so that the window looks like Figure 1–26.
6. Click **OK**.
7. You're finished with `Request.jsp`, so select **File → Save** to save your changes.
8. Switch back to **Response.jsp** in the Page Designer and select the link () in the design surface (be sure to select the link—not its label).
9. In the Properties view, click the **All Attributes** icon and enter **return** for the value of the action attribute.
10. Now that you've set the outcome value for this link, you'll define its navigation rule. Click the **All Attributes** icon again to return to the normal Properties view.
11. In the Properties view, click **Add Rule...**
12. In the Add Navigation Rule window, select **Request.jsp** for the page to go to, select **The outcome named:**, and enter **return** as the outcome name.
13. Click **OK**, and select **File → Save** to save your changes.

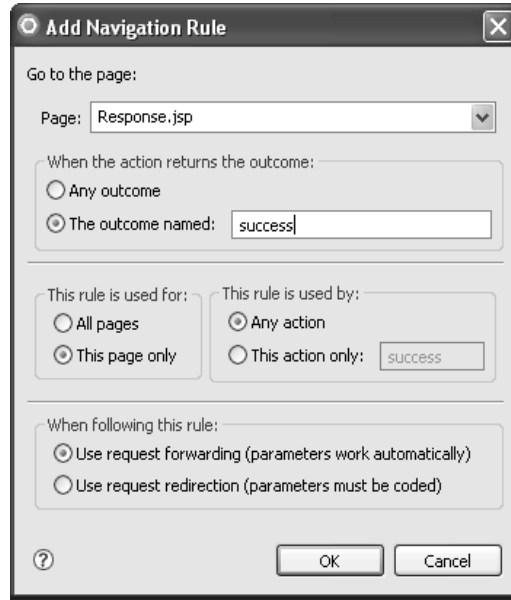


Figure 1–26: Setting a navigation rule for the input form.

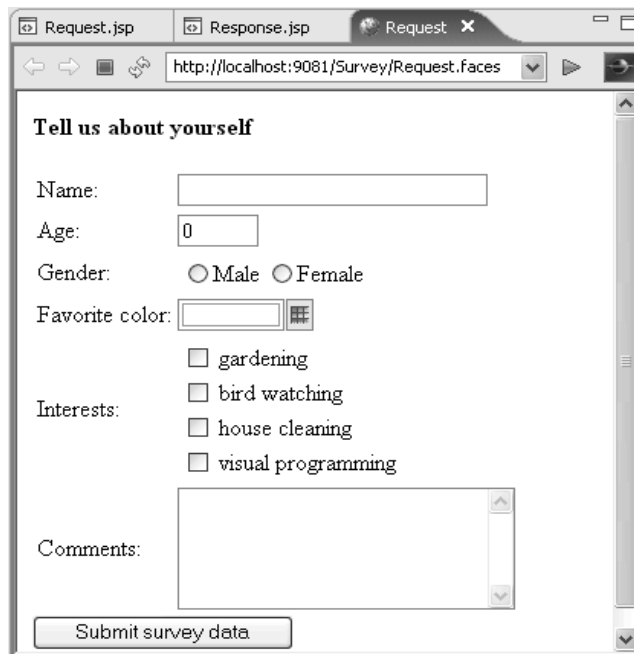
Running the Web Application

Follow these steps to run your Web application:

1. Right-click **Request.jsp** in the Project Explorer and select **Run As → Run on Server** from the pop-up menu.

- When prompted to select a server to launch, make sure that **WebSphere Application Server v6.1** is selected (this is the WebSphere Test Environment that comes with Rational Application Developer) and click **Finish**. Be patient—Rational Application Developer has to start the server and publish your Web application before your Web application can start in a Web browser window (as shown in Figure 1–27).

Tip: You can skip the Server Selection window when running Web applications in this Web project in future by checking the **Set server as project default** check box.




The screenshot shows a web browser window with the address bar displaying `http://localhost:9081/Survey/Request.faces`. The page content is a survey form titled "Tell us about yourself". The form contains the following elements:

- Name:** A text input field.
- Age:** A text input field containing the value "0".
- Gender:** Two radio buttons labeled "Male" and "Female".
- Favorite color:** A color selection box with a small color palette icon.
- Interests:** A list of four checkboxes: "gardening", "bird watching", "house cleaning", and "visual programming".
- Comments:** A large text area for entering comments.
- Submit survey data:** A button at the bottom of the form.

Figure 1–27: The Survey application's input form.

- Clear the age input field to blanks and click **Submit survey data** without entering any other values in the input form. The default validation error messages appear next to the first three input fields (the controls that require input to be entered). JSF handles all the validation processing for you using the options you specified for each component's properties.

4. Enter an invalid value for the age input field (such as an alphabetic value or an integer outside the valid range of 1–120), and click **Submit survey data**. You will see the corresponding default error message (“conversion error” if you entered an alphabetic value or “not between specified values” if you entered an integer outside the valid range).
5. Enter valid values for all fields. To select a color, click the **Show Colors** icon () next to the control and select the color you want.
6. Click **Submit survey data**. This time, you see a response Web page like that shown in Figure 1–28. The favorite color output field is filled with the color you selected, since the same value is specified for both the color of the text in the field and the field’s background (the text itself is the internal value for the color).

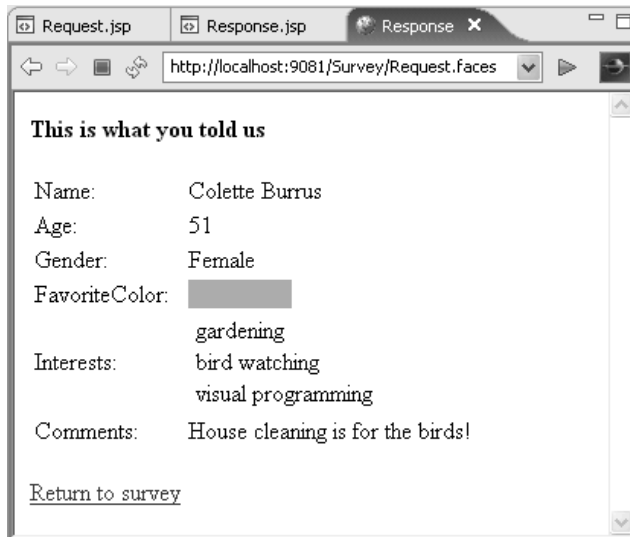


Figure 1–28: The response Web page with survey data filled in.

7. To return to the input form, click **Return to survey**. Notice that the input form’s fields are reset. That’s because your managed bean’s scope was set to “request”. If it had been a serializable bean with a scope of “session”, which is what real-world applications typically use, the data would have been retained for the user’s entire session.
8. When you’re finished testing, close the Web browser and Page Designer windows.

Importing the Solution File

If you ran into any problems following the instructions in this chapter, import the solution file in the `\solutions\chap01` subdirectory of the CD-ROM included with this book. Follow these instructions to import the solution file, and publish and run the Web application in the WebSphere Test Environment:

1. Select **File** → **Import...** in the Workbench menu.
2. In the Import Source window, click the plus sign next to J2EE, select **EAR file**, and click **Next**.
3. Click the **Browse...** button and browse to the `\solutions\chap01` folder on the CD-ROM.
4. Select **Chap01SurveyEAR.ear** and click **Open** so that the Import window looks like Figure 1–29. (The project name will be automatically filled in.)

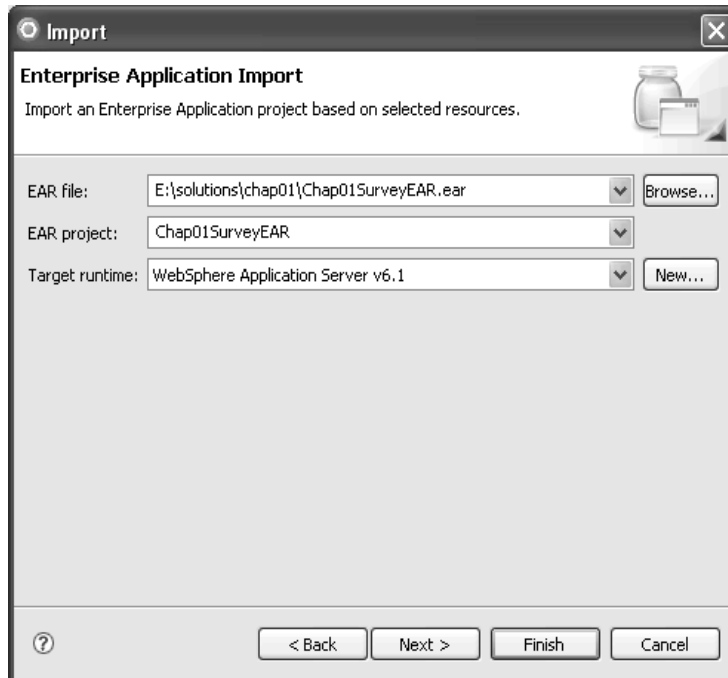


Figure 1–29: Importing the solution file for the Survey Web application.

5. Click **Finish**. If you're prompted to use newer Faces resources for the project, click **Yes**. If you're prompted to switch to the J2EE perspective, click **No**. (To avoid this prompt in future, check the **Remember my decision** check box.) Two new projects appear in the Project Explorer: Chap01Survey and Chap01SurveyEAR; they correspond to your own Survey and SurveyEAR projects.

Tip: If you see a warning message in the faces-config file that a concrete implementation is needed for a class, select **Project → Clean...** in the Workbench menu. Select **Clean projects selected below**, check the **Chap01Survey** check box, and click **OK**. This should rebuild the project and create the missing class file. If you see warning messages about type safety in the pagecode classes for the imported project, you can ignore those messages.

6. To deploy the imported project to your WebSphere Test Environment, click the **Servers** tab. Right-click **WebSphere Application Server v6.1**, and select **Add and Remove Projects...** from the pop-up menu.
7. Select **Chap01SurveyEAR** in the list of available projects, and click **Add** to move the project to the list of configured projects so that the window looks like Figure 1–30.

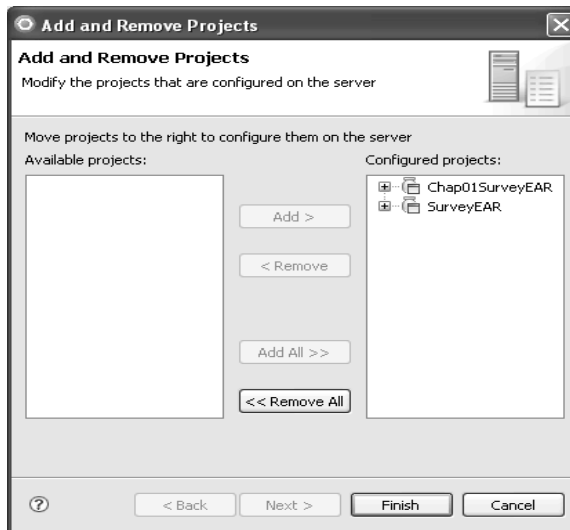


Figure 1–30: Adding the imported project to the server configuration.

8. Click **Finish**.

Tip: You can also use the Add and Remove Projects wizard to uninstall applications from the WebSphere Test Environment by removing the associated EAR file from the list of configured projects. If you ever want to delete solution files you've imported into your Workbench, make sure you remove the projects from your WebSphere Test Environment before deleting the projects from the Workbench.

9. To run the Web application, right-click **Request.jsp** in the WebContent folder of the Chap01Survey project and select **Run As → Run on Server** from the pop-up menu. When prompted to select a server to launch, make sure that **WebSphere Application Server v6.1** is selected and click **Finish**.

Tip: If you see an error message saying the server can't instantiate a class in the Chap01Survey project, remove the Chap01SurveyEAR project from the server, delete both the Chap01Survey and Chap01SurveyEAR projects from your Workbench, and repeat the steps to import the Chap01SurveyEAR project and add it to your server. Sometimes the project isn't completely built during import, so classes may be missing when you try to run the Web page.

Stopping the WebSphere Test Environment

Because the WebSphere Test Environment doesn't automatically stop when you exit Rational Application Developer, you can free up resources on your computer by stopping the server when you no longer need it for testing. Follow these steps:

1. Click the **Servers** tab.
2. Right-click **WebSphere Application Server v6.1** in the Servers view and select **Stop** from the pop-up menu.
3. If prompted with a message saying the server is not responding, click **OK** to terminate the server. The server's status changes from "Started" to "Stopped".

In Review

In this chapter, you learned how to use several of the basic Faces UI components to build a simple Web application—an input form that accepts user input, and a Web page to echo the data entered by the user. You learned how to bind Faces components to properties in a Faces managed bean, how to add validation to your Web pages, and how to define navigation rules telling JSF how to navigate from one page to another. You did all this using Rational Application Developer's Page Designer to visually build the Web application. In the next chapter, you'll learn how to create Web diagrams defining your Web application's flow, as well as how to use more complex Faces UI components.

Exercise

Several of the Faces UI components extend common superclasses, which means that, if you know how to use one component in that group, you can easily use any components that extend the same class. `RadioButtonGroup` and `ComboBox`, for example, both extend a super class that allows a single item to be selected from a list of choices—the only difference is in how the components are rendered on the Web page. Likewise, `CheckBoxGroup` and `List Box – Multiple Select` both extend a superclass that allows multiple items to be selected from a list of choices. Change your Survey Web application to use the `ComboBox` component instead of `RadioButtonGroup` for the gender and to use the `List Box – Multiple Select` component instead of `CheckBoxGroup` for the interests.

A solution file for the exercise, called `Chap01ExerciseEAR.ear`, is in the `\solutions\chap01` subdirectory on the CD-ROM included with this book. After importing this EAR file, don't forget to add the EAR project to your WebSphere Test Environment.

References

IBM Rational Application Developer trial
<http://www.ibm.com/developerworks/downloads/r/rad>