CHAPTER 41

JSF AND VISUAL WEB DEVELOPMENT

Objectives

- To explain what JSF is (§41.1).
- To create a JSF page using NetBeans (§41.2).
- To use JSF UI components (e.g., Static Text, Text Field, Button, Drop Down List, List Box, Radio Button Group, Check Box Group, Text Area, Table) (§41.3).
- To use JSF containers Grid Panel, Group Panel, and Layout Panel to group components (§41.4).
- To bind data with JSF UI components (§41.5).
- To maintain persistency using session tracking (§41.6).
- To validate input using Message components (§41.7).
- To improve efficiency using virtual forms (§41.8).

41.1 Introduction

servlets
JSP

The use of servlets, introduced in Chapter 39, is the foundation of the Java Web technology. It is a primitive way to write server-side applications. JSP provides a scripting capability and allows you to embed Java code in HTML. It is easier to develop Web programs using JSP than servlets. However, JSP has some problems. First, it can be very messy, because it mixes Java code with HTML. Second, using JSP to develop user interface is tedious. JavaServer Faces (JSF) comes to rescue. JSF supports visual Web development. You can create a Web user interface using a tool without writing any code. JSF completely separates Web UI from Java code, so the application developed using JSF is easy to debug and maintain.

JSF

NetBeans Visual Web Tool

A couple of tools support JSF. This chapter demonstrates visual JSF using NetBeans 6.7. Visual JSF is a plug-in for NetBeans 6.7. To install it, choose **Tools**, **Pluggins**, and select **Visual JSF**.

41.2 Getting Started with Visual JSF

A simple example will illustrate the basics of developing visual JSF projects using NetBeans. The example is a server-side application that displays the date and time on the server, as shown in Figure 41.1.

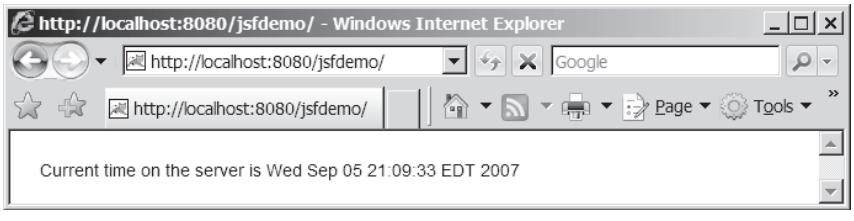


FIGURE 41.1 The application displays the date and time on the server.

Here are the steps to create the application.

create a project

1. Choose **File** › **New Project** to display the New Project dialog box. In this box, choose *Web* in the Categories pane and *Web Application* in the Projects pane. Click *Next* to display the New Web Application dialog box.

2. In the New Web Application dialog box, enter and select the following fields, as shown in Figure 41.2(a):

```
Project Name: jsfdemo
Project Location: c:\book
Check Set as Main Project
```

Click *Next* to display the dialog box for choosing servers and settings. Select the following fields as shown in Figure 41.2(b). (Note: Some examples do not work in V3 without modifications of project files due to bugs in the new version. So, this chapter uses V2.)

```
Server: GlassFish V2
Java EE Version: Java EE 5
```

Click *Next* to display the dialog box for choosing frameworks, as shown in Figure 41.3. Check *Visual Web JavaServer Faces*. Click *Finish* to create the project, as shown in Figure 41.4.

FIGURE 41.2 The New Web Application dialog box enables you to create a new Web project.

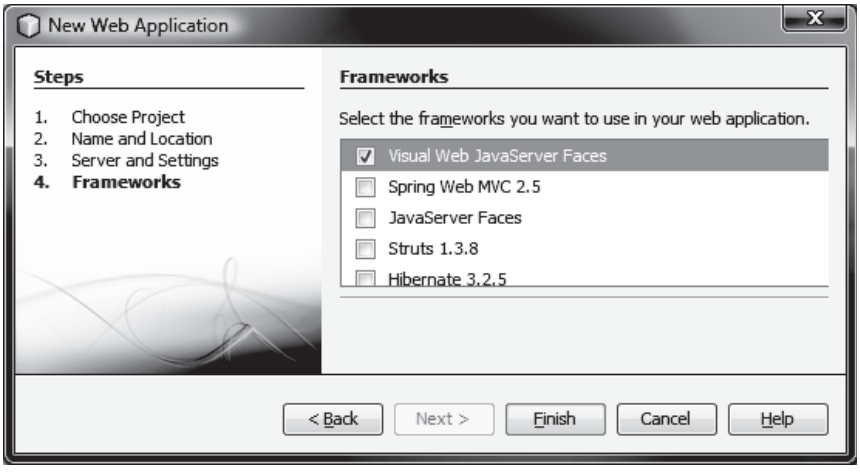


FIGURE 41.3 Checking Visual Web JavaServer Faces creates a visual Web project.

A visual Web project is created with an empty page named Page1.jsp, as shown in Figure 41.4. You see `Page1` with subtabs in the content pane. The **Design** tab enables you to design the UI visually. The **JSP** tab enables you to view and modify the JSP codes. The **Java** tab enables you to view and modify the Java code. These three tabs are synchronized. Whenever you make a change in the Design, the corresponding code in the JSP and Java will change and vice versa.

Design tab
JSP tab
Java tab

The Palette window contains the UI components that you can drag and drop to the Design pane.

palette window

Tip

There are several windows for a visual Web project. If a window (e.g., the Palette window) is not displayed, choose an appropriate menu from the Windows menu to display it.

Now you are ready to create a page for displaying the current date and time on the server. Here are the steps:

1. Rename Page1.jsp to CurrentTime.jsp. To do so, right-click the context menu on Page1.jsp in the Project pane and choose **Refactor** › **Rename** to display the Rename dialog box. Enter `CurrentTime` as the new name and click *Refactor* to finish renaming.

renaming files

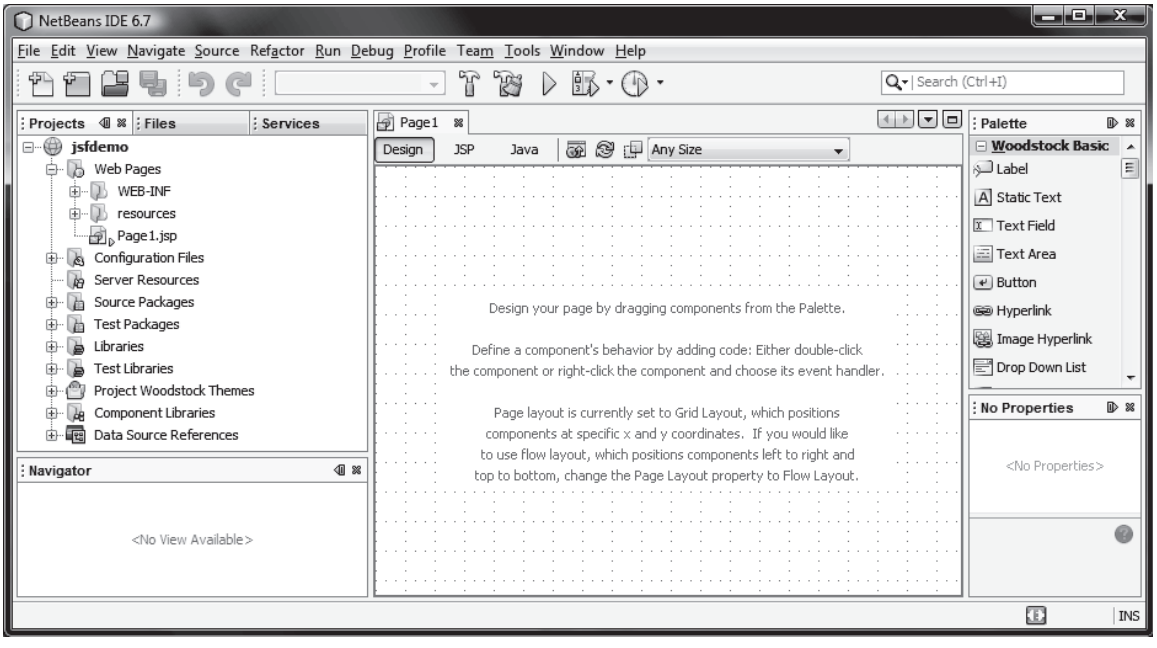


FIGURE 41.4 An empty page is created in a new Visual Web project.

Static Text

2. Click to select the *Static Text* icon from the Palette pane and drop it to the Design pane, as shown in Figure 41.5. In the id field in the Properties pane for the static text, enter `stCurrentTime` to change the id. Whenever you click a component in the Design pane, its corresponding properties are displayed in the Properties pane. You can view and modify the component's properties in the Properties pane.

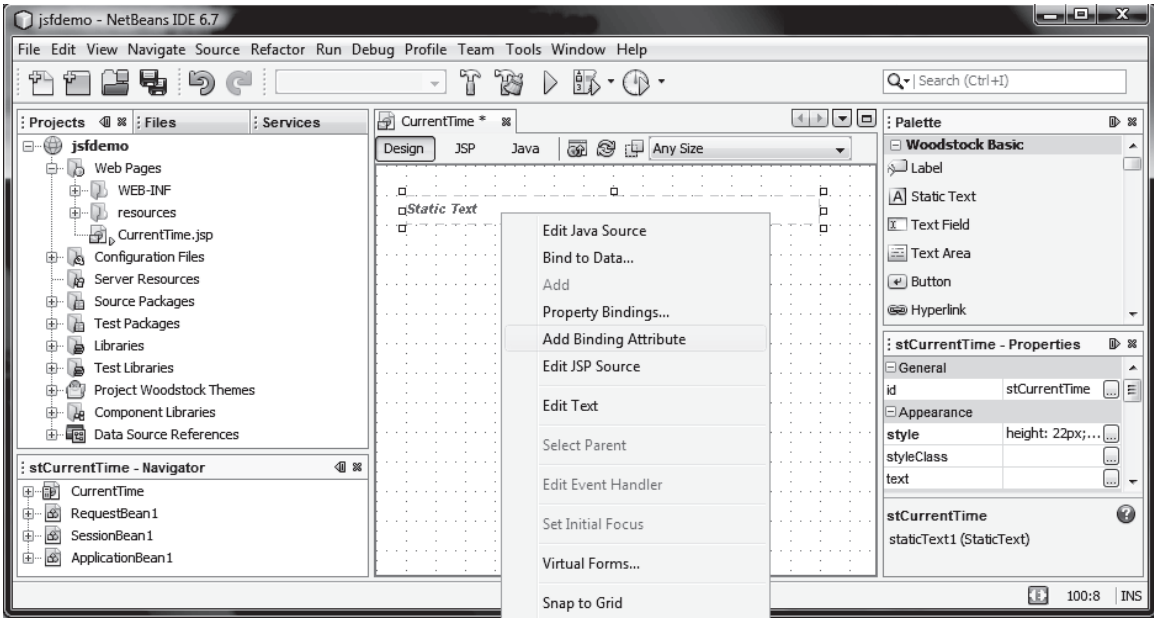


FIGURE 41.5 A static text is placed in the Design pane.

3. Right-click on `stCurrentTime` in the Design pane to display a content menu, as shown in Figure 41.5. Choose *Add Binding Attributes*. This step automatically generates code that enables you to access `stCurrentTime` in the Java source code.

add binding attributes

4. Click the Java tab in the content pane and enter

```
stCurrentTime.setText("Current time on the server is "
  + new java.util.Date());
```

in the `prerender` method, as shown in Figure 41.6.

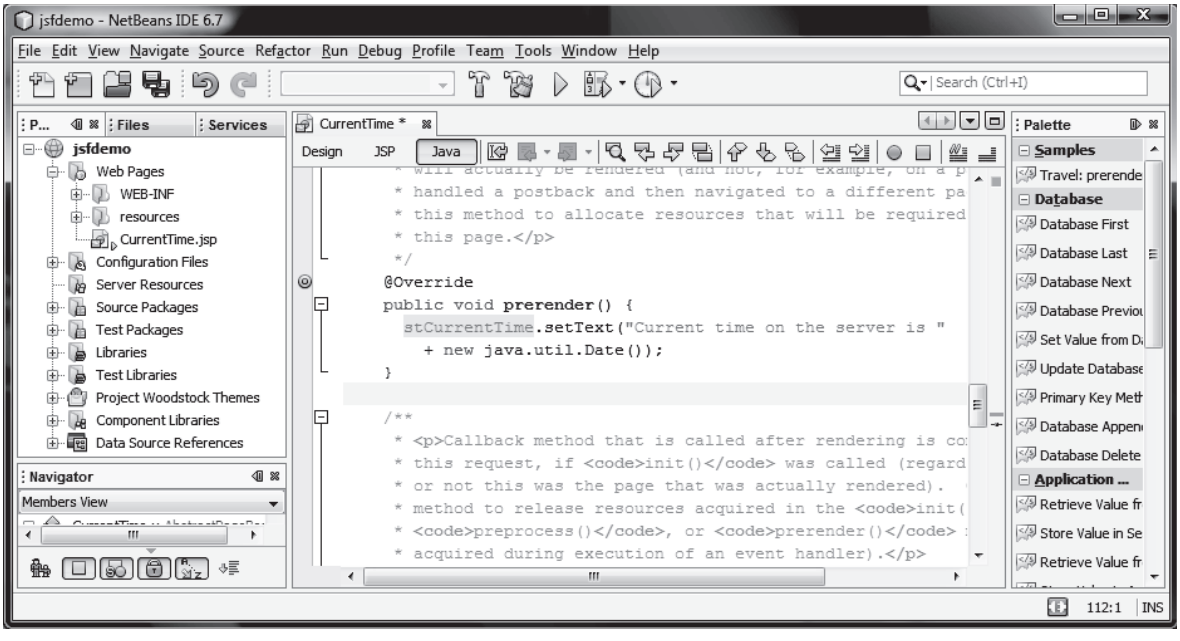


FIGURE 41.6 You can modify the code in the Java tab.

5. Right-click on the `jsfdemo` project to display a context menu and choose **Run** to run the JSF. This starts a Web browser to display the current date and time, as shown in Figure 41.1.

Note

By default, the first page in the project is set as the main page. You can access it using the URL http://localhost:8080/jsfdemo. The explicit URL for this page is http://localhost:8080/jsfdemo/faces/CurrentTime.jsp.

explicit URL

Note

If you choose the JSP tab, you will see the corresponding JSP file, as shown in Figure 41.7. The contents of this file in XML mirrors the UI components in the Design pane. Whenever you add, remove, or change the UI components in the Design pane, the contents in the JSP are also updated. It is possible to modify the JSP file directly, but this is not recommended for new users. Modifying the JSP file mistakenly could corrupt the entire project. You can completely ignore the JSP file when using this tool.

JSP file

Note

Clicking the Java tab in the content pane, you will see the Java source file, known as the *page bean file*. This file contains several methods. Among them are the four JSF life-cycle methods `init`, `preprocess`, `prerender`, and `destroy`.

The `init` method is called whenever the page is navigated to, either by directly via a URL, or indirectly via page navigation. The `preprocess` method is called after the component tree has been restored, but before any event processing takes place. The `prerender` method is called

page bean file
`init`
`preprocess`
`prerender`

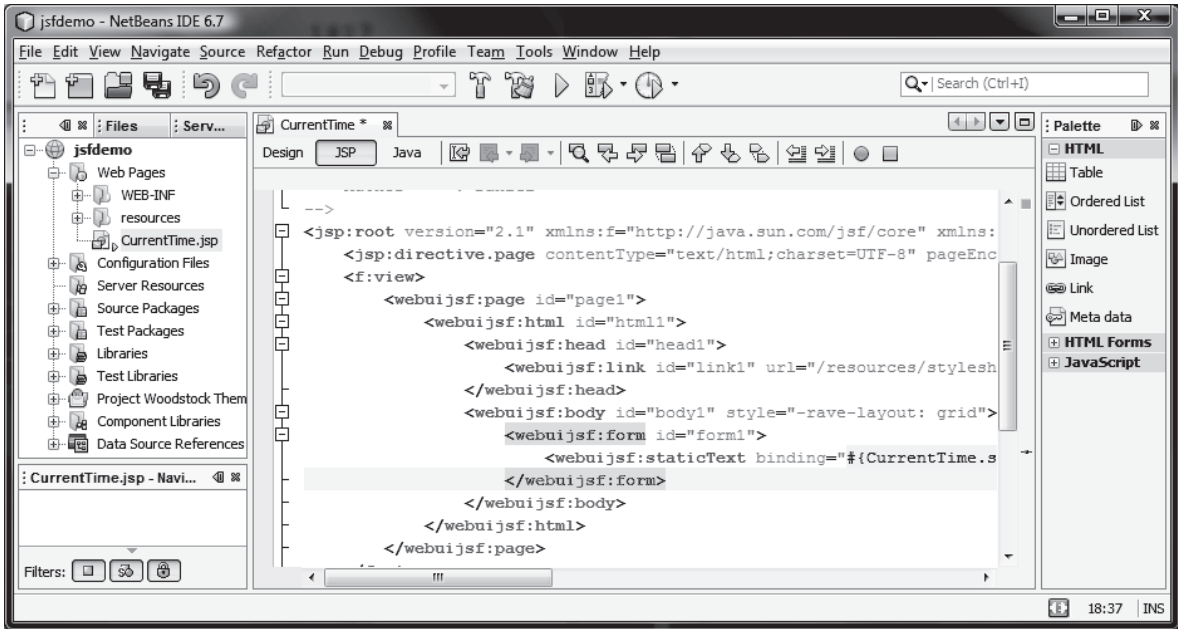


FIGURE 41.7 You can see the code in the JSP tab.

destroy

just before rendering takes place. This method will be called only for the page that will actually be rendered. The `destroy` method is called after rendering is completed for this request.

All these methods are automatically implemented. You can add the code to customize their behaviors. Normally you should customize the `prerender` method to set appropriate properties for the components for rendering.

41.3 JSF UI Components

JSF provides many user interface components that you can use in visual design. These components are comparable to Swing components. They are JavaBeans components with similar properties and events as their counterparts in Swing. The frequently used components are grouped into the *Basic* and *Layout* nodes in the Palette window. This section introduces some standard components such as text fields, drop down lists, list boxes, radio buttons, check boxes, text areas, and buttons, through an example. The example is to design a form, as shown in Figure 41.8. Clicking the *Register* button displays the input in a separate form, as shown in Figure 41.9.

The example can be completed in three phases:

1. Create the initial input form.
2. Create the result form.
3. Write the code for handling the event.

Phase 1: Creating the input form

create a new page

1. Right-click the `jsfdemo` node in the Project window to display a context menu, and choose **New** › **Visual Web JSF Page** to display the New Visual Web JSF Page dialog box. Enter `Registrations` in the File Name field. Click *Finish* to create Registrations.jsp, as shown in Figure 41.10. Right-click Registration.jsp in the Projects pane to display a context menu and choose *Set As Start Page* so this page will start when you run the `jsfdemo` project.

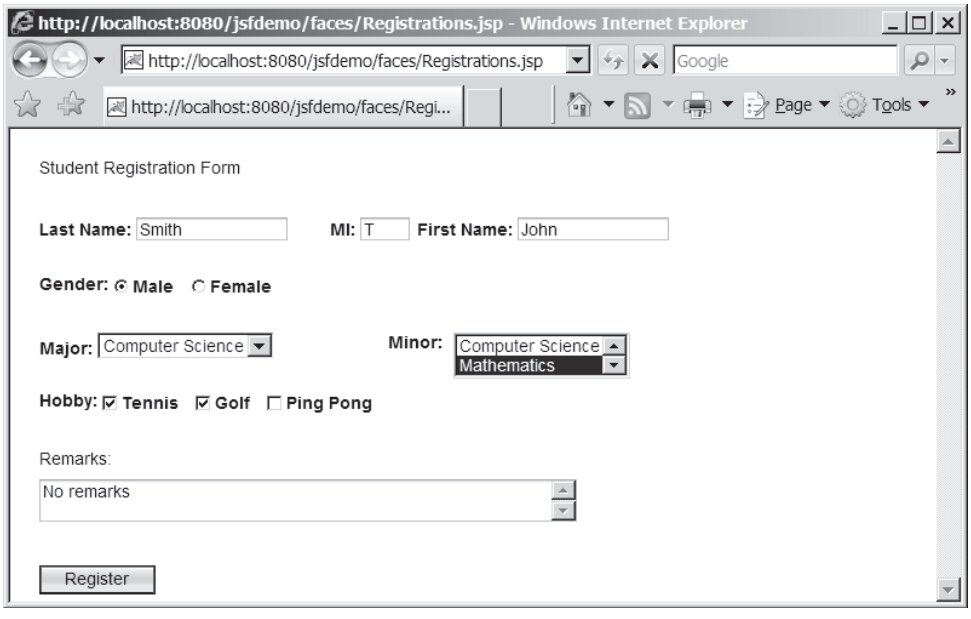


FIGURE 41.8 The form consists of various UI components.

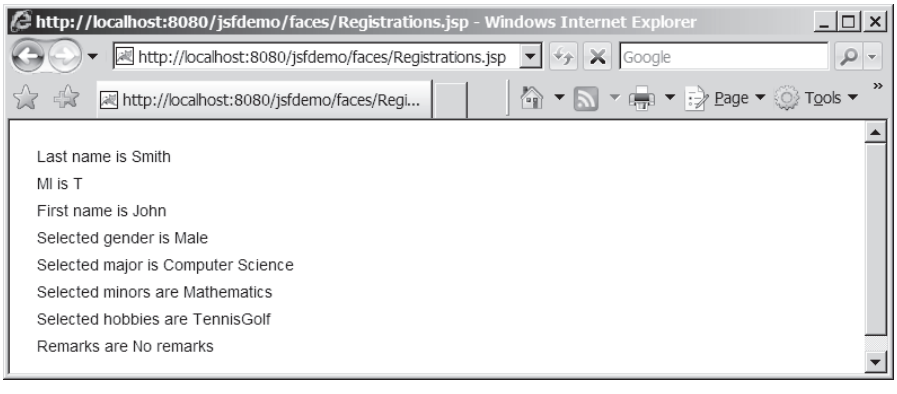


FIGURE 41.9 The result from the input form is displayed in this form.

2. Drop a Static Text in the Design pane with the text `Student Registration Form`. You can set a new font and color in the `style` property in the Properties window, if you wish. — set font and color

3. Drop a text field and set its `id` to `tfLastName`, `column` to `15`, and `label` to `Last Name:`, as shown in Figure 41.11. — create a text field

4. Drop a Text Field and change its `id` to `tfMi`, `column` to `1`, and `label` to `MI:`. — create a text field

5. Drop a Text Field and change its `id` to `tfFirstName`, `column` to `10`, and `label` to `First Name:`. — create a text field

6. Drop a Radio Button Group. Change its `id` property to `rbgGender`, `columns` property to `2`, and `label` to `Gender:`. — radio button group

7. Right-click the radio button group object in the Design to display its context menu, choose *Configure Default Options* to display the Options Customizer dialog box. Set the appropriate display and value, as shown in Figure 41.12(a). — radio button values

8. Drop a Drop Down List and change its `id` to `ddMajor` and `label` to `Major:`. — drop down list

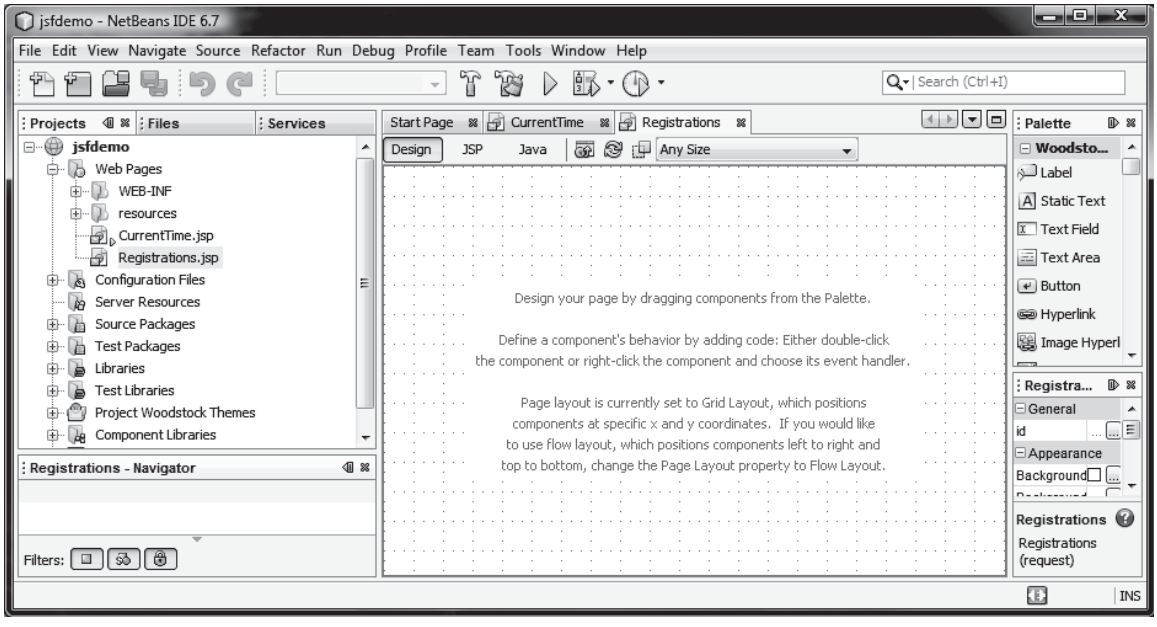


FIGURE 41.10 A new JSF page is created.

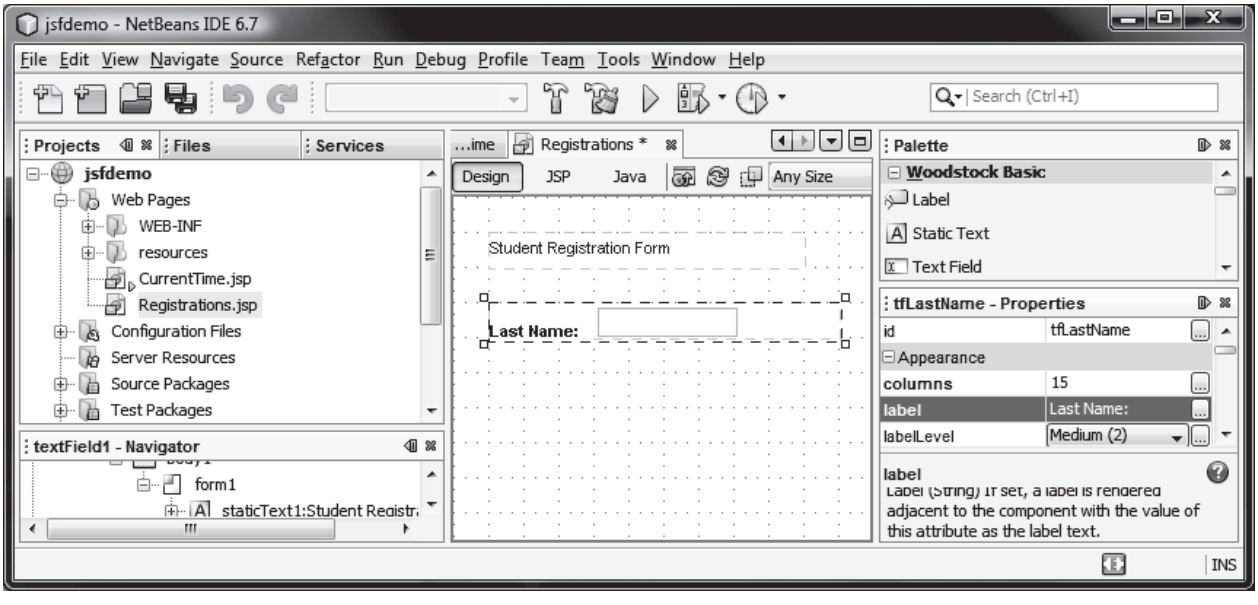
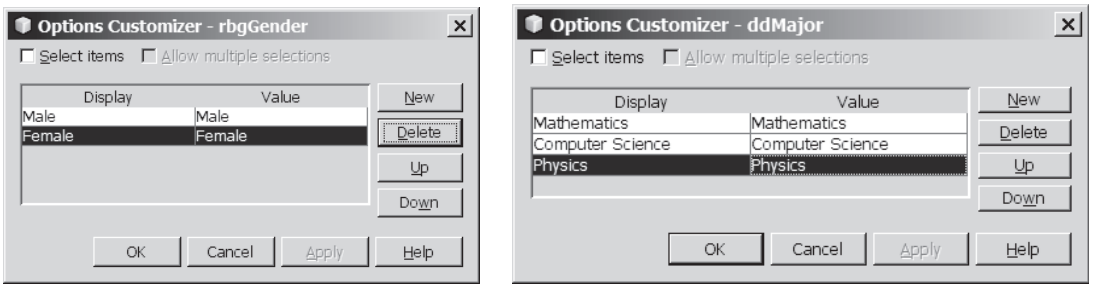


FIGURE 41.11 `tfLastName` has been moved to desired location.



(a) For radio buttons (b) For list box

FIGURE 41.12 You can customize the display name and value for radio buttons and list boxes.

9. Right-click the list in the Design to display its context menu, and choose *Configure Default Options* to display the Options Customizer dialog box. Set the appropriate display and value, as shown in Figure 41.12(b). — list values
10. Drop a List Box and change its `id` to `lbxMinor`, `label` to `Minor:`, `rows` to `2`, and `multiple` to `true`. — list box
11. Right-click the box in the Design to display its context menu, and choose *Configure Default Options* to display the Options Customizer dialog box. Set display and value the same as shown in Figure 41.12(b). — list values
12. Drop a Check Box Group. Change its `id` to `chkgHobby`, `label` to `Hobby:`, and `columns` to `3`. Right-click the box in the Design to display its context menu, choose *Configure Default Options* to display the Options Customizer dialog box. Set the display name and values for Tennis, Golf, and Ping Pong. — check box group
13. Drop a Static Text with the text `Remarks:`.
14. Drop a Text Area with `id` and `columns` set to `taRemarks` and `70`. — create a text area
15. Drop a Button and set its `id` to `btRegister` and its `text` to `Register`. — create a button
16. Add binding attribute for `tfLastName`, `tfMi`, `tfFirstName`, `rbgGender`, `ddMajor`, `lbxMinor`, `chkgHobby`, and `taRemarks`. To add binding attribute for a component, right-click the component in the Design pane to display the context menu and choose *Add Binding Attribute.* — add binding attribute

The user interface for the input is now created as shown in Figure 41.13. All these components are contained in `form1`. Now create `form2` for displaying the data collected from the first form.

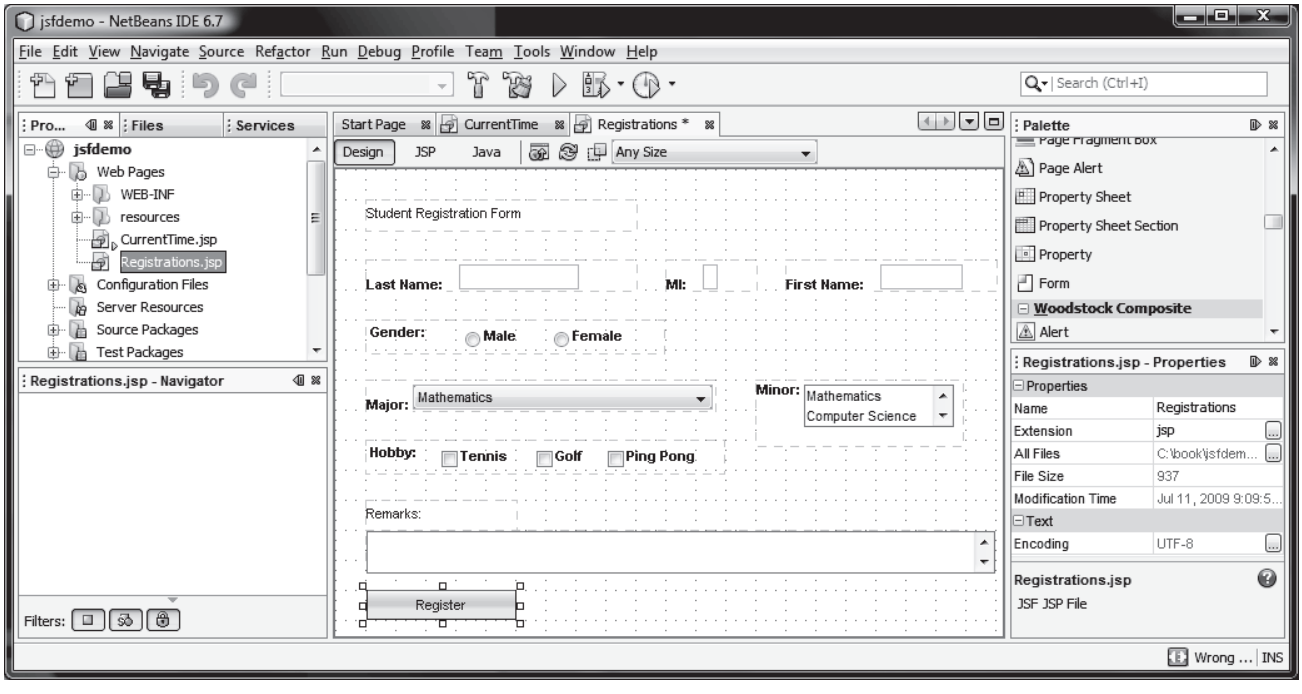


FIGURE 41.13 `form1` is created to contain input components.

Phase II: Creating the result form

1. Uncheck the `visible` property for `form1` to set it false so `form1` is not displayed in the Design pane.
2. Drop a Form from the *Layout* tab in the Palette pane to create a form named `form2`, as shown in Figure 41.14. — create another form

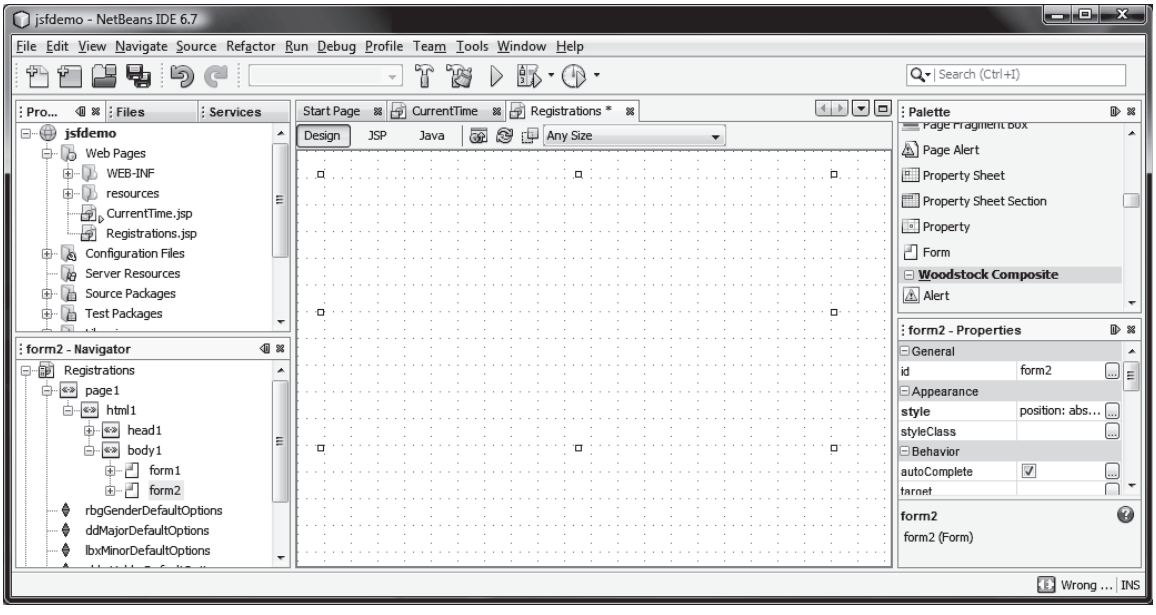


FIGURE 41.14 `form1` is invisible. `form2` is created.

3. Drop eight Static Texts into `form2` and set their `id` to `stLastName`, `stMi`, `stFirstName`, `stGender`, `stMajor`, `stMinor`, `stHobby`, and `stRemark`, respectively, as shown in Figure 41.15. Add binding attributes for all these static texts.

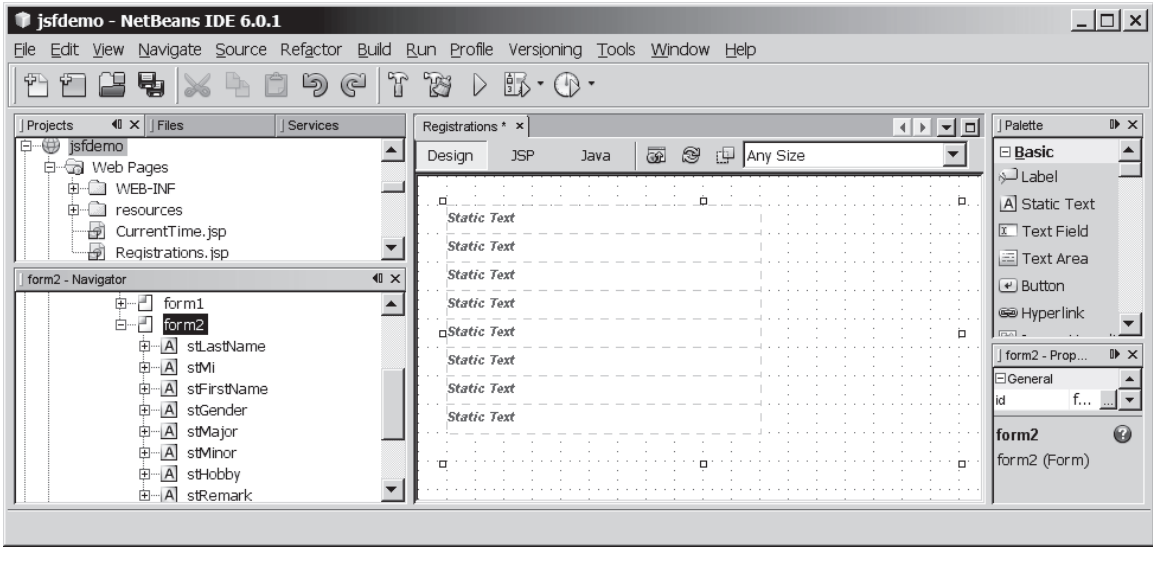


FIGURE 41.15 Static texts are added to `form2`.

4. Set the `rendered` property in `form2` to `false`. `form2` will not be displayed when the page is started.
5. Now check the `visible` property of `form1` to `true` and the `visible` property of `form2` `false`.
6. Add binding attributes for `form1` and `form2`. In the Navigator pane, right-click `form1` to display the context menu and choose *Add Binding Attribute*. Similarly you can add binding attribute for `form2`.

Phase III: Handling the event

We are ready to implement the code to handle the event for the *Register* button. Double-click the *Register* button to generate the handler method `btRegister_action` in the page bean file. Implement the methods as follows:

event handler

```
1 public String btRegister_action() {
2   // TODO: Replace with your code
3   form1.setRendered(false);                                    form1 not rendered
4   form2.setRendered(true);                                     form2 rendered
5   form2.setVisible(true);
6   stLastName.setText("Last name is " + tfLastName.getText());  get text
7   stMi.setText("MI is " + tfMi.getText());
8   stFirstName.setText("First name is " + tfFirstName.getText());
9
10  stGender.setText("Selected gender is " +
11    rbgGender.getSelected().toString());                       selected radio button
12  stMajor.setText("Selected major is " +
13    ddMajor.getSelected().toString());                         selected item
14
15  String[] selectedMinors = (String[])(lbxMinor.getSelected()); selected items
16  String minors = "";
17  for (int i = 0; i < selectedMinors.length; i++)
18    minors += selectedMinors[i];
19  stMinor.setText("Selected minors are " + minors);
20
21  String[] selectedHobbies = (String[])(chkgHobby.getSelected());
22  String hobbies = "";
23  for (int i = 0; i < selectedHobbies.length; i++)
24    hobbies += selectedHobbies[i];
25  stHobby.setText("Selected hobbies are " + hobbies);
26
27  stRemark.setText("Remarks are " + taRemarks.getText());
28  return null;
29 }
```

Clicking the *Register* button displays `form2`. `form1`'s `rendered` property is set to `false` (line 3) in order to hide it. `form2`'s `rendered` property is set `true` to make it visible (line 4). `form2`'s `visible` property is set `true` to ensure that the form itself is visible (line 5). For a component to be displayed, both `visible` and `rendered` properties must be `true`.

All the JSF UI components are JavaBeans components. Their properties have the associated get and set methods. You can obtain the text in a text field and text area using the `getText()` method (lines 6–8). You can obtain the selected items from the a radio button group, check box group, drop down list, or list box using the `getSelected()` method (lines 11, 13, 15, 21).

`getText()`

`getSelected()`

Note

The second form is displayed when the *Register* button is clicked. When `form2` is displayed, `form1` will not be shown. This is achieved by setting `form1`'s `rendered` property to `false`. Note in the Design pane, you are free to move `form2` to a desired location even if it overlaps `form1`.

41.4 JSF UI Containers

Like Swing, JSF provides containers that can be used to group components to achieve a desired layout. Three containers are available under the Layout node in the Palette: *`GridPanel`*, *`GroupPanel`*, and *`LayoutPanel`*. `GridPanel` is like the Swing `GridLayout`. `GroupPanel` is similar to the Swing `FlowLayout`. `LayoutPanel` organizes child components using flow or absolute positioning.

`GridPanel`

`GroupPanel`

`LayoutPanel`

Consider the following example that computes loan payments. The example lets the user enter a loan amount, number of years, and annual interest rate, as shown in Figure 41.16.

Clicking the *Compute Loan* button displays the monthly payment and total payment, as shown in Figure 41.17.

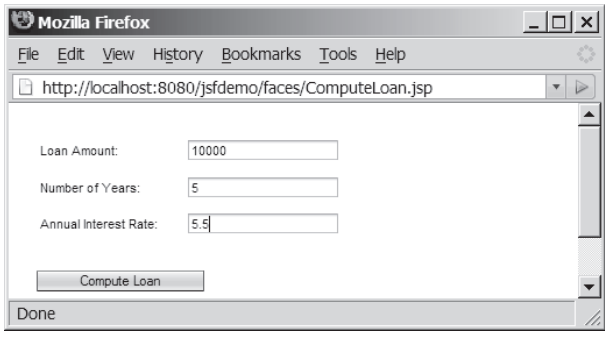


FIGURE 41.16 The form lets the user enter loan information.

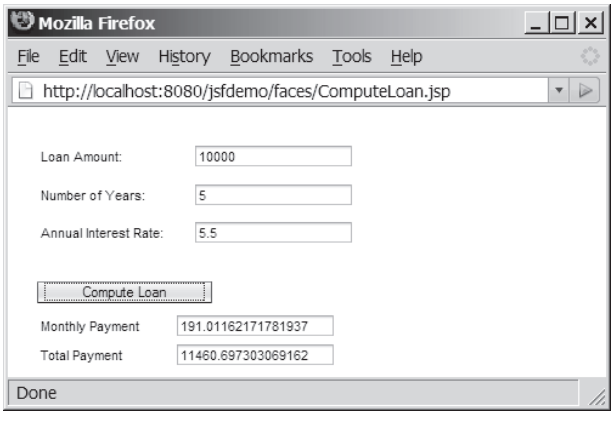


FIGURE 41.17 The loan payment is computed and displayed.

Here are the steps to create the UI:

create a new page

1. Right-click the `jsfdemo` node in the Project window to display a context menu, choose **New** › **Visual Web JSF Page** to display the New Page dialog box. Enter `ComputeLoan` in the File Name field. Click *Finish* to create ComputeLoan.jsp. Set ComputeLoan.jsp as the start page for the project.

create a grid panel

2. Drop a `GridPanel` to the Design pane and set its `id` to `gridPanel1` and `columns` to `2`.
3. Drop a Static Text, a Text Field, a Static Text, a Text Field, a Static Text, and a Text Field to the grid panel in this order. Set the static text to `Loan Amount`, `Number of Years`, and `Annual Interest Rate`. Change the text fields id to `tfLoanAmount`, `tfNumberOfYears`, and `tfAnnualInterestRate`. Add binding attributes for the text fields.
4. Drop a Button to the grid panel and set its `id` to `btComputeLoan` and text to *Compute Loan*.

create a grid panel

5. Drop another `GridPanel` to the Design pane below the preceding grid panel and set its `id` to `gridPanel2` and `columns` to `2`.
6. Drop a Static Text, a Text Field, a Static Text, and a Text Field to `gridPanel2` in this order. Set the static text to `Monthly Payment` and `Total Payment`. Change the `id` of the text fields to `tfMonthlyPayment` and `tfTotalPayment`. Add binding attributes for `tfMonthlyPayment`, `tfTotalPayment`, and `gridPanel2`.

The user interface is created as shown in Figure 41.18.

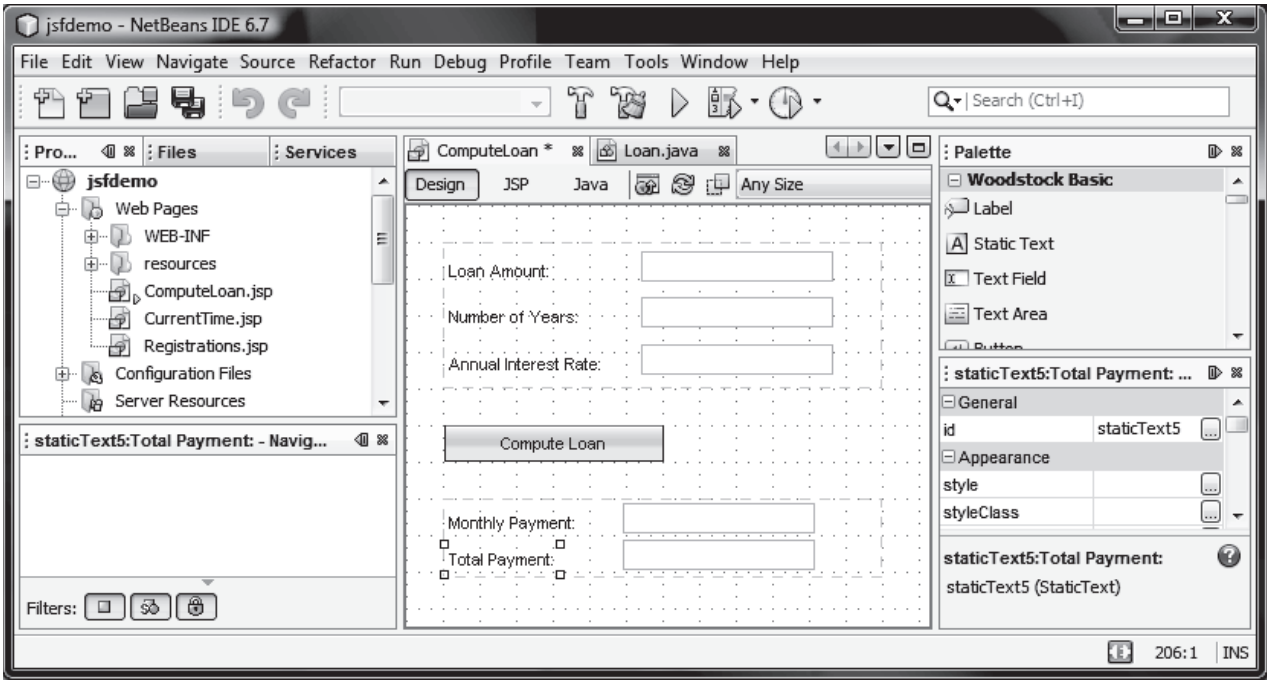


FIGURE 41.18 The panels are created to group UI components.

Tip

If the order of the components in a panel is not correct, you can reorder them in the Navigator pane using the mouse.

reorder components

Double-click the *Compute Loan* button in the Design pane to generate the method `btComputeLoan_action()`. Implement it as follows:

```
1  public String btComputeLoan_action() {
2    double loanAmount =                                          get loanAmount
3      Double.parseDouble(tfLoanAmount.getText().toString().trim())
4    int numberOfYears =                                          get numberOfYears
5      Integer.parseInt(tfNumberOfYears.getText().toString().trim());
6    double annualInterestRate = Double.parseDouble(              get annualInterestRate
7      tfAnnualInterestRate.getText().toString().trim());
8
9    chapter41.Loan loan = new chapter41.Loan(                    create a loan
10     annualInterestRate, numberOfYears, loanAmount);
11
12   gridPanel2.setRendered(true);                                gridPanel2 rendered true
13   tfMonthlyPayment.setText(loan.getMonthlyPayment() + "");     display monthly payment
14   tfTotalPayment.setText(loan.getTotalPayment() + "");         display total payment
15
16   return null;
17 }
```

The handler obtains `loanAmount`, `numberOfYears`, and `annualInterestRate` from the input text fields (lines 2–7). The program creates a `Loan` object (lines 9–10). You need to create a `Loan` class for this method to work. To create the `Loan` class, choose **File** › **New** › **Java Class** to display the New Java Class dialog box. Enter `Loan` in the Class Name field and `chapter41` in the package field. The `Loan` class was given in Listing 10.2.

create **Loan** class

Set `gridPanel2`'s `rendered` property to `false`. Run the program using the URL http://localhost:8080/jsfdemo/faces/ComputeLoan.jsp. You will see the user interface, as shown in

run ComputeLoan.jsp

Figure 41.16. When you click the *Compute Loan* button, `gridPanel2`'s `rendered` property to `true` (line 12). You will see the monthly payment and total payment displayed in Figure 41.17.

41.5 Binding Data with UI Components

data-aware components

You can bind data from a SQL query to a JSF UI component. This is a powerful feature and makes database programming easy. These JSF UI components are known as *data-aware components*.

Consider the following example that lets the user choose a course, as shown in Figure 41.19. After a course is selected in the drop down list, the students enrolled in the course are displayed in the table, as shown in Figure 41.20. In this example, all the course titles in the `Course` table are bound to the drop down list and the query result for the students enrolled in the course is bound to the table.

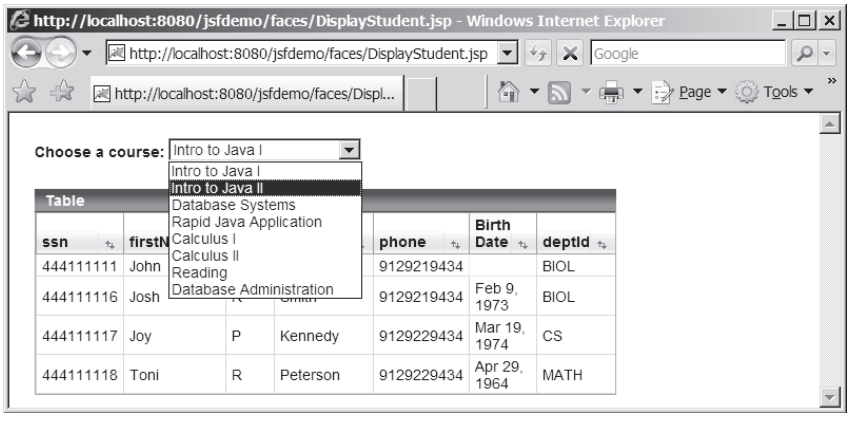


FIGURE 41.19 You need to choose a course and display the students enrolled in the course.

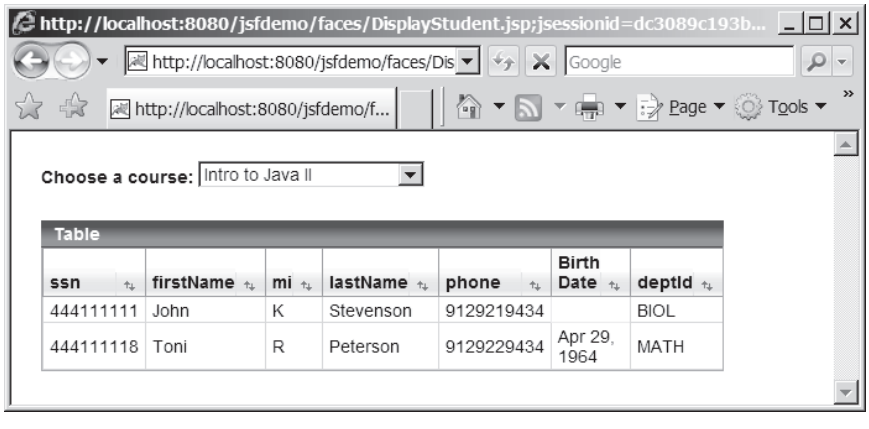
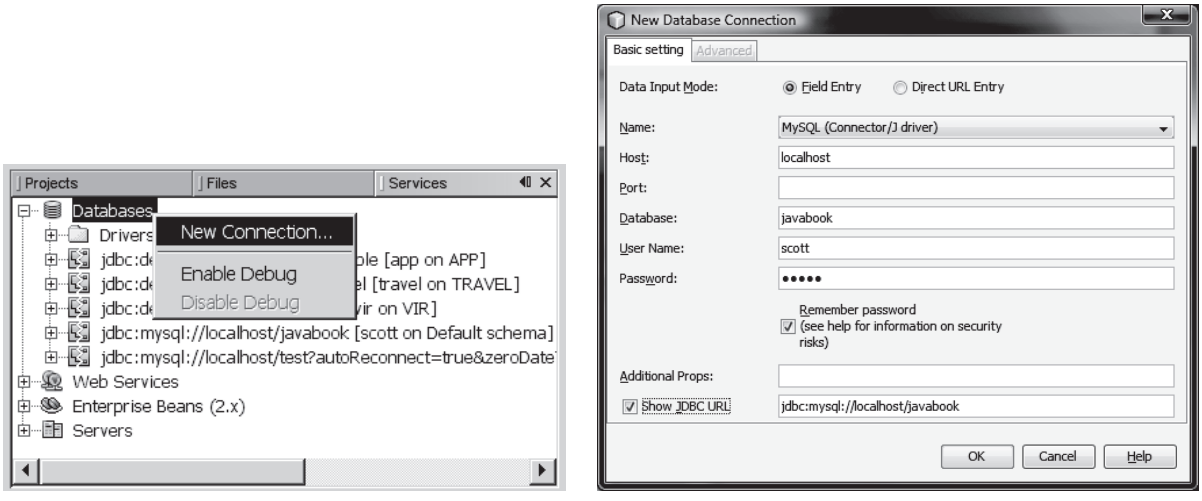


FIGURE 41.20 The table displays the students enrolled in the course.

create database connection

To develop this example, first you have to create a new database connection. In the Services pane, right-click the Database node and choose *New Connection* in the context menu (Figure 41.21(a) to display the New Database Connection dialog box, as shown in Figure 41.21(b)).

Use the same MySQL database we have been using since Chapter 37. Enter the database information, as shown in Figure 41.21(b). Click *OK* to create the database connection. You will see the database under the Databases node in the Services tab, as shown in Figure 41.22.



(a) (b)

FIGURE 41.21 You can create a database connection from the Services pane.

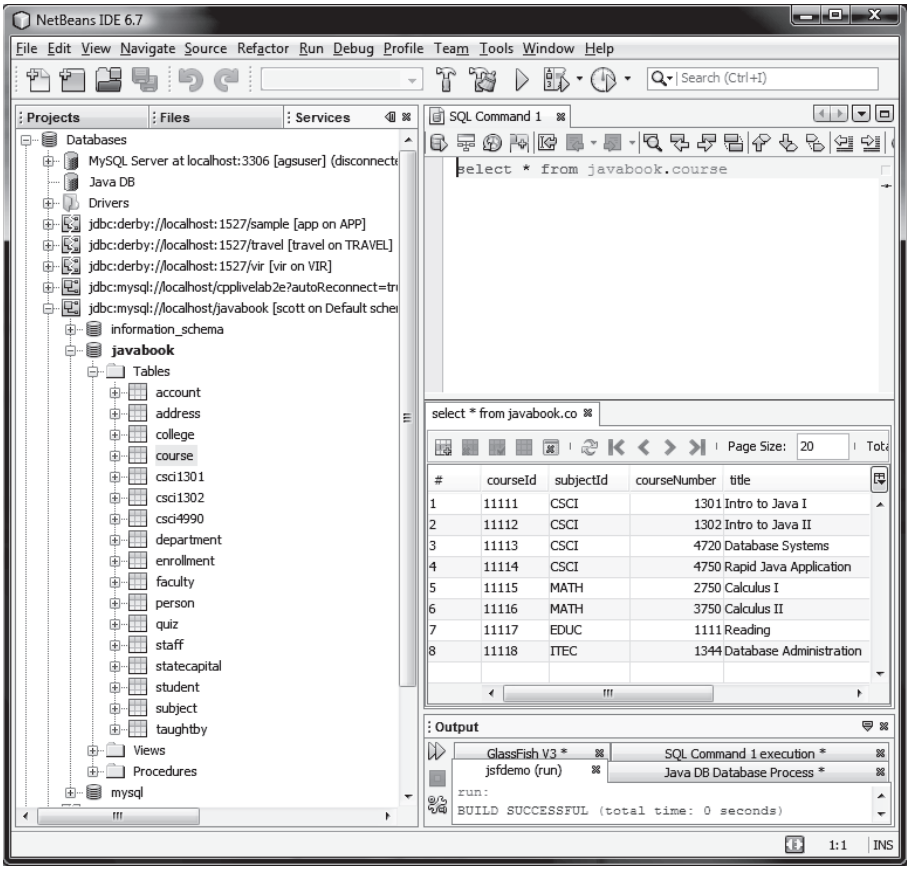


FIGURE 41.22 You can access the database from NetBeans.

Create a new Visual Web JSF page named `DisplayStudent` and create the user interface by dropping a Drop Down List, and a Table to the Design pane, as shown in Figure 41.23. Name the drop down list and table as `ddCourse` and `tbStudent`. Set the `label` property for `ddCourse` to `Choose a course:`.

create new page

create UI

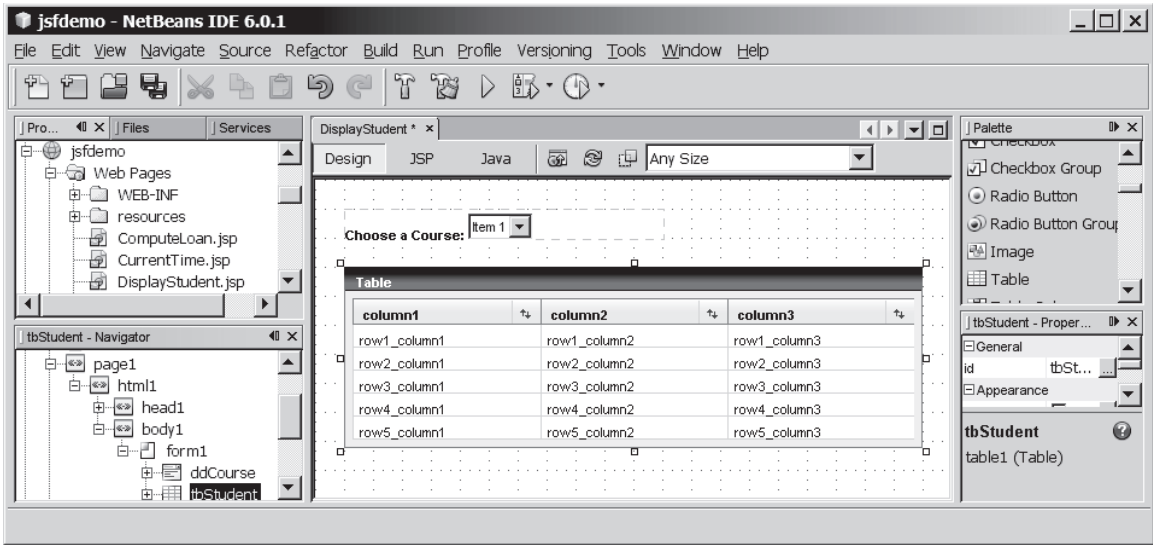


FIGURE 41.23 The user interface is created in the Design pane.

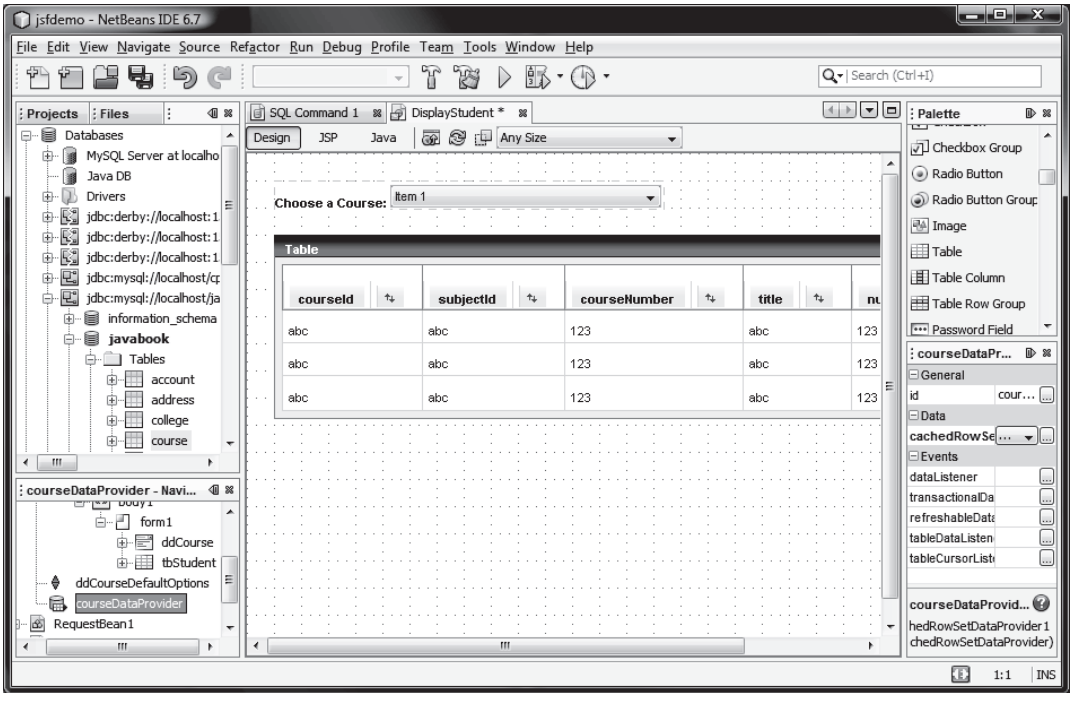


FIGURE 41.24 A data provider is created.

create data provider

bind data

The drop down list should be bound to the course titles from the `Course` table. Click the `Course` table in the database connection under the Database node in the Services pane, as shown in Figure 41.22. Drop the `Course` table to the table component in the Design pane. You will see `courseDataProvider` created in the Navigator pane, as shown in Figure 41.24.

To bind the data for `ddCourse`, right-click on `ddCourse` to display a context menu in the Design and choose *Bind to Data* to display the Bind to Data dialog box, as shown in Figure 41.25. Select `course.courseId` in the Value field and `course.title` in the Display field. Click *OK* to close the dialog box. Add binding attribute for `ddCourse`.

The table should be bound to a query result for the students enrolled in a selected course. Click the `Student` table in the database connection under the Database node in the Services pane. Drop the table to the table component `tbStudent` in the Design pane. You will see `studentDataProvider` created in the Navigator pane, as shown in Figure 41.26. create data provider

Each data provider has a `cachedRowSet` property that specifies a `RowSet`. `studentDataProvider` uses `studentRowSet`, as shown in Figure 41.27. Double-click `studentRowSet` under the `SessionBean1` node in the Navigator pane to display the row set in the content pane, as shown in Figure 41.28. display query visually

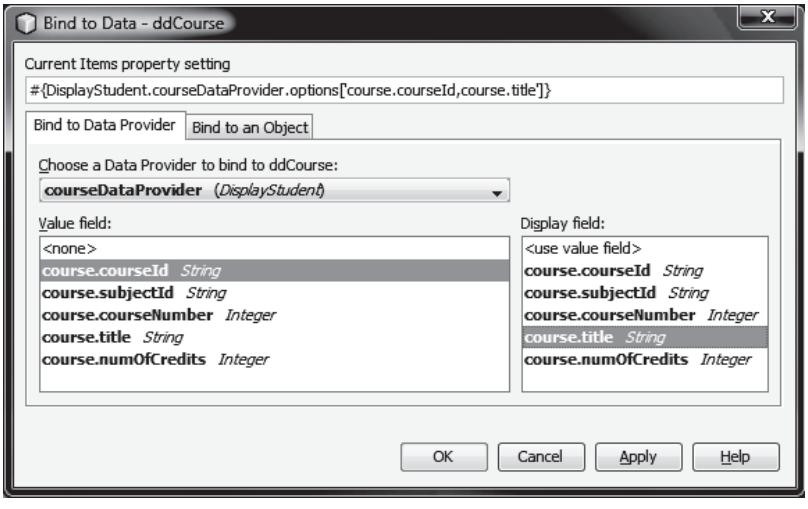


FIGURE 41.25 You can set value and display name in a drop down box.

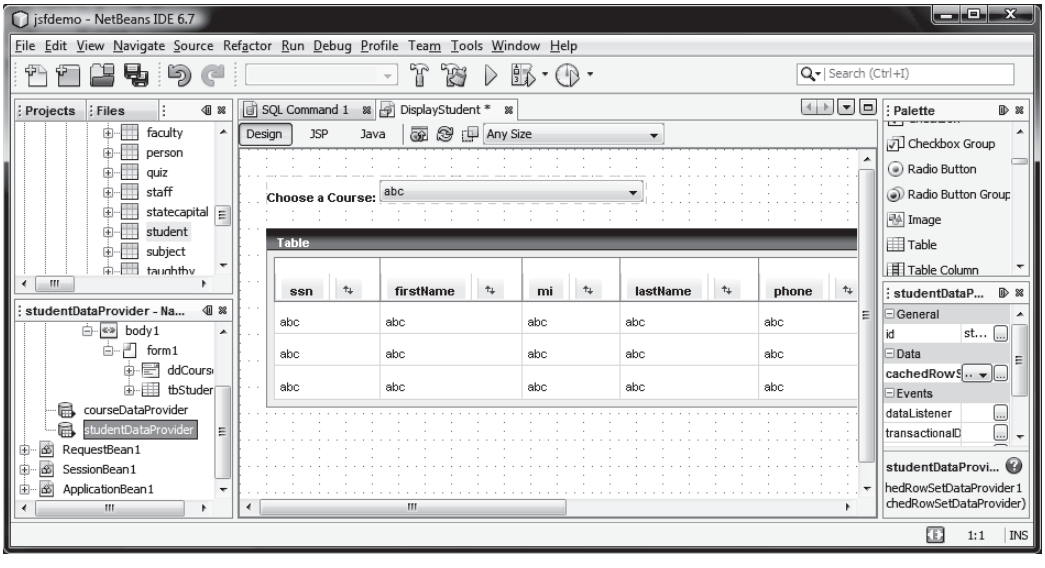


FIGURE 41.26 The `studentDataProvider` is created.

Modify the query as follows: modify query

```
select all Student.ssn,
           Student.firstName,
           Student.mi,
```

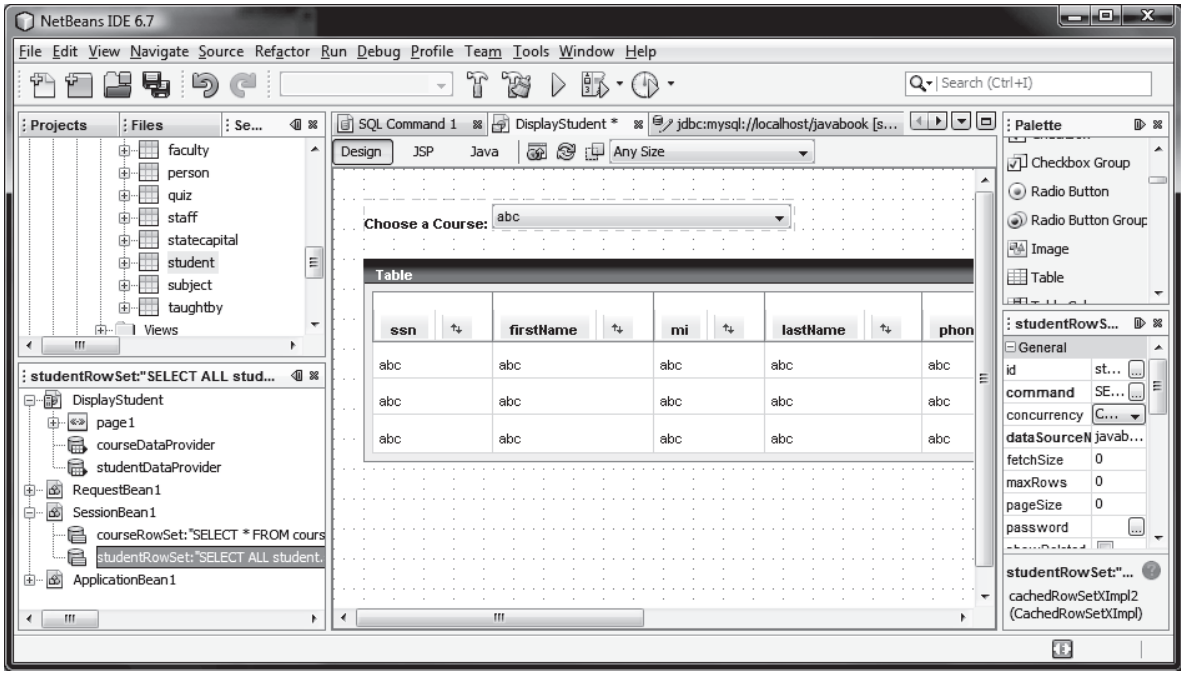


FIGURE 41.27 You can define a SQL statement in `studentRowSet`.

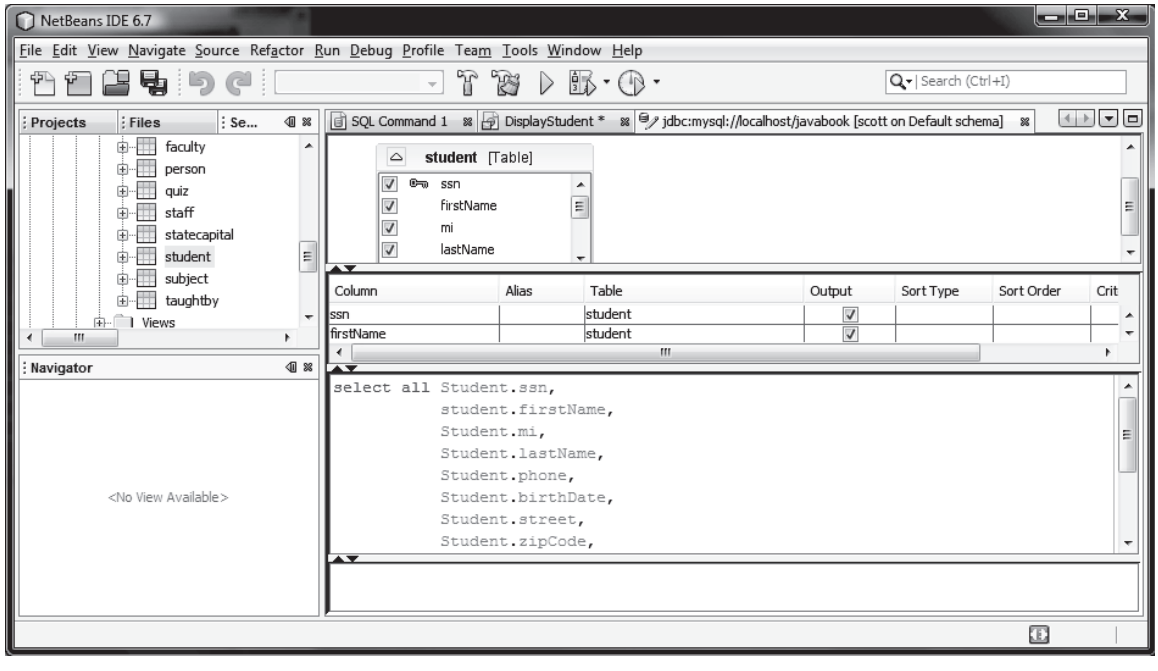


FIGURE 41.28 The query is visually displayed in the content pane.

```
Student.lastName,
Student.phone,
Student.birthDate,
Student.street,
Student.zipCode,
```

```
          Student.deptId
from Student, Enrollment, Course
where course.courseId = ?
          and student.ssn = enrollment.ssn
          and enrollment.courseId = course.courseId
```

Note that the `?` mark is a placeholder for the prepared statement. Close and reopen `studentRowSet`. You will see the new data diagram, as shown in Figure 41.29. If you don't see it, go back to the Design pane and refresh the project, then reopen `studentRowSet`.

prepared statement

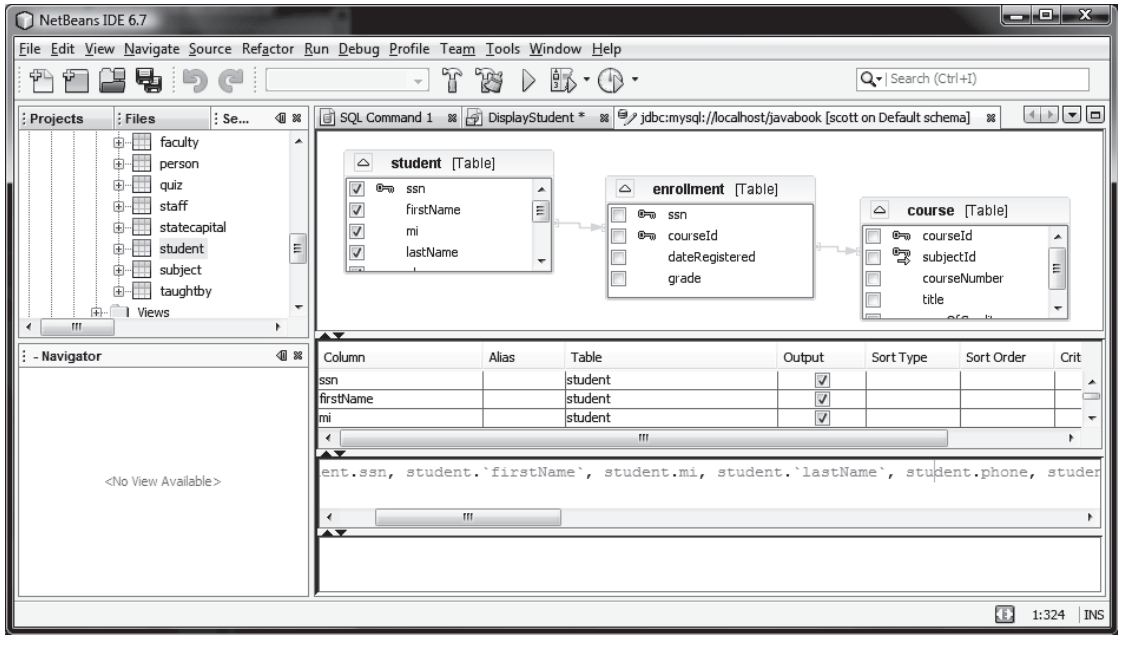


FIGURE 41.29 The tables and their relationship in the SQL query are visually displayed.

You can customize the table's layout by right-clicking the table in the Design pane and choosing *Table Layout* to display the Table Layout dialog box, as shown in Figure 41.30. Choose the selected columns and set the header for birthDate to Birth Date, as shown in Figure 41.30. Click *OK* to close the dialog box.

bind data

Drop a Message Group component to the upper right corner of the page. This component is useful to display diagnostic errors.

Message Group

Double-click the drop down list to generate the `ddCourse_processValueChange(ValueChangeEvent event)` handler and implement it as follows:

```
1 public void ddCourse_processValueChange(ValueChangeEvent e) {
2   try {
3     getSessionBean1().getStudentRowSet().setObject(
4       1, ddCourse.getSelected());
5     studentDataProvider.refresh();
6   }
7   catch (Exception ex) {
8     error(ex.toString());
9   }
10 }
```

set **courseId**

The `studentRowSet` object can be obtained using `setSessionBean1().getStudentRowSet()` (line 3). `studentRowSet` object is an instance of JDBC `RowSet`. The

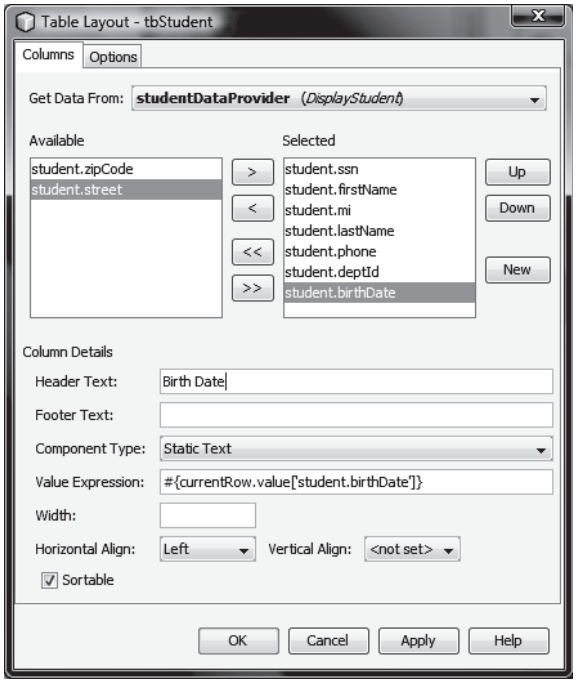


FIGURE 41.30 You can customize the layout of the table.

`setObject` method sets the parameter value (line 4). The `refresh()` method executes the query for the row set.

The `error` method displays the error message in the Message Group component (line 8). The Message Group component is not displayed if the `error` method is not called.

override **prerender**

Replace the body of the `prerender` method with the following code to ensure that the table is properly displayed initially.

```
public void prerender() {
  try {
    if (ddCourse.getSelected() == null) {
      courseDataProvider.cursorFirst();
      getSessionBean1().getStudentRowSet().setObject(
        1, courseDataProvider.getValue("Course.courseId"));
      studentDataProvider.refresh();
    }
  }
  catch (Exception ex) {
    error(ex.toString());
  }
}
```

auto-submit

Right-click on the drop down list in the Design pane to choose *Auto-Submit on Change*. In the Properties window, the following code appears in the `onchange` property:

```
webui.suntheme.common.timeoutSubmitForm(this.form, 'ddCourse');
```

Now when the user changes a value in the drop down list, the Web browser automatically submits the change.

Run the program using the URL http://localhost:8080/jsfdemo/faces/DisplayStudent.jsp. You will see the user interface, as shown in Figure 41.19.

41.6 Session Tracking

Chapter 40, "JSP," introduced session tracking using JavaBeans by sharing the JavaBeans objects among different pages. You can specify the JavaBeans objects at the application scope, session scope, page scope, or request scope. JSF supports session tracking using JavaBeans at the application scope, session scope, and request scope.

Consider the following example that prompts the user to guess a number. When the page starts, the program randomly generates a number between 0 and 99. This number is stored in the session. When the user enters a guess, the program checks the guess with the random number in the session and tells the user whether the guess is too high, too low, or just right, as shown in Figure 41.31.

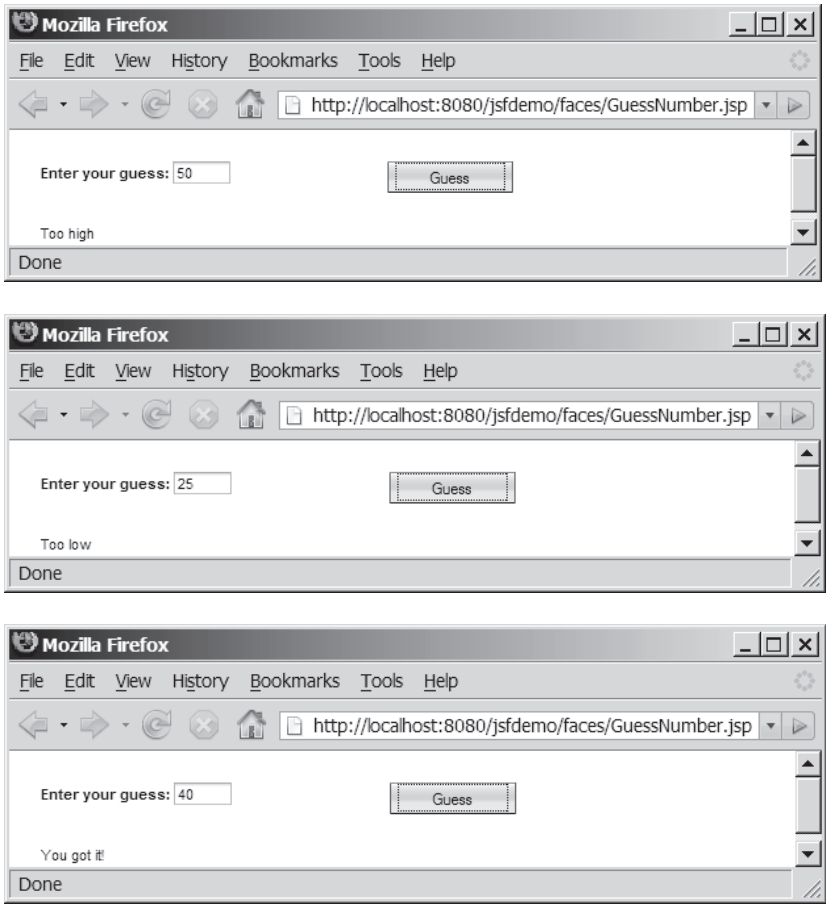


FIGURE 41.31 The user enters a guess and the program displays the result.

Create a new page named `GuessNumber` and create the user interface by dropping a text field with `id tfGuess` and `label` "Enter your guess: " and a Button with `id btGuess`, and a Static Text with `id stResponse` for displaying the program's response to the user's guess, as shown in Figure 41.32. Add binding attributes for `tfGuess` and `stResponse`.

create **GuessNumber** page

create UI

To store the random number in the session, create a property named `number` in `SessionBean1.java` with get and set methods as follows:

session bean property

```
int number;

public int getNumber() {
  return number;
}
```

```
public void setNumber(int number) {
  this.number = number;
}
```

SessionBean1.java is in the Source Packages folder in the **jsfdemo** project.

Add the following highlighted code in the constructor of SessionBean1.java, as shown in Figure 41.33:

create random number

```
public SessionBean1() {
  number = (int)(Math.random() * 100);
}
```

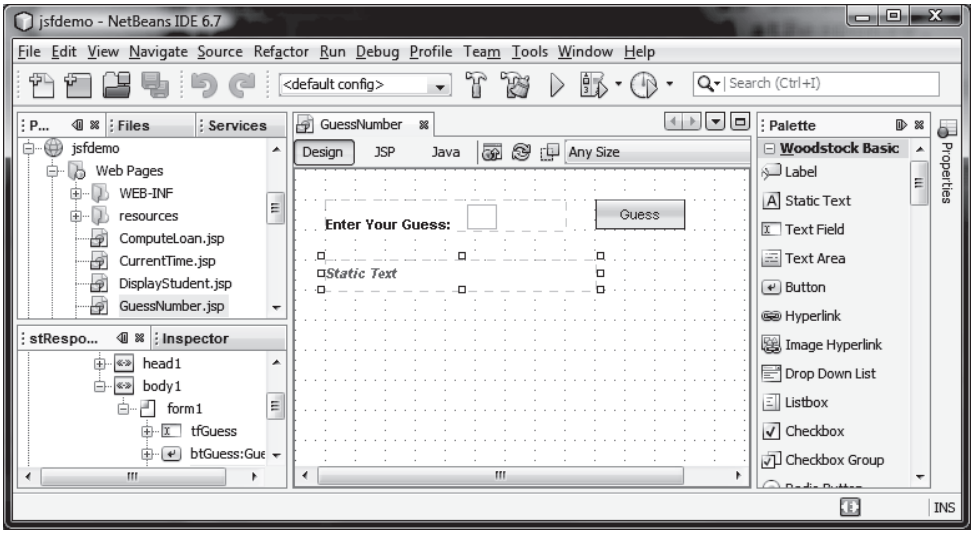


FIGURE 41.32 The user interface is created in the Design pane.

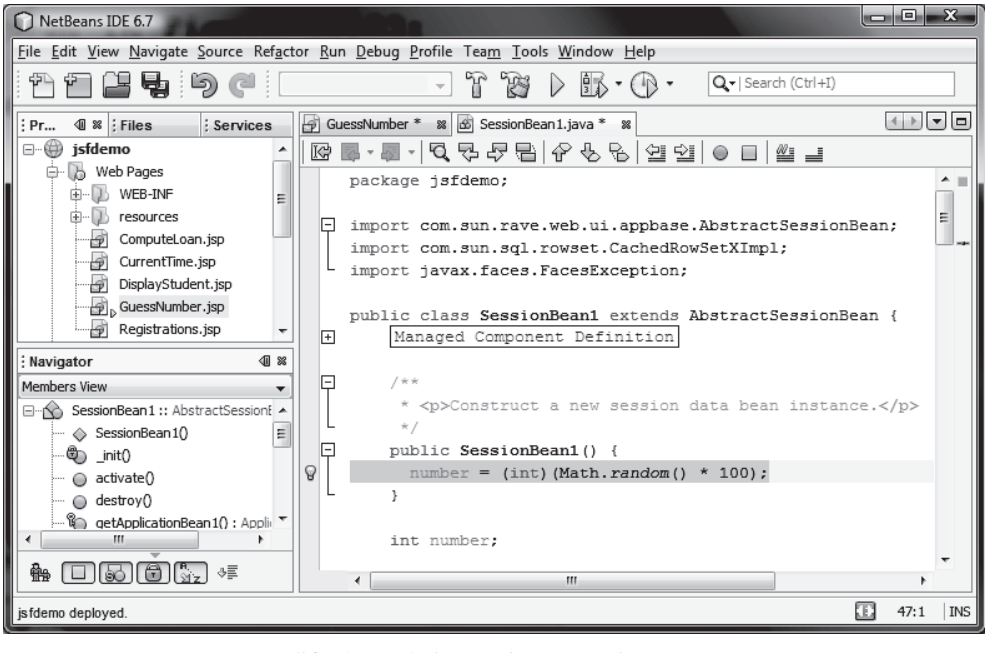


FIGURE 41.33 You can modify the code in SessionBean1.java.

Double-click the Guess button in the Design to generate the handler `btGuess_action()` and implement the method as follows:

implement button handler

```
1 public String btGuess_action() {
2   int guess = Integer.parseInt(tfGuess.getText().toString().trim());
3
4   if (guess < getSessionBean1().getNumber())
5     stResponse.setText("Too low");
6   else if (guess > getSessionBean1().getNumber())
7     stResponse.setText("Too high");
8   else
9     stResponse.setText("You got it!");
10
11  return null;
12 }
```

get guess

too low

too high

A session bean object is automatically created when a session starts. The `getSessionBean1()` method (lines 4, 6) returns the session bean object.

`getSessionBean1()`

41.7 Validating Input

In the preceding `GuessNumber` page, an error would occur if you did not enter a guess or you entered a string rather than an integer before clicking the *Guess* button. A simple way to fix the problem is to check the text field before processing any event. JSF provides several convenient and powerful ways for input validation. You can perform validation to ensure that the required field has been filled and the input is an integer for the preceding example.

Consider the following example that displays a form for collecting user input as shown in Figure 41.34. All text fields in the form must be filled. If not, error messages are displayed. The SSN must be formatted corrected. If not, an error is displayed. If all input is correct, clicking *Submit* displays the result in a static text, as shown in Figure 41.35.

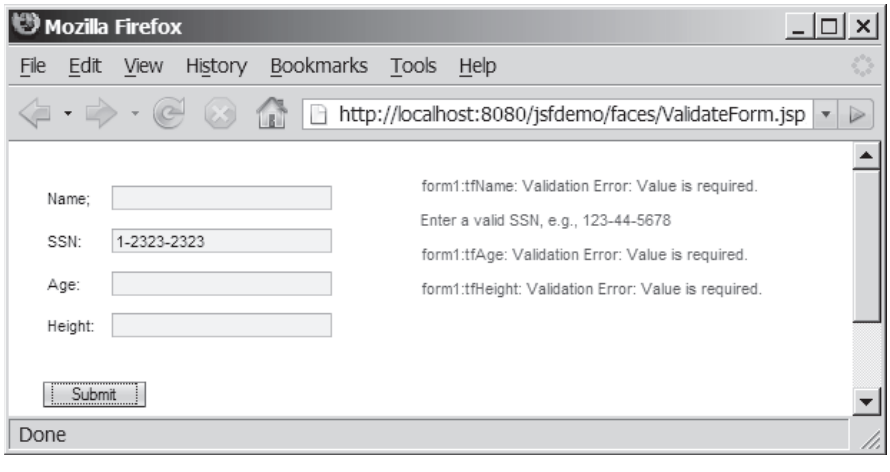


FIGURE 41.34 The error messages are displayed if input is invalid.

Create a new page named `ValidateForm` and create the user interface as follows:

create `ValidateForm` page
create UI

1. Drop a GridPanel to the Design and set its `columns` to `2`. Add four Static Texts, four Text Fields, and a Button to the grid panel, as shown in Figure 41.36. Set the `id`

create a `GridPanel`

property of the text fields to `tfName`, `tfSSN`, `tfAge`, and `tfHeight`, respectively. Set the button's `id` to `btSubmit` and `label` to Submit.

escape property

2. Drop a Static Text below the grid panel and set its `id` to `stResult`. Set its `escape` property to `false` so you can display HTML contents in the static text.

add binding attribute

3. Add binding attributes for `tfName`, `tfSSN`, `tfAge`, `tfHeight`, and `stResult`.

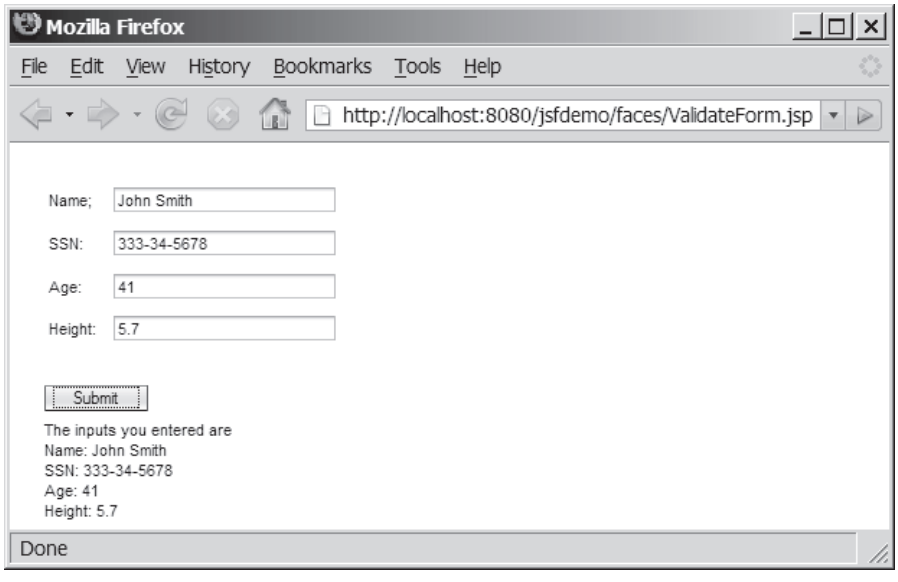


FIGURE 41.35 The Submit button is processed after input is validated.

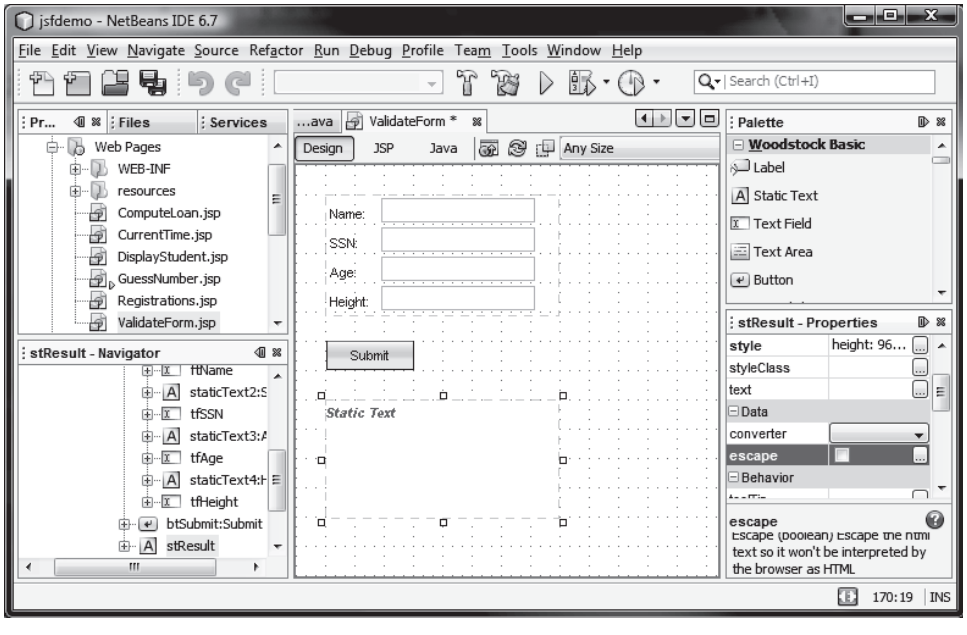


FIGURE 41.36 The user interface is created in the Design pane.

You can use the Message component to display input errors. Drop four Message components next to each text field, as shown in Figure 41.37. To associate a Message component with a text field, choose an appropriate value in the Message's `for` property, as shown in Figure 41.37. Message will automatically display an error message when input validation fails in the associated text field.

display error in Message

for property

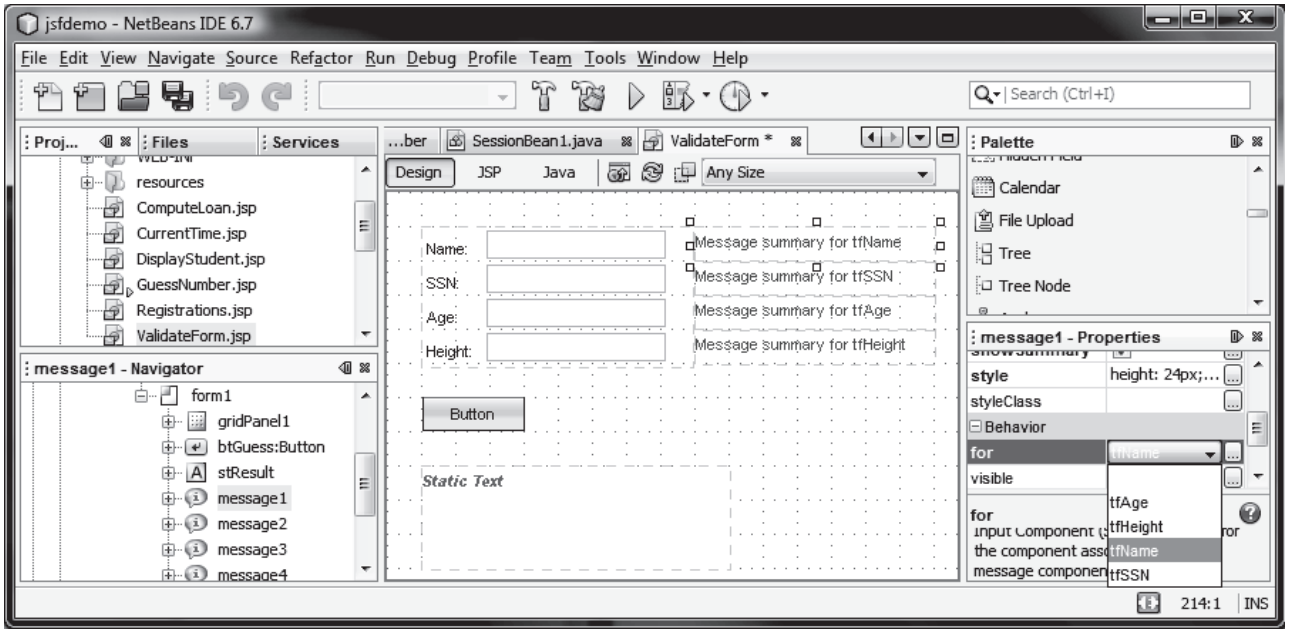


FIGURE 41.37 Four Message components are created in the Design.

For each text field, set its `required` property `true` to indicate that an input value is required for the field.

`required` property

You can validate the length of the input using the `LengthValidator`. Suppose the name must have minimum `1` character and maximum `10` characters. To set its length validator, choose `new LengthValidator()` in the `validatorExpression` property for `tfName`. You will see `lengthValidator1` appearing in the Navigator pane, as shown in Figure 41.38. Set the minimum and maximum properties for `lengthValidator1` to `1` and `10`.

`LengthValidator`

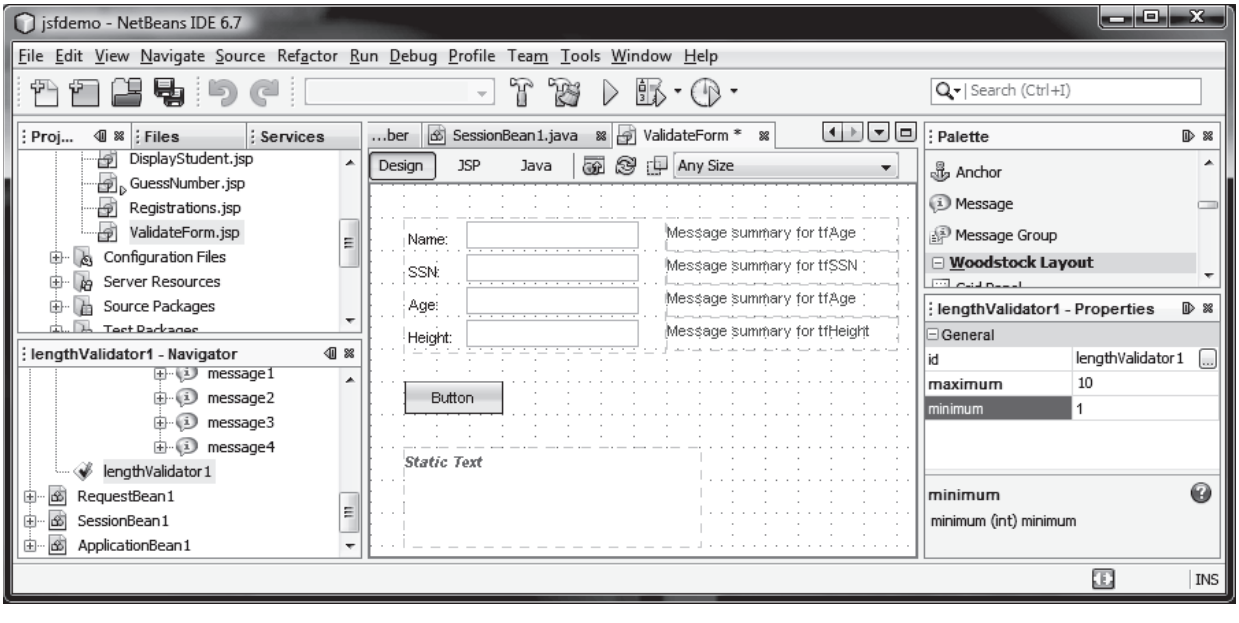


FIGURE 41.38 You can specify the `maximum` and `minimum` properties for a `lengthValidator`.

regular expression

The format for SSN is `ddd-dd-dddd`, where `d` is a digit. You need to create a custom validator and use a regular expression to validate a SSN. Right-click `tfSSN` in the Design, and choose **Edit Event Handler › validate** to generate the method named `tfSSN_validate` in the Java source page. Implement the method as follows:

```
1 public void tfSSN_validate(FacesContext context,
2     UIComponent component, Object value) {
3   String ssn = String.valueOf(value);
4
5   if (!ssn.matches("[\\d]{3}-[\\d]{2}-[\\d]{4}")) {
6     throw new ValidatorException(new FacesMessage(
7       "Enter a valid SSN, e.g., 123-44-5678"));
8   }
9 }
```

regular expression
throw `ValidatorException`

`ValidatorException`

The program checks whether the ssn is correctly formatted. If not, a `ValidatorException` is thrown in line 6.

Tip

Right-click on ValidatorException in the Editor pane to display a context menu and choose *Fix Imports* to import `javax.faces.validator.ValidatorException`.

Similarly, you can write the code to ensure that age is an integer between `1` and `135` and height is a double value between `2.0` and `10.0`.

implement Submit button

The *Submit* button can be implemented as follows:

```
1 public String btSubmit_action() {
2   stResult.setText("The inputs you entered are <br />" +
3     "Name: " + tfName.getText() + "<br />" +
4     "SSN: " + tfSSN.getText() + "<br />" +
5     "Age: " + tfAge.getText() + "<br />" +
6     "Height: " + tfHeight.getText() + "<br />");
7
8   return null;
9 }
```

41.8 Virtual Forms

When you submit a form, all input components in the form participate in the submission. However, not all these components are needed for a particular submission. For example, when you click the *Compute Loan* button in the form in Figure 41.39, only the text fields for loan amount, interest rate, and number of years need to be submitted. When you click the *Compute Body Mass Index* button, only the text fields for weight and height need to be submitted.

virtual form

To improve efficiency, you may define a virtual form to group input components with a submission component (e.g., a button or a drop down list with autosubmit enabled). The input components participate in the virtual form. When the user interacts with a submission component that submits a virtual form, the input components in the virtual form are submitted while the other input components not defined in the virtual form on the page are ignored.

create UI

Let us build an application for Figure 41.39 using virtual forms. Create a new page named `VirtualFormDemo` and create the user interface, as shown in Figure 41.40. The text fields are named `tfLoanAmount`, `tfAnnualInterestRate`, `tfNumberOfYears`, `tfWeight`, and `tfHeight`. Two buttons are named `btComputeLoan` and `btComputeBMI`. Two static texts are named `stLoanResult` and `stBMIResult`. Add binding attributes for all text fields and static texts.

We are going to create two virtual forms: one for grouping the components for computing loan and the other for grouping the components for computing body mass index.

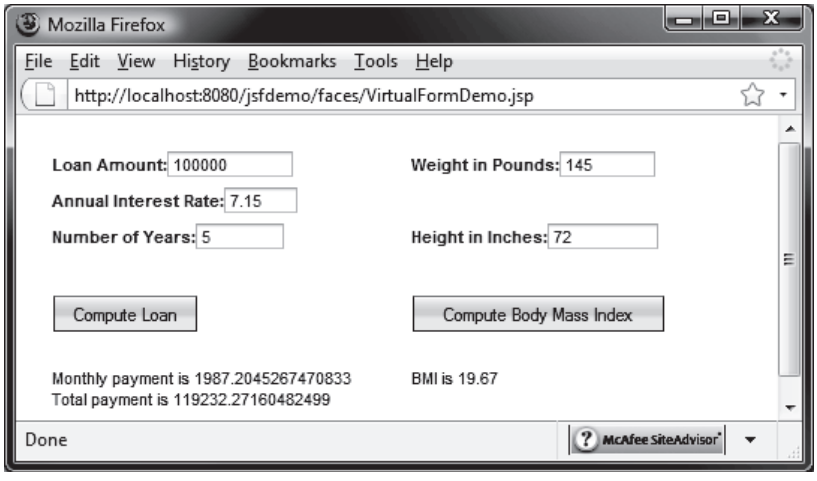


FIGURE 41.39 The application lets you compute loan and body mass index.

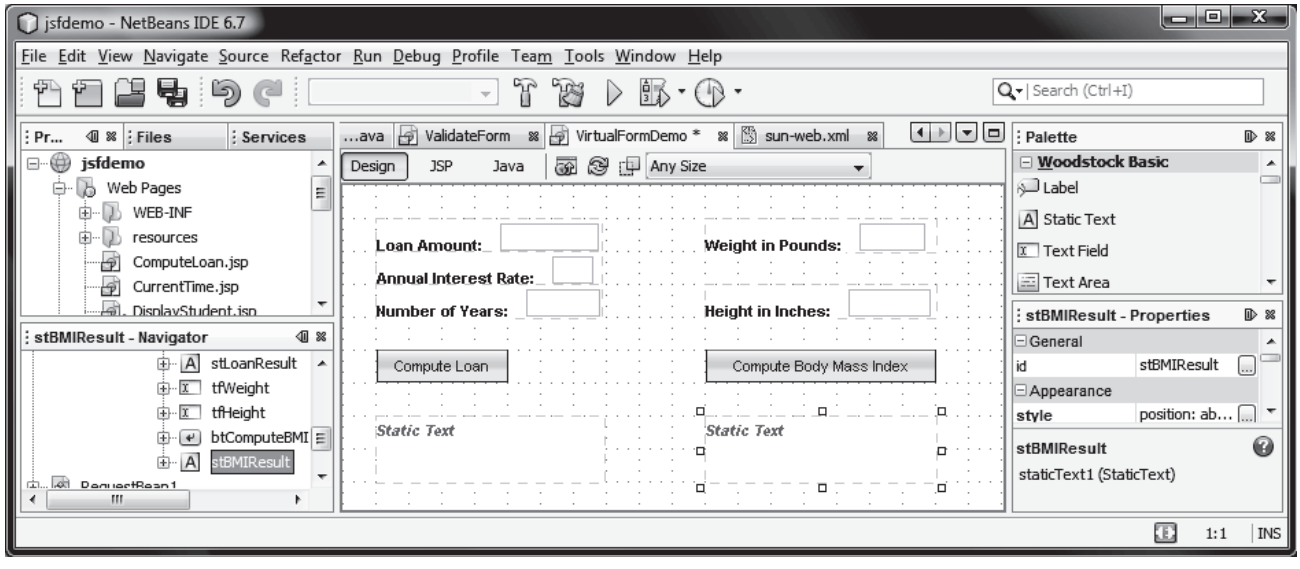
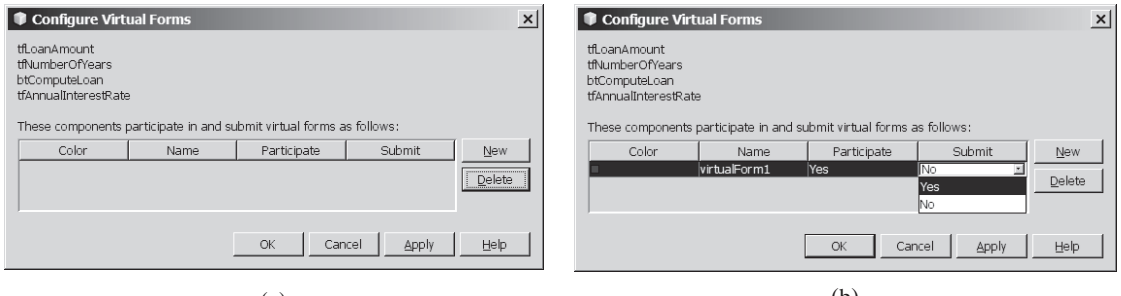


FIGURE 41.40 The user interface is created for computing loan and body mass index.

1. Ctrl-click to select `tfLoanAmount`, `tfAnnualInterestRate`, `tfNumberOfYears`, and `btComputeLoan`. Right-click one of the selected components to display a context menu and choose Configure Virtual Forms to display the Configure Virtual Form dialog box, as shown in Figure 41.41(a). You should see `tfLoanAmount`, `tfAnnualInterestRate`,



(a) (b)

FIGURE 41.41 You can create/delete/configure virtual forms.

tfNumberOfYears, and btComputeLoan in the upper left corner in the dialog box. Click the *New* button to create a virtual form named virtualForm1. Choose *Yes* in both the Participate and Submit columns, as shown in Figure 41.41(b).

2. Ctrl-click to select tfWeight, tfHeight, and btComputeBMI. Right-click one of the selected components to display a context menu and choose Configure Virtual Forms to display the Configure Virtual Form dialog box. You should see tfWeight, tfHeight, and btComputeBMI in the upper left corner in the dialog box. Click the *New* button to create a virtual form named virtualForm2. Choose *Yes* in both the Participate and Submit columns, as shown in Figure 41.42.

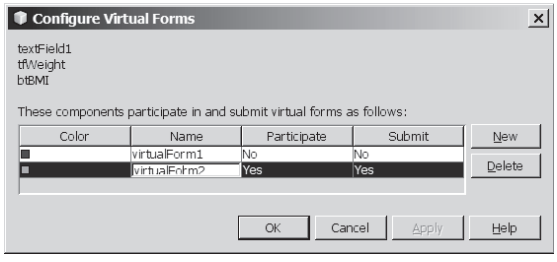


FIGURE 41.42 Two virtual forms are created.

Implement the handlers for the two buttons as follows:

```
public String btComputeLoan_action() {
  double loanAmount =
    Double.parseDouble(tfLoanAmount.getText().toString().trim());
  int numberOfYears =
    Integer.parseInt(tfNumberOfYears.getText().toString().trim());
  double annualInterestRate = Double.parseDouble(
    tfAnnualInterestRate.getText().toString().trim());

  chapter41.Loan loan = new chapter41.Loan(
    annualInterestRate, numberOfYears, loanAmount);

  stLoanResult.setText("Monthly payment is " +
    loan.getMonthlyPayment() + " Total payment is " +
    loan.getTotalPayment());

  return null;
}

public String btBMI_action() {
  final double KILOGRAMS_PER_POUND = 0.45359237;
  final double METERS_PER_INCH = 0.0254;

  double weight = Double.parseDouble(tfWeight.getText().toString());
  double height = Double.parseDouble(tfHeight.getText().toString());

  double bmi = weight * KILOGRAMS_PER_POUND /
    ((height * METERS_PER_INCH) * (height * METERS_PER_INCH));

  stBMIResult.setText("BMI is " + (int)Math.round(bmi * 100) / 100.0);

  return null;
}
```

Now you can run the program (see Figure 41.39). Clicking the *Compute Loan* button will submit the text fields for loan amount, annual interest rate, and number of years. Clicking *Compute BMI* will submit the text fields for weight and height.

Note

Supplement V.F, "More on JSF and Visual Web Development," provides additional examples on visual Web development, such as performing database inserts, updates, and deletes, and working with Ajax.

more supplemental examples

Key Terms

`init` 41–5
`preprocess` 41–5
`prerender` 41–5
`destroy` 41–6
data binding 41–14
palette window 41–3
UI container 41–11

Chapter Summary

1. JSF enables you to use a visual design tool to develop UI for Web applications.
2. A JSF page has three tabs: Design, JSP, and Java. The Design tab enables you to visually design UI. The JSP tab enables you to view and modify the JSP script for the page. The Java tab enables you to view and modify the page bean Java source code.
3. JSF uses the life-cycle methods `init`, `preprocess`, `prerender`, and `destroy` to control the execution of a JSF page.
4. JSF contains Swinglike UI components. You can click and drop these components to the Design pane to create UI.
5. JSF contains Grid Panel, Group Panel, and Layout Panel containers for grouping components.
6. You can create a database connection from the Databases node in the Services pane and bind database queries with JSF UI components.
7. You can perform session tracking using the `SessionBean`.
8. You can validate input to ensure that the required field has been filled and the input is formatted correctly.
9. You can use virtual forms to group the participants of a form to improve efficiency.

Review Questions

Sections 41.1–41.2

41.1 What is JSF?
41.2 How do you create a JSF project in NetBeans?
41.3 How do you create a JSF page in a JSF project?
41.4 Describe the life-cycle methods `init`, `preprocess`, `prerender`, and `destroy`.
41.5 Each JSF page has a .jsp file. What is in the JSP file? Can you modify it directly?
41.6 Each JSF page has a page bean file. What is in the page bean file? Can you modify it directly?

Section 41.3

41.7 How do you set the initial value and display names in a drop down list and in a list box?
41.8 How do you specify radio buttons and their names in a radio button group?
41.9 How do you specify check box buttons and their names in a check box button group?
41.10 How do you hide a component so that it will not be displayed?
41.11 What method should be used to obtain the input from a text field or a text area?

41.12 What method should be used to obtain the selected item(s) from a drop down list or a list box?

41.13 Can you create two forms in one JSF page?

Section 41.4

41.14 What containers are available in JSF?

41.15 If the order of the components in a container is not correct, how do you reorder them?

Section 41.5

41.16 How do you create a database connection?

41.17 How do you create a data provider?

41.18 How do you modify a row set?

41.19 How do you create a table and bind a data provider with the table?

Section 41.6

41.20 How does JSP implement session tracking?

41.21 How do you create a JavaBeans property?

41.22 How do you obtain a JavaBeans property in the page bean file?

Section 41.7

41.23 What component can be used to automatically display the error message if invalid input is entered?

41.24 How do you set a Message component to associate with an input component?

41.25 What property for an input component should be checked to specify that an input is required?

41.26 How do you check the format of input using regular expressions?

Programming Exercises

41.1* (*Factorial table in JSF*) Rewrite Exercise 40.1 using JSF. Drop a Static Text to display the result. Set its escape property to false to display it as HTML contents.

41.2* (*Multiplication table in JSP*) Rewrite Exercise 40.2 using JSF.

41.3* (*Calculator*) Write a calculator to perform addition, subtraction, multiplication, and division, as shown in Figure 41.43.

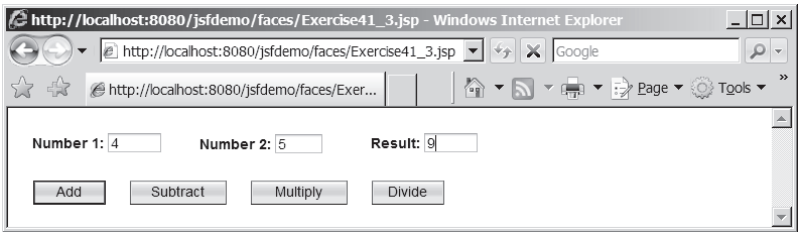


Figure 41.43 This JSF application enables you to perform addition, subtraction, multiplication, and division.

41.4* (*Calculating tax in JSP*) Rewrite Exercise 40.4 using JSF.

41.5* (*Multiple-question opinion poll*) Rewrite Exercise 40.13 using JSF.

41.6* (*Addition quiz*) Rewrite Exercise 40.14 using JSF.

41.7* (*Subtraction quiz*) Rewrite Exercise 40.15 using JSF.

41.8* (*Guessing birth date*) Rewrite Exercise 40.16 using JSF.

41.9* (*Guessing the capitals*) Rewrite Exercise 40.17 using JSF.