

CHAPTER 42

WEB SERVICES

Objectives

- To describe what a Web service is (§42.1).
- To create a Web service class (§42.2).
- To publish and test a Web service (§42.3).
- To create a Web service client reference (§42.4).
- To explain the role of WSDL (§42.4).
- To pass arguments of object type in a Web service (§42.5).
- To discover how a client communicates with a Web service (§42.5).
- To describe what SOAP requests and SOAP responses are (§42.5).
- To track a session in Web services (§42.6).



42.1 Introduction

Web service is a technology that enables programs to communicate through HTTP on the Internet. Web services enable a program on one system to invoke a method in an object on another system. You can develop and use Web services using any languages on any platform. Web services are simple and easy to develop.

Web services run on the Web using HTTP. There are several APIs for Web services. A popular standard is the *Simple Object Access Protocol* (SOAP), which is based on XML. The computer on which a Web service resides is referred to as a server. The server needs to make the service available to the client, known as *publishing a Web service*. Using a Web service from a client is known as *consuming a Web service*.

A client interacts with a Web service through a *proxy object*. The proxy object facilitates the communication between the client and the Web service. The client passes arguments to invoke methods on the proxy object. The proxy object sends the request to the server and receives the result back from the server, as shown in Figure 42.1.

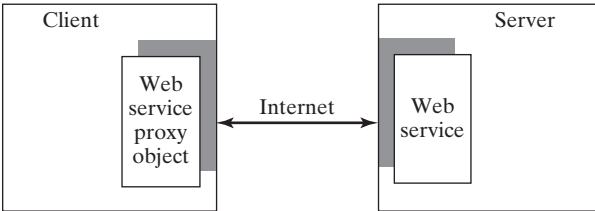


FIGURE 42.1 A proxy object serves as a facilitator between a client and a Web service.

42.2 Creating Web Services

There are many tools for creating Web services. This book demonstrates creating Web services using NetBeans.

This section creates a Web service for obtaining student scores. A Web service is a class that contains the methods for the client to invoke. Name the class **ScoreWebService** with a method named **findScore(String name)** that returns the score for a student.

First you need to create a *Web project* using the following steps:

1. Choose **File > New Project** to display the New Project dialog box.
2. In the New Project dialog box, choose **Web** in the Categories pane and choose **Web Application** in the Projects pane. Click *Next* to display the New Web Application dialog box.
3. Enter **WebServiceProject** as the project name, specify the location where you want the project to be stored, and select GlassFish V2 as the server and Java EE 5 as the Java EE version. Click *Finish* to create the project.

Now you can create the **ScoreService** class in the project as follows:

1. Right-click the **WebServiceProject** in the Project pane to display a context menu. Choose **New > Web Service** to display the New Web Service dialog box.
2. Enter **ScoreService** in the Web Service Name field and enter **chapter42** in the Package field. Click *Finish* to create **ScoreService**.
3. Complete the source code as shown in Listing 42.1.

platform independent
language independent

SOAP

publishing Web services
consuming Web services
proxy object

Web service tool

NetBeans Web project

create Web service class

LISTING 42.1 ScoreService.java

```

1 package chapter42;
2
3 import java.util.HashMap;
4 import javax.jws.WebService; // For annotation @WebService
5 import javax.jws.WebMethod; // For annotation @WebMethod
6
7 @WebService(name = "ScoreService", serviceName = "ScoreWebService")
8 public class ScoreService {
9     // Stores scores in a map indexed by name
10    private HashMap<String, Double> scores =
11        new HashMap<String, Double>();
12
13    public ScoreService() {
14        scores.put("John", 90.5);
15        scores.put("Michael", 100.0);
16        scores.put("Michelle", 98.5);
17    }
18
19    @WebMethod(operationName = "findScore")
20    public double findScore(String name) {
21        Double d = scores.get(name);
22
23        if (d == null) {
24            System.out.println("Student " + name + " is not found ");
25            return -1;
26        }
27        else {
28            System.out.println("Student " + name + "'s score is "
29                + d.doubleValue());
30            return d.doubleValue();
31        }
32    }
33 }

```

import for @WebService
import for @WebMethod
define WebService
define WebMethod

Lines 4–5 import the annotations used in the program in lines 7 and 19. Annotation is a new feature in Java, which enables you to simplify coding. The compiler will automatically generate the code for the annotated directives. So, it frees the programmer from writing the detailed *boilerplate code* that could be generated mechanically. The annotation (line 7)

what is annotation?

boilerplate code

```
@WebService(name = "ScoreService", serviceName = "ScoreWebService")
```

tells the compiler that the class **ScoreService** is associated with the Web service named **ScoreWebService**.

The annotation (line 19)

```
@WebMethod(operationName = "findScore")
```

indicates that **findScore** is a method that can be invoked from a client.

The **findScore** method returns a score if the name is in the hash map. Otherwise, it returns **-1.0**.

You can manually type the code for the service, or create it from the Design tab, as shown in Figure 42.2.

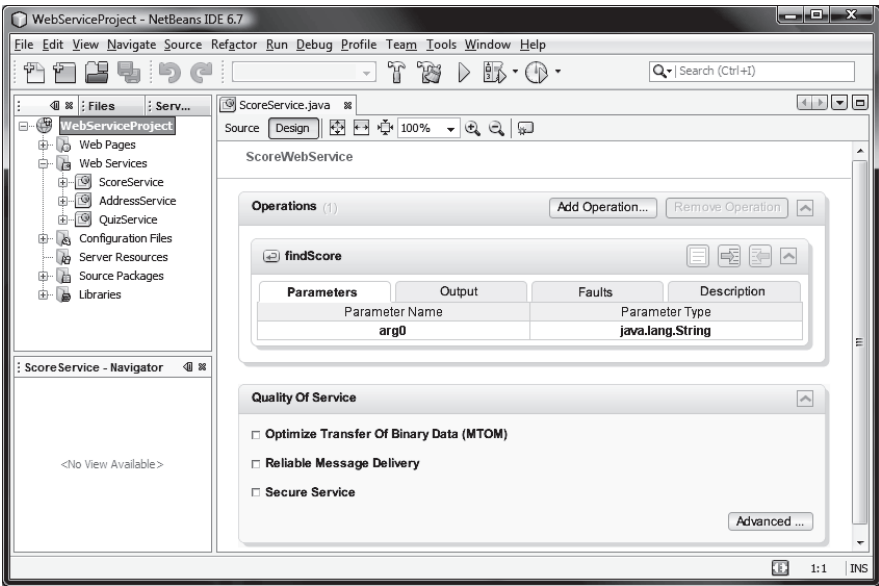


FIGURE 42.2 The services can also be created from the Design pane.

42.3 Deploying and Testing Web Services

publishing Web services

After a Web service is created, you need to deploy it for clients to use. Deploying Web services is also known as *publishing Web services*. To deploy it, right-click the **WebServiceProject** in the Project to display a context menu and choose **Deploy**. This command will first undeploy the service if it was deployed and then redeploy it.

Now you can test the Web service by entering the follow URL in a browser, as shown in Figure 42.3.

`http://localhost:8080/WebServiceProject/ScoreWebService?Tester`



FIGURE 42.3 The test page enables you to test Web services.

Note that **ScoreWebService** is the name you specified in line 7 in Listing 42.1. This Web service has only one remote method named **findScore**. You can define an unlimited number of remote methods in a Web service class. If so, all these methods will be displayed in the test page.

To test the **findScore** method, enter **Michael** and click **findScore**. You will see that the method returns **100.0**, as shown in Figure 42.4.

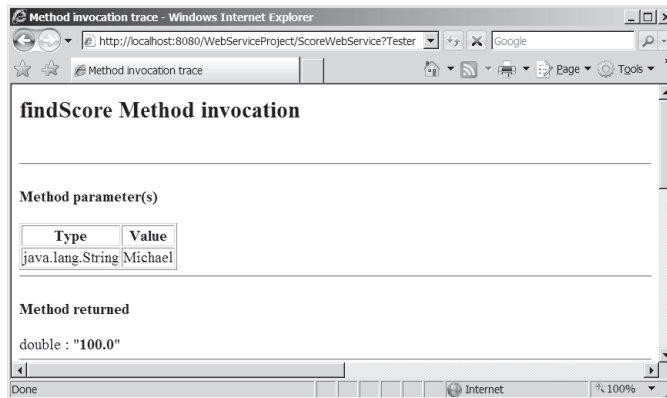


FIGURE 42.4 The method returns a test value.



Note

If your computer is connected to the Internet, you can test Web services from another computer by entering the following URL:

`http://host:8080/WebServiceProject/ScoreWebService?Tester`

Where *host* is the host name or IP address of the server on which the Web service is running. On Windows, you can find your IP address by typing the command **ipconfig**.

testing from another machine

ipconfig



Note

If you are running the server on Windows, the firewall may prevent remote clients from accessing the service. To enable it, do the following:

1. In the Windows control panel, click Windows Firewall to display the Windows Firewall dialog box.
2. In the Advanced tab, double-click Local Area Connection to display the Advanced Settings dialog box. Check *Web Server(HTTP)* to enable HTTP access to the server.
3. Click OK to close the dialog box.

Windows firewall



Note

If your application server is launched from NetBeans and you stop NetBeans, the application server is also stopped. To keep the application server live, you may launch it independently from the Windows *Start* button by choosing *All Programs, Sun Microsystems, Application Server, Start Default Server*. To shut it down, choose **Stop Default Server** from the same location.

keeping the server live

42.4 Consuming Web Services

After a Web service is published, you can write a client program to use it. A client can be any program (standalone application, applet, servlet/JSP/JSF application, or another Web service) and written in any language.

consuming Web services

We will use NetBeans to create a Web service client. Our client is a Java applet with a main method, so you can also run it standalone. The applet simply lets the user enter a name and displays the score, as shown in Figure 42.5.

Let us create a project for the client. The project named **ScoreWebServiceClientProject** can be created as follows:

1. Choose **File > New Project** to display the New Project dialog box.

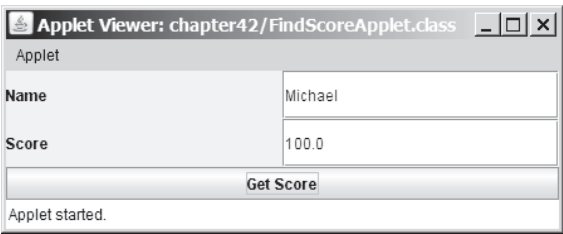


FIGURE 42.5 The applet client uses the Web service to find scores.

2. In the New Project dialog box, choose **Java** in the Categories pane and choose **Java Application** in the Projects pane. Click *Next* to display the New Java Application dialog box.
3. Enter **ScoreWebServiceClientProject** as the project name, specify the location where you want the project to be stored, and uncheck the *Create Main Class* check box. Click *Finish* to create the project.

Web service reference

You need to create a Web service reference to this project. The reference will enable you to create a proxy object to interact with the Web service. Here are the steps to create a Web service reference:

1. Right-click the **ScoreWebServiceClientProject** in the Project pane to display a context menu. Choose **New > Web Service Client** to display the New Web Service Client dialog box, as shown in Figure 42.6.

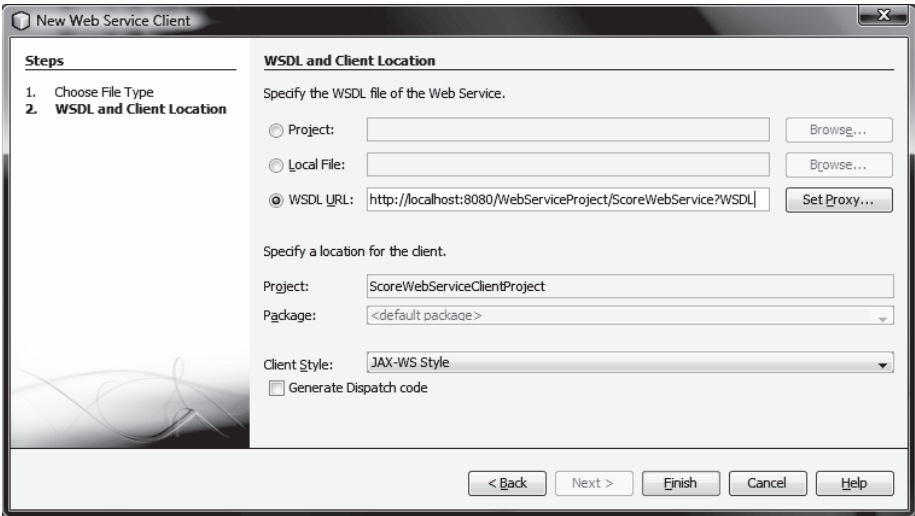


FIGURE 42.6 The New Web Service Client dialog box creates a Web service reference.

2. Check the *WSDL URL* radio button and enter `http://localhost:8080/WebServiceProject/ScoreWebService?WSDL` in the *WSDL URL* field.
3. Enter **chapter42** in the package name field and choose **JAX-WS** as the JAX version. Click *Finish* to generate the Web service reference.

Now you will see **ScoreWebService** created in the Web Service References folder in the Projects tab. The IDE has generated many supporting files for the reference. You can view all the generated .java files from the Files tab in the project pane, as shown in Figure 42.7. These files will be used by the proxy object to interact with the Web service.

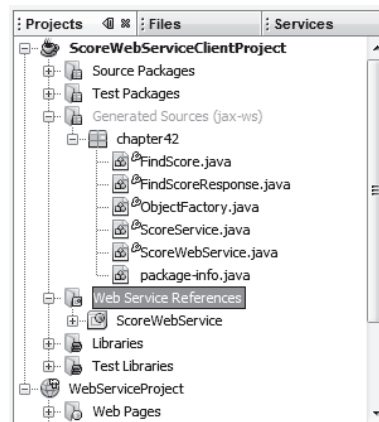


FIGURE 42.7 You can see the automatically generated boilerplate code for Web services in the Files tab in the client's project.



Note

When you created a Web service reference, you entered a WSDL URL, as shown in Figure 42.6. A .wsdl file is created under the Web Service References folder, as shown in Figure 42.8. So *what is WSDL?* WSDL stands for *Web Service Description Language*. A .wsdl file is an XML file that describes the available Web service to the client—i.e., the remote methods, their parameters and return value types, and so on.

what is WSDL?

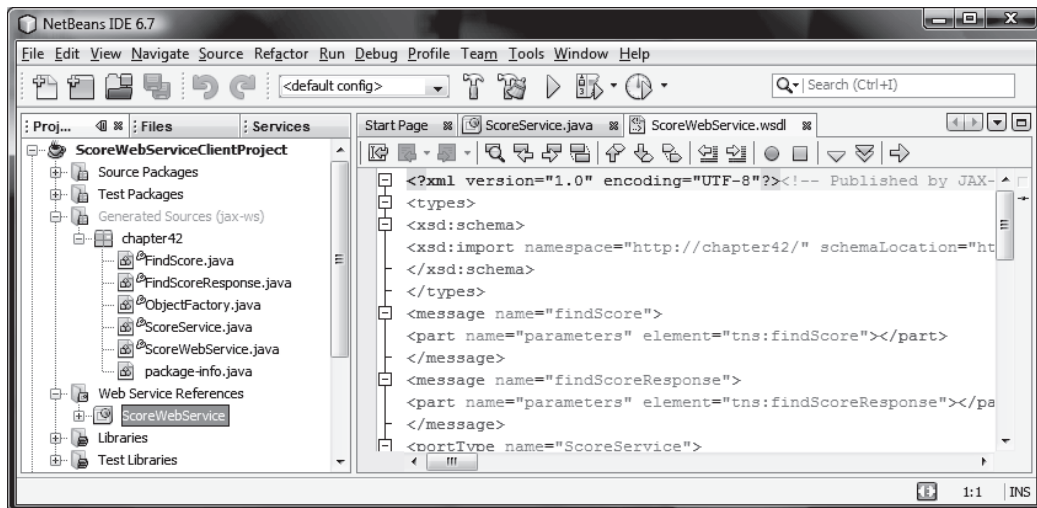


FIGURE 42.8 The .wsdl file describes Web services to clients.



Note

If the Web service is modified, you need to refresh the reference for the client. To do so, right-click the Web service node under Web Service References to display a context menu and choose **Refresh Client**.

refresh reference

Now you are ready to create an applet client for the Web service. Right-click the **ScoreWebServiceClientProject** node in the Project pane to display a context menu, and choose **New > JApplet** to create a Java applet named **FindScoreApplet** in package **chapter42**, as shown in Listing 42.2.

LISTING 42.2 FindScoreApplet.java

```

1 package chapter42;
2
3 import chapter42.ScoreWebService;
4 import chapter42.ScoreService;
5 import javax.swing.*;
6 import java.awt.*;
7 import java.awt.event.*;
8
9 public class FindScoreApplet extends JApplet {
10     // Declare a service object and a proxy object
11     private ScoreWebService scoreWebService = new ScoreWebService();
12     private ScoreService proxy = scoreWebService.getScoreServicePort();
13
14     private JButton jbtGetScore = new JButton("Get Score");
15     private JTextField jtfName = new JTextField();
16     private JTextField jtfScore = new JTextField();
17
18     public void init() {
19         JPanel jPanel1 = new JPanel();
20         jPanel1.setLayout(new GridLayout(2, 2));
21         jPanel1.add(new JLabel("Name"));
22         jPanel1.add(jtfName);
23         jPanel1.add(new JLabel("Score"));
24         jPanel1.add(jtfScore);
25
26         add(jbtGetScore, BorderLayout.SOUTH);
27         add(jPanel1, BorderLayout.CENTER);
28
29         jbtGetScore.addActionListener(new ActionListener() {
30             public void actionPerformed(ActionEvent evt) {
31                 getScore();
32             }
33         });
34     }
35
36     private void getScore() {
37         try {
38             // Get student score
39             double score = proxy.findScore(jtfName.getText().trim());
40
41             // Display the result
42             if (score < 0)
43                 jtfScore.setText("Not found");
44             else
45                 jtfScore.setText(new Double(score).toString());
46         }
47         catch (Exception ex) {
48             ex.printStackTrace();
49         }
50     }
51 }

```

create a service object
create a proxy object

invoke remote method

main method omitted

The program creates a Web service object (line 11) and creates a proxy object (line 12) to interact with the Web service.

To find a score for a student, the program invokes the remote method **findScore** on the proxy object (line 39).

42.5 Passing and Returning Arguments

In the preceding example, a Web service client that you created invokes the **findScore** method with a string argument, and the Web service executes the method and returns a score as a **double** value. How does this work? It is the *Simple Object Access Protocol* (SOAP) that facilitates communications between the client and server.

SOAP

SOAP is based on XML. The message between the client and server is described in XML. Figure 42.9 shows the SOAP request and SOAP response for the **findScore** method.

SOAP request

SOAP response

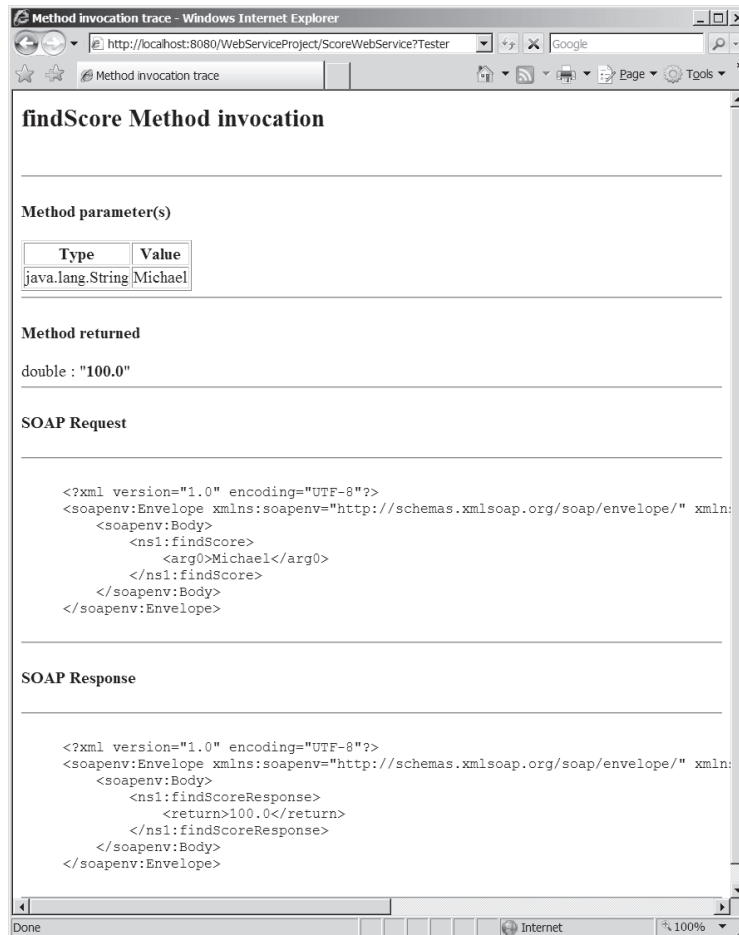


FIGURE 42.9 The client request and server response are described in XML.

When invoking the **findScore** method, a *SOAP request* is sent to the server. The request contains the information about the method and the argument. As shown in Figure 42.9, the XML text

```
<ns1:findScore>
  <arg0>Michael</arg0>
</ns1:findScore>
```

specifies that the method **findScore** is called with argument **Michael**.

Upon receiving the SOAP request, the Web service parses it. After parsing it, the Web service invokes an appropriate method with specified arguments (if any) and sends the response back in a *SOAP response*. As shown in Figure 42.9, the XML text

```
<ns1:findScoreResponse>
  <return>100.0</return>
</ns1:findScoreResponse>
```

specifies that the method returns **100.0**.

The proxy object receives the SOAP response from the Web service and parses it. This process is illustrated in Figure 42.10.

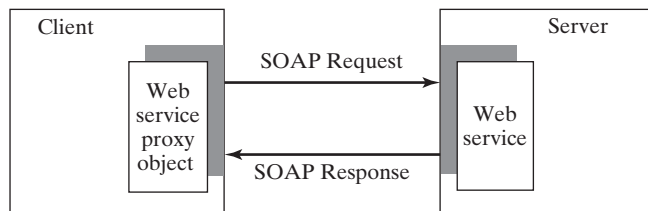


FIGURE 42.10 A proxy object sends SOAP requests and receives SOAP responses.

Can you pass an argument of any type between a client and a Web service? No. SOAP supports only primitive types, wrapper types, arrays, **String**, **Date**, **Time**, **List**, and several other types. It also supports certain custom classes. An object that is sent to or from a server is serialized into XML. The process of serializing/deserializing objects, called *XML serialization/deserialization*, is performed automatically. For a custom class to be used with Web methods, the class must meet the following requirements:

1. The class must have a no-arg constructor.
2. Instance variables that should be serialized must have public get and set methods. The classes of these variables must be supported by SOAP.

To demonstrate how to pass an object argument of a custom class, Listing 42.3 defines a Web service class named **AddressService** with two remote methods:

- **getAddress(String firstName, String lastName)** that returns an **Address** object for the specified **firstName** and **lastName**.
- **storeAddress(Address address)** that stores a **Student** object to the database.

Address information is stored in a table named **Address** in the database. The **Address** class was defined in Listing 39.12, **Address.java**. An **Address** object can be passed to or returned from a remote method, since the **Address** class has a no-arg constructor with get and set methods for all its properties.

Here are the steps to create a Web service named **AddressService** and the **Address** class in the project.

1. Right-click the **WebServiceProject** node in the project pane to display a context menu. Choose **New** > **Web Service** to display the New Web Service dialog box.
2. In the Web Service Name field, enter **AddressService**. In the Package field, enter **chapter42**. Click **Finish** to create the service class.
3. Right-click the **WebServiceProject** node in the project pane to display a context menu. Choose **New** > **Java Class** to display the New Java Class dialog box.
4. In the Class Name field, enter **Address**. In the Package field, enter **chapter39**. Click **Finish** to create the class.

XML serialization
XML deserialization

no-arg constructor
get and set methods

The **Address** class is the same as shown in Listing 39.12. Complete the **AddressService** class as shown in Listing 42.3.

LISTING 42.3 AddressService.java

```

1 package chapter42;
2
3 import chapter39.Address;
4 import java.sql.*;
5 import javax.jws.WebMethod;
6 import javax.jws.WebService;
7
8 @WebService(name = "AddressService",           define service name
9   serviceName = "AddressWebService")
10 public class AddressService {
11   // statement1 for retrieving an address and statement2 for storing
12   private PreparedStatement statement1;         prepared statement
13
14   // statement2 for storing an address
15   private PreparedStatement statement2;         prepared statement
16
17   public AddressService() {
18     initializeJdbc();                           initialize database
19   }
20
21   @WebMethod(operationName = "getAddress")       define remote method
22   public Address getAddress(String firstName, String lastName) {      getAddress
23     try {
24       statement1.setString(1, firstName);
25       statement1.setString(2, lastName);
26       ResultSet resultSet = statement1.executeQuery();
27
28       if (resultSet.next()) {
29         Address address = new Address();
30         address.setFirstName(resultSet.getString("firstName"));
31         address.setLastName(resultSet.getString("lastName"));
32         address.setMi(resultSet.getString("mi"));
33         address.setTelephone(resultSet.getString("telephone"));
34         address.setFirstName(resultSet.getString("email"));
35         address.setCity(resultSet.getString("telephone"));
36         address.setState(resultSet.getString("state"));
37         address.setZip(resultSet.getString("zip"));
38         return address;
39       }
40       else
41         return null;
42     } catch (SQLException ex) {
43       ex.printStackTrace();
44     }
45
46     return null;
47   }
48
49   @WebMethod(operationName = "storeAddress")     define remote method
50   public void storeAddress(Address address) {      storeAddress
51     try {
52       statement2.setString(1, address.getLastName());
53       statement2.setString(2, address.getFirstName());
54       statement2.setString(3, address.getMi());
55       statement2.setString(4, address.getTelephone());

```

```

56         statement2.setString(5, address.getEmail());
57         statement2.setString(6, address.getStreet());
58         statement2.setString(7, address.getCity());
59         statement2.setString(8, address.getState());
60         statement2.setString(9, address.getZip());
61         statement2.executeUpdate();
62     } catch (SQLException ex) {
63         ex.printStackTrace();
64     }
65 }
66
67 /** Initialize database connection */
68 public void initializeJdbc() {
69     try {
70         Class.forName("com.mysql.jdbc.Driver");
71
72         // Connect to the sample database
73         Connection connection = DriverManager.getConnection(
74             "jdbc:mysql://localhost/javabook", "scott", "tiger");
75
76         statement1 = connection.prepareStatement(
77             "select * from Address where firstName = ? and lastName = ?");
78         statement2 = connection.prepareStatement(
79             "insert into Address " +
80             "(lastName, firstName, mi, telephone, email, street, city, "
81             + "state, zip) values (?, ?, ?, ?, ?, ?, ?, ?, ?)");
82     } catch (Exception ex) {
83         ex.printStackTrace();
84     }
85 }
86 }

```

initialize database

The new Web service is named **AddressWebService** (line 9) for the **AddressService** class.

When the service is deployed, the constructor (lines 17–19) of **AddressWebService** is invoked to initialize a database connection and create prepared **statement1** and **statement2** (lines 68–85).

The **findAddress** method searches the address in the **Address** table for the specified **firstName** and **lastName**. If found, the address information is returned in an **Address** object (lines 29–38). Otherwise, the method returns **null** (line 41).

The **storeAddress** method stores the address information from the **Address** object into the database (lines 52–61).



Note

Don't forget that you have to add the MySQL library to the **WebServiceProject** for this example to run.

database driver

Before you can use the service, deploy it. Right-click the **WebServiceProject** node in the Project to display a context menu and choose **Deploy**.

Now you are ready to develop a Web client that uses the **AddressWebService**. The client is a JSP program, as shown in Figure 42.11. The program has two functions. First, the user can enter the last name and first name and click the Search button to **search** for a record, as shown in Figure 42.12. Second, the user can enter the complete address information and click the **Store** button to store the information to the database, as shown in Figure 42.13.

Address Information - Windows Internet Explorer

http://localhost:8080/langweb/TestAddressWebService.jsp

Last Name * First Name * MI

Telephone Email

Street

City State Georgia-GA Zip

Search Store Reset

* required fields

FIGURE 42.11 The `TestAddressWebService` page allows the user to search and store addresses.

Address Information - Windows Internet Explorer

http://localhost:8080/langweb/TestAddressWebService.jsp?last

Last Name * Pan First Name * Peter MI

Telephone Email

Street

City State Georgia-GA Zip

Search Store Reset

* required fields

Peter Pan is not in the database

FIGURE 42.12 The `Search` button finds and displays an address.

Address Information - Windows Internet Explorer

http://localhost:8080/langweb/TestAddressWebService.jsp?last

Last Name * Smith First Name * John MI

Telephone (405) 342-2323 Email smith@yahoo.com

Street 100 Main Street

City Savannah State Georgia-GA Zip 31415

Search Store Reset

* required fields

John Smith has been added to the database

FIGURE 42.13 The `Store` button stores the address to the database.

For convenience, let us create this JSP program in the **liangweb** project. The **liangweb** project was created and used in Chapters 39–40. To use **AddressWebService**, first you have to create a Web service client as follows:

1. Right-click the **liangweb** project in the Project pane to display a context menu. Choose **New > Web Service Client** to display the New Web Service Client dialog box.
2. Check the **WSDL URL** radio button and enter `http://localhost:8080/WebServiceProject/AddressWebService?WSDL` in the WSDL URL field.
3. Click the *Finish* button.

Now a reference to **AddressWebService** is created. Note that this process also copies **Address.java** to the client project, as shown in Figure 42.14.

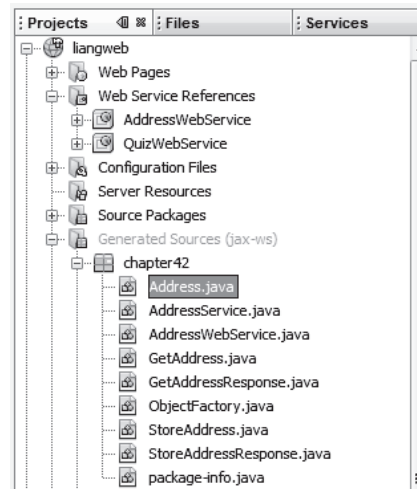


FIGURE 42.14 The **Address.java** is automatically copied to the Web service client reference package.

Create a JSP named **TestAddressWebService** in the **liangweb** project, as shown in Listing 42.4.

LISTING 42.4 TestAddressWebService.jsp

```

import Address
import
    AddressWebServices
import AddressServices

1 <!-- TestAddressWebService.jsp -->
2 <%@ page import = "chapter42.Address" %>
3 <%@ page import = "chapter42.AddressWebService" %>
4 <%@ page import = "chapter42.AddressService" %>
5 <jsp:useBean id = "addressId"
6   class = "chapter42.Address" scope = "session"></jsp:useBean>
7 <jsp:setProperty name = "addressId" property = "*" />
8
9 <html>
10 <head>
11   <title>Address Information</title>
12 </head>
13 <body>
14   <form method = "post" action = "TestAddressWebService.jsp">
15     Last Name <font color = "#FF0000">*</font>
16     <input type = "text" name = "lastName"

```

invoke the same page

```

17     <%if (addressId.getLastName() != null) {
18         out.print("value = \"" + addressId.getLastName() + "\"");}%>
19     size = "20" />&nbsp;
20
21 First Name <font color = "#FF0000">*</font>
22 <input type = "text" name = "firstName"
23     <%if (addressId.getFirstName() != null) {
24         out.print("value = \"" + addressId.getFirstName() + "\"");}%>
25     size = "20" />&nbsp;
26
27 MI
28 <input type = "text" name = "mi"
29     <%if (addressId.getMi() != null) {
30         out.print("value = \"" + addressId.getMi() + "\" "); } %>
31     size = "3" />&nbsp;
32
33 <p>Telephone
34 <input type = "text" name = "telephone"
35     <%if (addressId.getTelephone() != null) {
36         out.print("value = \"" + addressId.getTelephone() + "\" ");}%>
37     size = "20" />&nbsp;
38
39 Email
40 <input type = "text" name = "email"
41     <%if (addressId.getEmail() != null) {
42         out.print("value = \"" + addressId.getEmail() + "\" ");}%>
43     size = "28" />&nbsp;
44 </p>
45
46 <p>Street
47 <input type = "text" name = "street"
48     <%if (addressId.getStreet() != null) {
49         out.print("value = \"" + addressId.getStreet() + "\" ");}%>
50     size = "50" />&nbsp;
51 </p>
52
53 <p>City
54 <input type = "text" name = "city"
55     <%if (addressId.getCity() != null) {
56         out.print("value = \"" + addressId.getCity() + "\" ");}%>
57     size = "23" />&nbsp;
58
59 State
60 <select size = "1" name = "state">
61     <option value = "GA">Georgia-GA</option>
62     <option value = "OK">Oklahoma-OK</option>
63     <option value = "IN">Indiana-IN</option>
64 </select>&nbsp;
65
66 Zip
67 <input type = "text" name = "zip"
68     <%if (addressId.getZip() != null) {
69         out.print("value = \"" + addressId.getZip() + "\" "); } %>
70     size = "9" />&nbsp;
71 </p>
72
73 <p><input type = "submit" name = "Submit" value = "Search">
74     <input type = "submit" name = "Submit" value = "Store">
75     <input type = "reset" value = "Reset">
76 </p>

```

```

77     </form>
78     <p><font color = "#FF0000">* required fields</font></p>
79
80     <%
81     if (request.getParameter("Submit") != null) {
create Web service
get proxy object
82         AddressWebService addressWebService = new AddressWebService();
83         AddressService proxy = addressWebService.getAddressServicePort();
84
85         if (request.getParameter("Submit").equals("Store")) {
process Store button
invoke remote method
86             proxy.storeAddress(addressId);
87             out.println(addressId.getFirstName() + " " +
88                 addressId.getLastName() + " has been added to the database");
89         }
90     else if (request.getParameter("Submit").equals("Search")) {
process Search button
invoke remote method
91         Address address = proxy.getAddress(addressId.getFirstName(),
92             addressId.getLastName());
93         if (address == null)
94             out.print(addressId.getFirstName() + " " +
95                 addressId.getLastName() + " is not in the database");
96         else
97             addressId = address;
98     }
99 }
100 %>
101 </body>
102 </html>

```

Lines 2–4 import the classes for the JSP page. The **Address** class (line 2) was created in the **WebServiceProject** and was automatically copied to the **Tiangweb** project when a Web service reference for **AddressWebService** was created. A JavaBeans object for **Address** was created and associated with input parameters in lines 5–7.

The UI interface was laid in the form (lines 14–77). The action for the two buttons *Search* and *Store* invokes the same page `TestAddressWebService.jsp` (line 14).

When a button is clicked, a proxy object for **AddressWebService** is obtained (lines 82–83). For the *Store* button, the proxy object invokes the **storeAddress** method to add an address to the database (line 86). For the *Search* button, the proxy object invokes the **getAddress** method to return an address (lines 91–92). If no address is found for the specified first and last names, the returned address is **null** (line 93).

42.6 Web Service Session Tracking

HttpSession

§39.8.3, “Session Tracking Using the Servlet API,” introduced session tracking for servlets using the **javax.servlet.http.HttpSession** interface. You can use **HttpSession** to implement session tracking for Web services. To demonstrate this, consider an example that generates random True/False questions for the client and grades the answers on these questions for the client.

The Web client consists of two JSP pages: `DisplayQuiz.jsp` and `GradeQuiz.jsp`. The **DisplayQuiz** page invokes the remote method **generateQuiz()** to generate a quiz and invokes the remote method **getQuestion()** to display the questions, as shown in Figure 42.15. When you click the *Submit* button, the program invokes the remote method **gradeQuiz** to grade the answers. The result is displayed in the **GradeQuiz** page, as shown in Figure 42.16.

Why is session tracking needed for this project? Each time a client displays a quiz, it invokes the remote **generateQuiz** method to randomly reorder the quiz for the client. Each client gets a different quiz every time the **DisplayQuiz** page is refreshed. When the client submits the answer, the Web service checks the answer against the previously generated quiz. So the quiz has to be stored in the session.

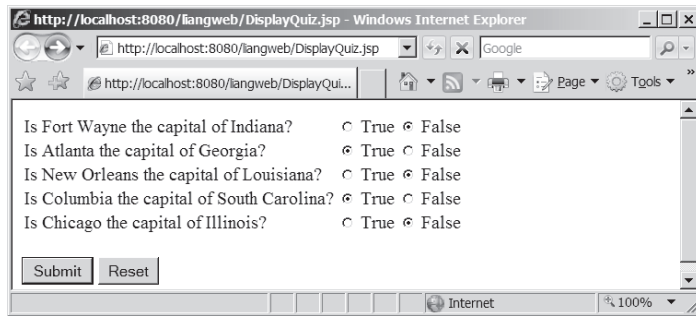


FIGURE 42.15 The *Submit* button submits the answers for grading.

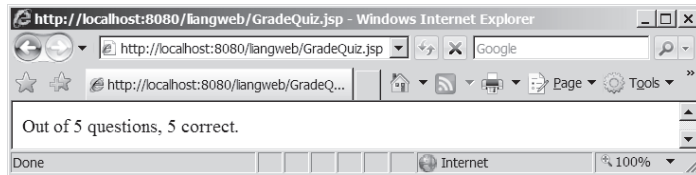


FIGURE 42.16 The answers are graded and displayed.

For convenience, let us create the Web service class named **QuizService** in the **WebServiceProject** in package **chapter42**. Listing 42.5 gives the program.

LISTING 42.5 QuizService.java

```

1 package chapter42;
2
3 import javax.annotation.Resource;
4 import javax.jws.WebMethod;
5 import javax.jws.WebService;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpSession;
8 import javax.xml.ws.WebServiceContext;
9 import javax.xml.ws.handler.MessageContext;
10 import java.util.List;
11 import java.util.ArrayList;
12
13 @WebService(name = "QuizService", serviceName = "QuizWebService")
14 public class QuizService {
15     private @Resource WebServiceContext webServiceContext;
16     private HttpSession session;
17     private ArrayList<Object[]> quiz = new ArrayList<Object[]>();
18
19     public QuizService() {
20         // Initialize questions and answers
21         quiz.add(new Object[]{
22             "Is Atlanta the capital of Georgia?", true});
23         quiz.add(new Object[]{
24             "Is Columbia the capital of South Carolina?", true});
25         quiz.add(new Object[]{
26             "Is Fort Wayne the capital of Indiana?", false});
27         quiz.add(new Object[]{
28             "Is New Orleans the capital of Louisiana?", false});
29         quiz.add(new Object[]{

```

define service name

@Resource annotation
HttpSession
quiz

initialize quiz

```

30         "Is Chicago the capital of Illinois?", false});
31     }
32
define remote method
generateQuiz
33     @WebMethod(operationName = "generateQuiz")
34     public void generateQuiz() {
35         MessageContext messageContext =
36             webServiceContext.getMessageContext();
get session
37         session = ((HttpServletRequest)messageContext.get(
38             MessageContext.SERVLET_REQUEST)).getSession();
39
clone quiz
40         java.util.ArrayList cloneQuiz =
41             (java.util.ArrayList)(quiz.clone());
42
shuffle
43         // Shuffle to generate a random quiz for a client
44         java.util.Collections.shuffle(cloneQuiz);
45
set attribute
46         // Store the quiz in the session
47         session.setAttribute("quiz", cloneQuiz);
48     }
49
define remote method
getQuestions
50     @WebMethod(operationName = "getQuestions")
51     public java.util.List<String> getQuestions() {
52         List<Object[]> quiz =
53             (List<Object[]>)(session.getAttribute("quiz"));
54
55         // Extract questions from quiz
56         List<String> questions = new ArrayList<String>();
57         for (int i = 0; i < quiz.size(); i++) {
58             questions.add((String)(quiz.get(i)[0]));
59         }
60
61         return questions; // Return questions in the quiz
62     }
63
define remote method
64     @WebMethod(operationName = "gradeQuiz")
65     public List<Boolean> gradeQuiz(List<Boolean> answers) {
66         // Retrieve the quiz from the session
67         List<Object[]> quiz =
68             (List<Object[]>)(session.getAttribute("quiz"));
69
70         List<Boolean> result = new ArrayList<Boolean>();
71         for (int i = 0; i < quiz.size(); i++)
check answers
72             result.add(quiz.get(i)[1] == answers.get(i));
73
74         return result;
75     }
76 }

```

The Web service class named **QuizService** contains three methods **generateQuiz**, **getQuestions**, and **gradeQuiz**. The new Web service is named **QuizWebService** (line 13).

The annotation **@Resource** generates supporting code for **WebServiceContext** (line 15). Line 16 declares an **HttpSession** for tracking a session.

Assume that five True/False questions are available from the service. The quiz is stored in an **ArrayList** (lines 21–30). Each element in the list is an array with two values. The first value is a string that describes the question and the second is a Boolean value indicating whether the answer should be true or false.

The **generateQuiz** method (lines 33–49) obtains a session from the **MessageContext** (lines 35–38). To ensure that each client gets an independent copy of the quiz, a clone of the

quiz is created in lines 40–41. A new quiz is generated by *randomly shuffling* the original quiz (line 44). The new quiz is stored in the session (line 47).

The `getQuestions` method (lines 50–62) returns questions in a list. Lines 52–53 obtain the quiz from the session. The questions are extracted from the quiz (lines 56–59) and are returned (line 61).

The `gradeQuiz` method (lines 64–75) checks the `answers` from the client with the answers in the quiz. Lines 67–68 obtain the quiz from the session. The client's answers are compared with the key, and the result of the grading is stored in a list. Each element in the list is a boolean value that indicates whether the answer is correct or incorrect (lines 70–72).

After creating and publishing the Web service, you can create `DisplayQuiz.jsp` and `GradeQuiz.jsp` in the `liangweb` project, which was created and used in Chapters 37–38. To use `QuizWebService`, you need to create a Web service client as follows:

1. Right-click the `liangweb` project in the Project pane to display a context menu. Choose **New > Web Service Client** to display the New Web Service Client dialog box.
2. Check the *WSDL URL* radio button and enter `http://localhost:8080/WebServiceProject-/QuizWebService?WSDL` in the WSDL URL field.
3. Enter `chapter42Quiz` in the Package field. (Note that you can create multiple Web client references in a project. However, each reference must be placed in a different package. Since the `AddressWebService` was placed in package `chapter42`, we place `QuizWebService` in package `chapter42Quiz`.)
4. Click *Finish* to create the reference for `QuizWebService`.

Now a reference to `QuizWebService` is created. You can create a proxy object to access the remote methods in `QuizService`. Listings 42.6 and 42.7 show `DisplayQuiz.jsp` and `GradeQuiz.jsp`.

LISTING 42.6 DisplayQuiz.jsp

```

1 <!-- DisplayQuiz.jsp -->
2 <%@ page import = "chapter42Quiz.QuizWebService" %>
3 <%@ page import = "chapter42Quiz.QuizService" %>
4 <jsp:useBean id = "quizWebService" scope = "session"
5   class = "chapter42Quiz.QuizWebService">
6 </jsp:useBean>
7
8 <html>
9 <body>
10   <%
11     QuizService proxy = quizWebService.getQuizServicePort();
12     proxy.generateQuiz();
13     java.util.List<String> questions =
14       (java.util.ArrayList<String>)(proxy.getQuestions());
15   %>
16   <form method = "post" action = "GradeQuiz.jsp">
17     <table>
18       <% for (int i = 0; i < questions.size(); i++) {%>
19         <tr>
20           <td>
21             <label><%= questions.get(i) %></label>
22           </td>
23           <td>
24             <input type = "radio" name = <%= "question" + i%>
25               value = "True" /> True
26           </td>

```

randomly shuffling

`getQuestions`

`gradeQuiz`

create Web service client

import `QuizWebService`
import `QuizServices`
create `QuizWebServices`

get proxy object
generate a quiz
get questions

display questions

```

27     <td>
28         <input type = "radio" name = <%= "question" + i%>
29             value = "False" /> False
30     </td>
31 </tr>
32 <%=}%>
33 </table>
34 <p><input type = "submit" name = "Submit" value = "Submit">
35     <input type = "reset" value = "Reset">
36 </p>
37 </form>
38 </body>
39 </html>

```

This page generates a quiz by invoking the remote `generateQuiz()` method (line 12). The quiz is obtained by invoking `getQuestions()` in lines 13–14. The questions are displayed in a table with radio buttons (lines 17–33).

Clicking *Submit* invokes `GradeQuiz.jsp`.

LISTING 42.7 GradeQuiz.jsp

```

import QuizWebService
import QuizServices
create QuizWebServices

1 <!-- GradeQuiz.jsp -->
2 <%@ page import = "chapter42Quiz.QuizWebService" %>
3 <%@ page import = "chapter42Quiz.QuizService" %>
4 <jsp:useBean id = "quizWebService" scope = "session"
5     class = "chapter42Quiz.QuizWebService">
6 </jsp:useBean>
7
8 <html>
9 <body>
10 <%
11 QuizService proxy = quizWebService.getQuizServicePort();
12 java.util.List<String> quiz = proxy.getQuestions();
13
14 // Get the answer from the DisplayQuiz page
15 java.util.List<Boolean> answers = new java.util.ArrayList<Boolean>();
16 for (int i = 0; i < quiz.size(); i++) {
17     String trueOrFalse = request.getParameter("question" + i);
18     if (trueOrFalse.equals("True"))
19         answers.add(true); // Answered true
20     else if (trueOrFalse.equals("False"))
21         answers.add(false); // Answered false
22 }
23
24 // Grade answers
25 java.util.List<Boolean> result = proxy.gradeQuiz(answers);
26
27 // Find the correct count
28 int correctCount = 0;
29 for (int i = 0; i < result.size(); i++) {
30     if (result.get(i))
31         correctCount++;
32 }
33 %>
34
35 Out of <%= result.size() %> questions, <%= correctCount %> correct.
36 </body>
37 </html>

```

This page collects the answers passed from the HTML form from the **DisplayQuiz** page (lines 15–21), invokes the **gradeQuiz** method to grade the quiz (line 25), finds the correct count (lines 28–31), and displays the result (line 35).



Note

You need to answer all five questions before clicking the Submit button. A runtime error will occur if a radio button is not checked. You can fix this problem in Exercise 42.5.

KEY TERMS

@WebService	42–3	publishing a Web service	42–2
@WebMethod	42–3	Web service	42–2
consuming a Web service	42–2	Web service reference	42–6
proxy object	42–2	WSDL	42–7

CHAPTER SUMMARY

-
1. Web services enable a Java program on one system to invoke a method in an object on another system.
 2. Web services are platform and language independent. You can develop and use Web services using any language.
 3. Web services run on the Web using HTTP. SOAP is a popular protocol for implementing Web services.
 4. The server needs to make the service available to the client, known as *publishing a Web service*. Using a Web service from a client is known as *consuming a Web service*.
 5. A client interacts with a Web service through a *proxy object*. The proxy object facilitates the communication between the client and the Web service.
 6. You need to use Java annotation **@WebService** to annotate a Web service and use annotation **@WebMethod** to annotate a remote method.
 7. A Web service class may have an unlimited number of remote methods.
 8. After a Web service is published, you can write a client program to use it. You have to first create a Web service reference. From the reference, you create a proxy object for facilitating communication between a server and a client.
 9. WSDL stands for *Web Service Description Language*. A *.wsdl* file is an XML file that describes the available Web service to the client—i.e., the remote methods, their parameters and return value types, and so on.
 10. The message between the client and server is described in XML. A SOAP request describes the information that is sent to the Web service and a SOAP response describes the information that is received from the Web service.
 11. The objects passed between client and Web service are serialized in XML. Not all object types are supported by SOAP.
 12. You can track sessions in Web services using the **HttpSession** in the same way as in servlets.

REVIEW QUESTIONS

Section 42.1

- 42.1** What is a Web service?
- 42.2** Can you invoke a Web service from a language other than Java?
- 42.3** Do Web services support callback? That is, can a Web service call a method from a client's program?
- 42.4** What is SOAP? What is it to publish a Web service? What is it to consume a Web service? What is the role of a proxy object?

Sections 42.2–42.6

- 42.5** What is the annotation to specify a Web service? What is the annotation to specify a Web method?
- 42.6** How do you deploy a Web service in NetBeans?
- 42.7** Can you test a Web service from a client?
- 42.8** How do you create a Web service reference for a client?
- 42.9** What is WSDL? What is SOAP? What is a SOAP request? What is a SOAP response?
- 42.10** Can you pass primitive type arguments to a remote method? Can you pass any object type to a remote method? Can you pass an argument of a custom type to a remote method?
- 42.11** How do you obtain an `HttpSession` object for tracking a Web session?
- 42.12** Can you create two Web service references in one package in the same project in NetBeans?
- 42.13** What happens if you don't clone the quiz in lines 40–41 in Listing 42.5, `QuizService.java`?

PROGRAMMING EXERCISES

- 42.1*** (*Getting a score from a database table*) Suppose that the scores are stored in the `Scores` table. The table was created as follows:

```
create table Scores (name varchar(20),
                    score number, permission boolean);

insert into Scores values ('John', 90.5, 1);
insert into Scores values ('Michael', 100, 1);
insert into Scores values ('Michelle', 100, 0);
```

Revise the `findScore` method in Listing 42.1, `ScoreService.java`, to obtain a score for the specified name. Note that your program does not need the `permission` column; ignore it. The next exercise will need the `permission` column.

- 42.2*** (*Permission to find scores*) Revise the preceding exercise so that the `findScore` method returns `-1` if permission is `false`. Add another method named `getPermission(String name)` that returns `1`, `0`, or `-1`. The method returns `1` if the student is in the `Scores` table and permission is `true`, `0` if the student is in the `Scores` table and permission is `false`, and `-1` if the student is not in the `Scores` table.

- 42.3*** (*Computing loan*) You can compute a loan payment for a loan with the specified amount, number of years, and annual interest rate. Write a Web service with two remote methods for computing monthly payment and total payment. Write a client program that prompts the user to enter loan amount, number of years, and annual interest rate.
- 42.4*** (*Web service visit count*) Write a Web service with a method named `getCount()` that returns the number of the times this method has been invoked from a client. Use a session to store the `count` variable.
- 42.5*** (*Quiz*) The user needs to answer all five questions before clicking the *Submit* button in the Quiz application in §42.6, Web Service Session Tracking. A runtime error will occur if a radio button is not checked. Fix this problem.

