CHAPTER

41

Web Services

Objectives

- To describe what a Web service is (§41.1).
- To create a Web service class (§41.2).
- To publish and test a Web service (§41.3).
- To create a Web service client reference (§41.4).
- To explain the role of WSDL (§41.4).
- To pass arguments of object type in a Web service (§41.5).
- To discover how a client communicates with a Web service (§41.5).
- To describe what SOAP requests and SOAP responses are (§41.5).
- To track a session in Web services (§41.6).

41.1 Introduction

Key Point

Web services is about sharing objects on the Internet.

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 41.1.

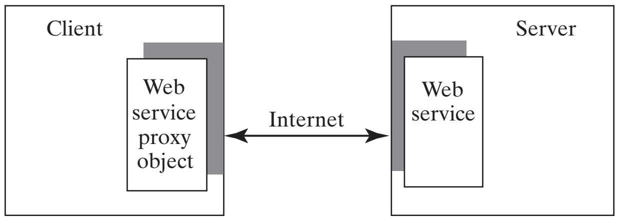


FIGURE 41.1 A proxy object serves as a facilitator between a client and a Web service.

Check Point

41.1.1 What is a Web service?

41.1.2 Can you invoke a Web service from a language other than Java?

41.1.3 Do Web services support callback? That is, can a Web service call a method from a client's program?

41.1.4 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?

41.2 Creating Web Services

Key Point

An IDE such as NetBeans is an effective tool for developing and deploying Web services.

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

Note

Apache Tomcat Server does not work well with Web services. To develop and deploy Web services using NetBeans, you need to install GlassFish. For information on how to install GlassFish on NetBeans.

We now create 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 `ScoreService` 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. In the New Project dialog box, choose **`Java Web`** in the Categories pane and choose **`Web Application`** in the Projects pane. Click *Next* to display the New Web Application dialog box.

2. Enter WebServiceProject as the project name, specify the location where you want the project to be stored, and click Next to display the Server and Setting dialog.
3. Select **GlassFish 4** as the server and **Java EE 7 Web** 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. (If you don't see **Web Service**, click **New**, **Other** to display the New File dialog box to choose Web Service in this dialog box.)
2. Enter ScoreService in the Web Service Name field and enter chapter41 in the Package field. Click *Finis*h to create ScoreService.
3. Complete the source code as shown in Listing 41.1.

LISTING 41.1 ScoreService.java

```
 1  package chapter41;
 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  }
```

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)

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

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

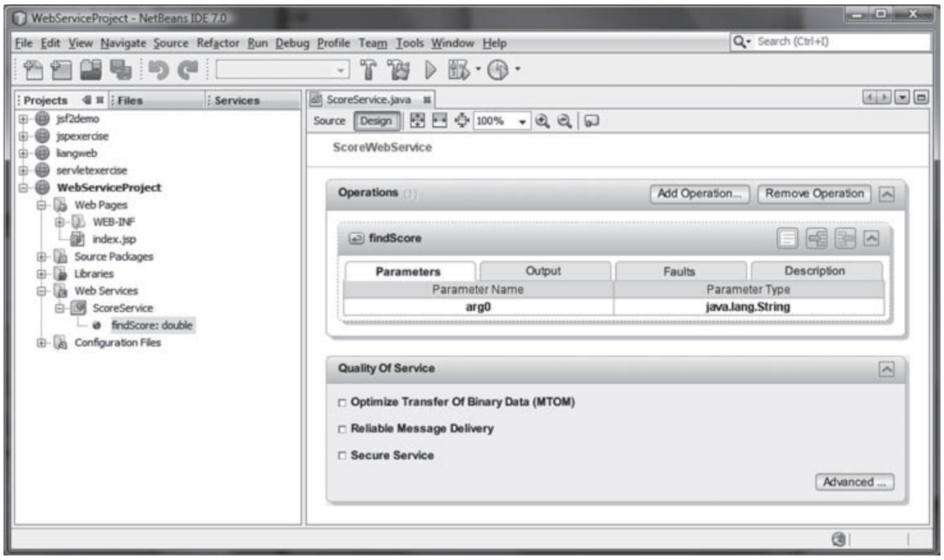
The annotation (line 19)

```
@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 41.2.



Parameter Name	Parameter Type
arg0	java.lang.String

Figure 41.2 The services can also be created from the Design pane.

41.3 Deploying and Testing Web Services

Key Point

Deploying a Web service is to make it available on the Internet for other programs to use.

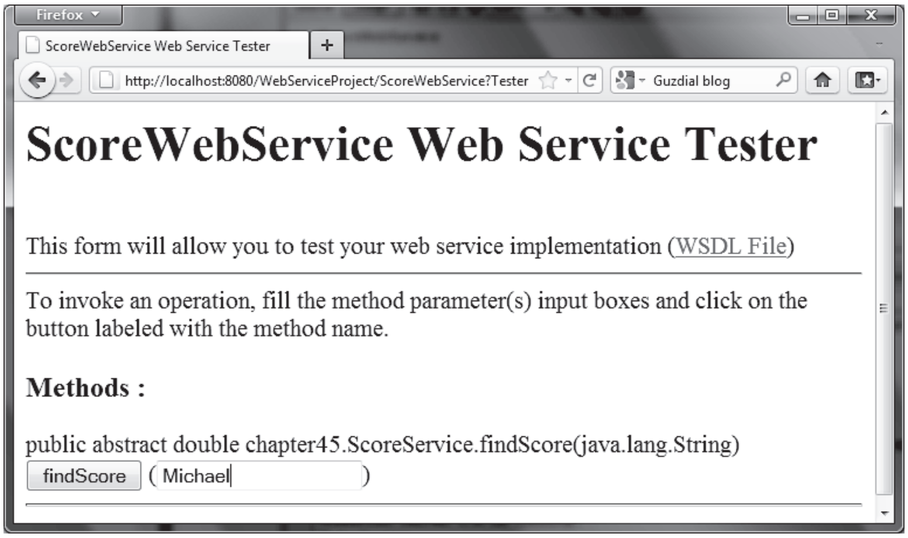
After a Web service is created, you need to deploy it for clients to use. Deploying Web services is also known as *publishing Web services*. To deploy it, right-click the `WebServiceProject` in the Project to display a context menu and choose **`Deploy`**. This command will first undeploy the service 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 41.3.

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

Note `ScoreWebService` is the name you specified in line 7 in Listing 41.1. 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 the *findScore* button. You will see that the method returns `100.0`, as shown in Figure 41.4.


Firefox
ScoreWebService Web Service Tester
http://localhost:8080/WebServiceProject/ScoreWebService?Tester
Guzdial blog

ScoreWebService Web Service Tester

This form will allow you to test your web service implementation (WSDL File)

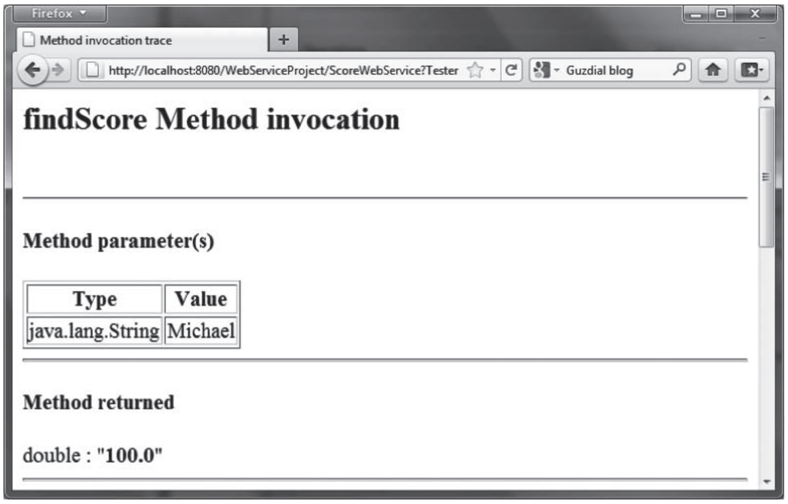
To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

public abstract double chapter45.ScoreService.findScore(java.lang.String)

findScore (Michael)

FIGURE 41.3 The test page enables you to test Web services.


Firefox
Method invocation trace
http://localhost:8080/WebServiceProject/ScoreWebService?Tester
Guzdial blog

findScore Method invocation

Method parameter(s)

Type	Value
java.lang.String	Michael

Method returned

double : "100.0"

FIGURE 41.4 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`**.

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.

41.4 Consuming Web Services

Key Point

Consuming a Web service is for a client to use a Web service.

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

We will use NetBeans to create a Web service client. Our client is a GUI application. The application simply lets the user enter a name and displays the score, as shown in Figure 41.5.

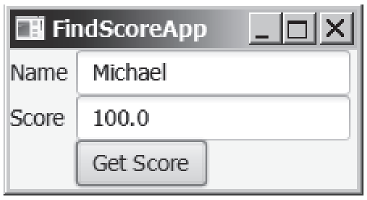


FIGURE 41.5 The client uses the Web service to find scores.

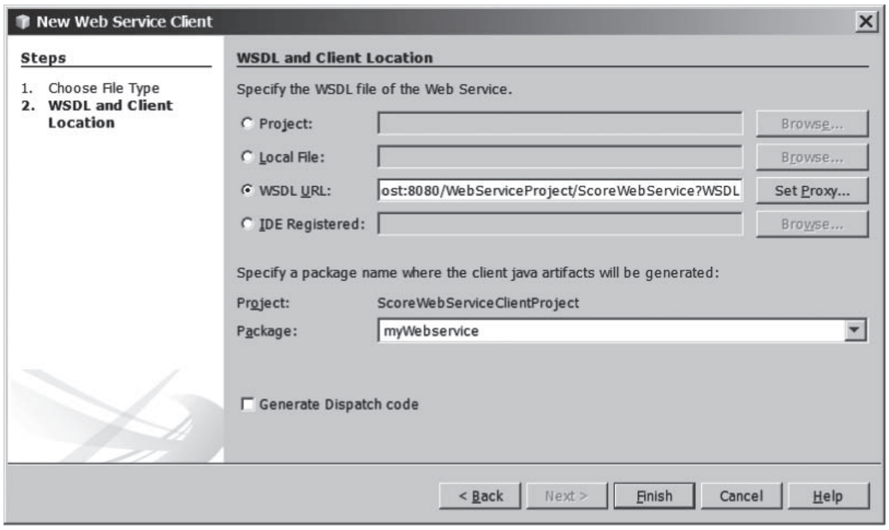
Let us create a project for the client. The project named `ScoreWebServiceClient-Project` can be created as follows:

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

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 41.6.
2. Check the *WSDL URL* radio button and enter http://localhost:8080/WebServiceProject/ScoreWebService?WSDL in the WSDL URL field.
3. Enter `myWebservice` in the package name field. 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 41.7. These files will be used by the proxy object to interact with the Web service.



Specify the WSDL file of the Web Service.

- Project:
- Local File:
- (selected) WSDL URL: ost:8080/WebServiceProject/ScoreWebService?WSDL
- IDE Registered:

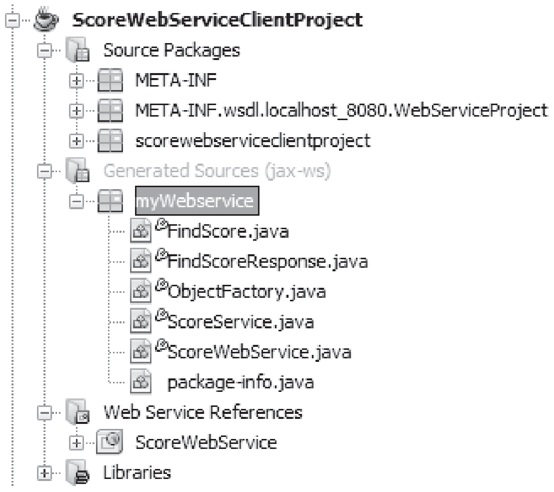
Specify a package name where the client java artifacts will be generated:

Project: ScoreWebServiceClientProject

Package: myWebservice

☐ Generate Dispatch code

FIGURE 41.6 The New Web Service Client dialog box creates a Web service reference.



```
ScoreWebServiceClientProject
  Source Packages
    META-INF
    META-INF.wsdl.localhost_8080.WebServiceProject
    scorewebserviceclientproject
  Generated Sources (jax-ws)
    myWebservice
      FindScore.java
      FindScoreResponse.java
      ObjectFactory.java
      ScoreService.java
      ScoreWebService.java
      package-info.java
  Web Service References
    ScoreWebService
  Libraries
```

FIGURE 41.7 You can see the automatically generated boilerplate code for Web services in the Generated Sources folder in the client's project.

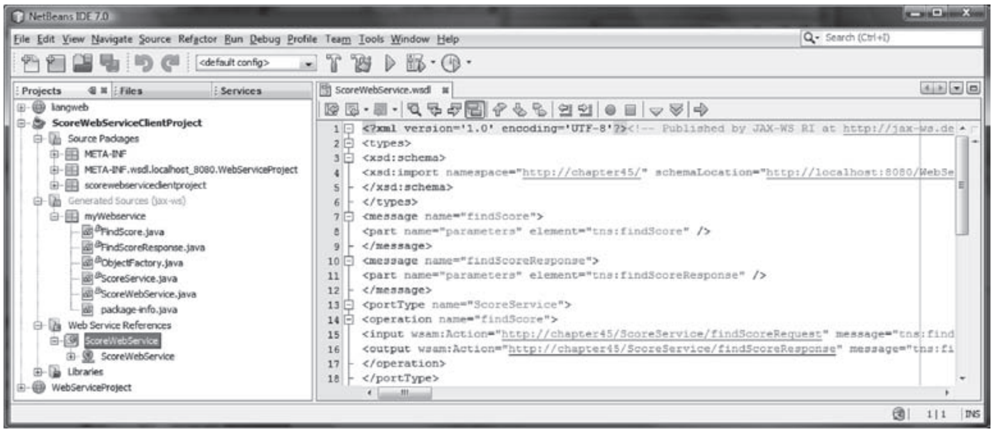
Note

When you created a Web service reference, you entered a WSDL URL, as shown in Figure 41.6. This creates a .wsdl file. In this case, it is named ScoreWebService.wsdl under the Web Service References folder, as shown in Figure 41.8. 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.

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`**.

Now you are ready to create a client for the Web service. Right-click the `ScoreWebService-ClientProject` node in the Project pane to display a context menu, and choose **`New, Class`** to create a Java client named `FindScoreApp` in package **`chapter41`**, as shown in Listing 41.2.



```
<?xml version='1.0' encoding='UTF-8'?><!-- Published by JAX-WS RI at http://jax-ws.de
<types>
<xsd:schema>
<xsd:import namespace="http://chapter45/" schemaLocation="http://localhost:8080/WebSe
</xsd:schema>
</types>
<message name="findScore">
<part name="parameters" element="tns:findScore" />
</message>
<message name="findScoreResponse">
<part name="parameters" element="tns:findScoreResponse" />
</message>
<portType name="ScoreService">
<operation name="findScore">
<input wsam:Action="http://chapter45/ScoreService/findScoreRequest" message="tns:find
<output wsam:Action="http://chapter45/ScoreService/findScoreResponse" message="tns:fi
</operation>
</portType>
```

FIGURE 41.8 The .wsdl file describes Web services to clients.

LISTING 41.2 FindScoreApp.java

```java
package  chapter41;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
import myWebservice.ScoreWebService;
import myWebservice.ScoreService;

public class FindScoreApp extends Application {
  // Declare a service object and a proxy object
  private ScoreWebService scoreWebService = new ScoreWebService();
  private ScoreService proxy
    = scoreWebService.getScoreServicePort();

  private Button btGetScore = new Button("Get Score");
  private TextField tfName = new TextField();
  private TextField tfScore = new TextField();

  public void  start(Stage primaryStage) {
    GridPane gridPane = new GridPane();
    gridPane.setHgap(5);
    gridPane.add(new Label("Name"), 0, 0);
    gridPane.add(new Label("Score"), 0, 1);
    gridPane.add(tfName, 1, 0);
    gridPane.add(tfScore, 1, 1);
    gridPane.add(btGetScore, 1, 2);

    // Create a scene and place the pane in the stage
    Scene scene = new Scene(gridPane, 250, 250);
    primaryStage.setTitle("FindScoreApp"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage

38      btGetScore.setOnAction(e -> getScore());
39    }
40
41    private void getScore() {
42      try {
43        // Get student score
44        double score = proxy.findScore(tfName.getText().trim());
45
46        // Display the result
47        if  (score < 0)
48          tfScore.setText("Not found");
49        else
50          tfScore.setText(new Double(score).toString());
51        }
52        catch(Exception ex) {
53          ex.printStackTrace();
54      }
55    }
56  }
```

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).

41.5 Passing and Returning Arguments

The Simple Object Access Protocol (SOAP) can be used to send and return values to and from a Web service.

Key Point

In the preceding example, a Web service client 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 the server.

SOAP is based on XML. The message between the client and the server is described in XML. Figure 41.9 shows the SOAP request and SOAP response for the `findScore` method.

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 41.9, the XML text

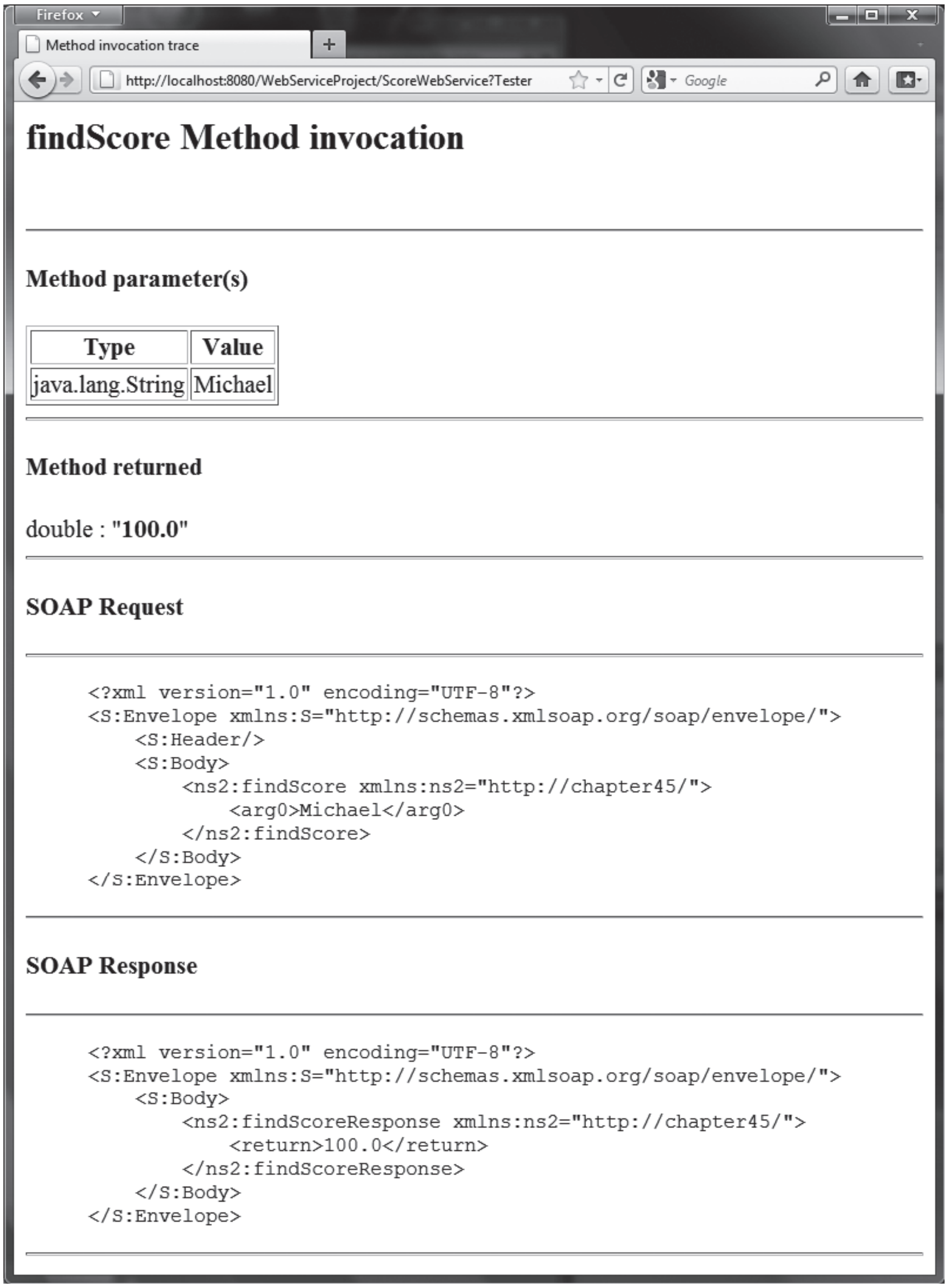
```
<ns1:findScore>
   <arg0>Michael</arg0>
</ns1:findScore>
```

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 41.9, the XML text

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

specifies that the method returns `100.0`.


Firefox
Method invocation trace
http://localhost:8080/WebServiceProject/ScoreWebService?Tester
Google

findScore Method invocation

Method parameter(s)

Type	Value
java.lang.String	Michael

Method returned

double : "100.0"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:findScore xmlns:ns2="http://chapter45/">
            <arg0>Michael</arg0>
        </ns2:findScore>
    </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:findScoreResponse xmlns:ns2="http://chapter45/">
            <return>100.0</return>
        </ns2:findScoreResponse>
    </S:Body>
</S:Envelope>
```

FIGURE 41.9 The client request and server response are described in XML.

The proxy object receives the SOAP response from the Web service and parses it. This process is illustrated in Figure 41.10.

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

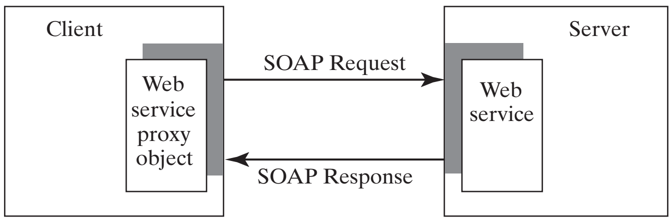


FIGURE 41.10 A proxy object sends SOAP requests and receives SOAP responses.

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 41.3 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 42.12, 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 `chapter41`. 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 `chapter37`. Click *Finish* to create the class.

The `Address` class is the same as shown in Listing 37.12. Complete the `AddressService` class as shown in Listing 41.3.

LISTING 41.3 AddressService.java

```
package chapter41;

import chapter37.Address;
import java.sql.*;
import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService(name = "AddressService",
  serviceName = "AddressWebService")
```

```java
public class AddressService {
  // statement1 for retrieving an address and statement2 for storing
  private PreparedStatement statement1;

  // statement2 for storing an address
  private PreparedStatement statement2;

  public AddressService() {
    initializeJdbc();
  }

  @WebMethod(operationName = "getAddress")
  public Address getAddress(String firstName, String lastName) {
    try {
      statement1.setString(1, firstName);
      statement1.setString(2, lastName);
      ResultSet resultSet = statement1.executeQuery();

      if (resultSet.next()) {
        Address address = new Address();
        address.setFirstName(resultSet.getString("firstName"));
        address.setLastName(resultSet.getString("lastName"));
        address.setMi(resultSet.getString("mi"));
        address.setTelephone(resultSet.getString("telephone"));
        address.setFirstName(resultSet.getString("email"));
        address.setCity(resultSet.getString("telephone"));
        address.setState(resultSet.getString("state"));
        address.setZip(resultSet.getString("zip"));
        return address;
      }
      else
        return null;
    } catch (SQLException ex) {
      ex.printStackTrace();
    }

    return null;
  }

  @WebMethod(operationName = "storeAddress")
  public void storeAddress(Address address) {
    try {
      statement2.setString(1, address.getLastName());
      statement2.setString(2, address.getFirstName());
      statement2.setString(3, address.getMi());
      statement2.setString(4, address.getTelephone());
      statement2.setString(5, address.getEmail());
      statement2.setString(6, address.getStreet());
      statement2.setString(7, address.getCity());
      statement2.setString(8, address.getState());
      statement2.setString(9, address.getZip());
      statement2.executeUpdate();
    } catch (SQLException ex) {
    ex.printStackTrace();
    }
  }

  /** Initialize database connection */
  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   }
```

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.

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

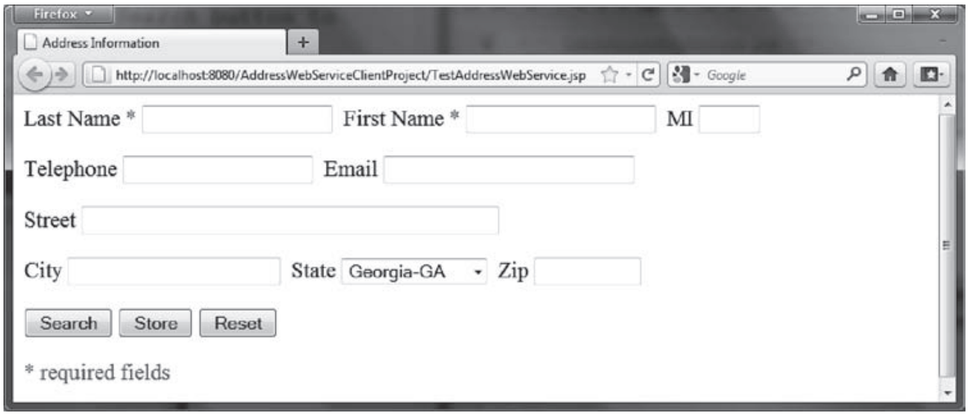
Now you are ready to develop a Web client that uses the `AddressWebService`. The client is a JSP program, as shown in Figure 41.11. The program has two functions. First, the user can enter the last name and first name and click the *Search* button to search for a record, as shown in Figure 41.12. Second, the user can enter the complete address information and click the *Store* button to store the information to the database, as shown in Figure 41.13.

Let us create a project for the client. The project named `AddressWebServiceClient-Project` can be created as follows:

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

2. Enter `AddressWebServiceClientProject` as the project name, specify the location where you want the project to be stored, and uncheck the *Set as Main Project* check box. Click *Next* to display the Server and Settings dialog box.

3. Choose **`GlassFish Server 4`** in the Server field, and **`Java EE 7 Web`** as in the Java EE Version field, and click *Finish* to create the project.


Firefox
Address Information
http://localhost:8080/AddressWebServiceClientProject/TestAddressWebService.jsp
Google

Last Name * First Name * MI

Telephone Email

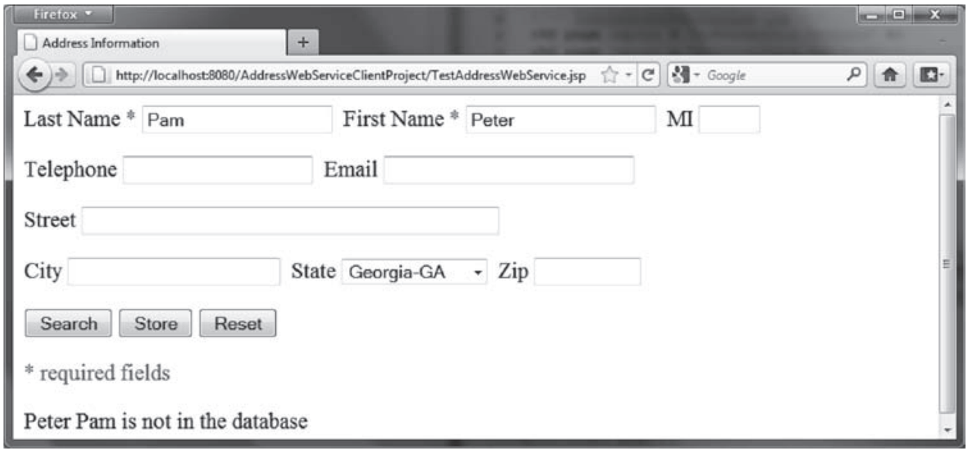
Street

City State Georgia-GA Zip

Search Store Reset

* required fields

FIGURE 41.11 The TestAddressWebService page allows the user to search and store addresses.


Firefox
Address Information
http://localhost:8080/AddressWebServiceClientProject/TestAddressWebService.jsp
Google

Last Name * Pam First Name * Peter MI

Telephone Email

Street

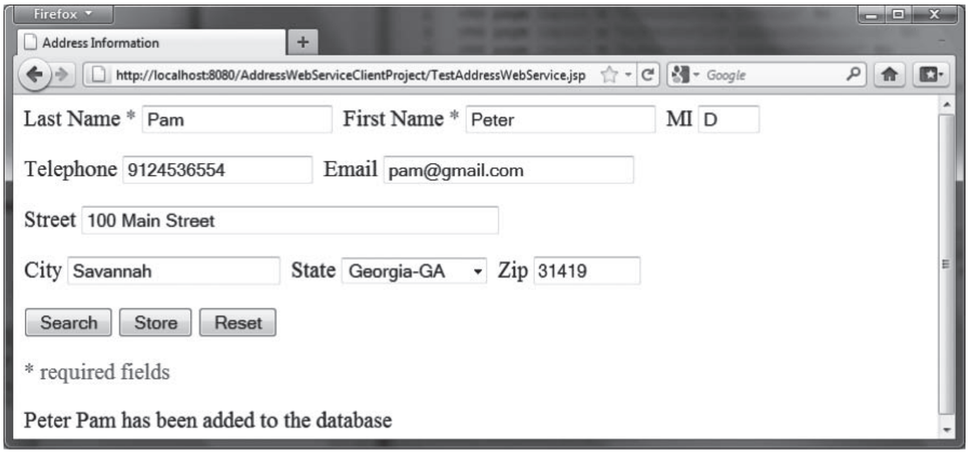
City State Georgia-GA Zip

Search Store Reset

* required fields

Peter Pam is not in the database

FIGURE 41.12 The *Search* button finds and displays an address.


Firefox
Address Information
http://localhost:8080/AddressWebServiceClientProject/TestAddressWebService.jsp
Google

Last Name * Pam First Name * Peter MI D

Telephone 9124536554 Email pam@gmail.com

Street 100 Main Street

City Savannah State Georgia-GA Zip 31419

Search Store Reset

* required fields

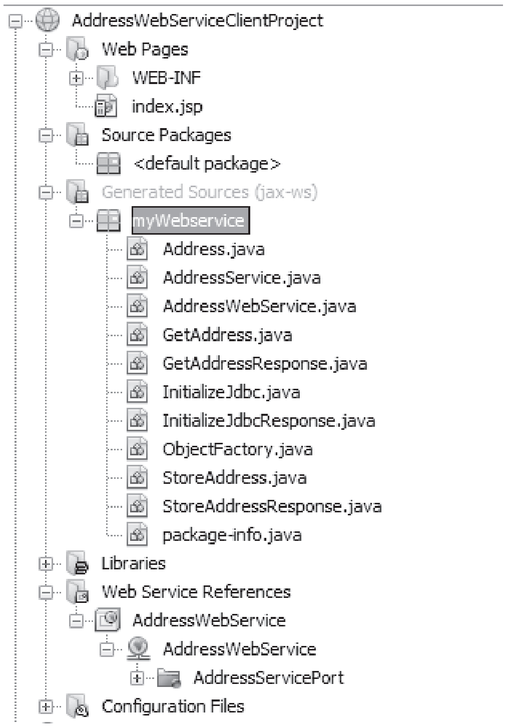
Peter Pam has been added to the database

FIGURE 41.13 The *Store* button stores the address to the database.

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 `AddressWebServiceClientProject` node 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. Enter `myWebservice` in the package name field and choose **JAX-WS** as the JAX version. Click *Finish* to generate the Web service reference.

Now a reference to `AddressWebService` is created. Note this process also copies `Address.java` to the client project, as shown in Figure 41.14.



```
AddressWebServiceClientProject
    Web Pages
        WEB-INF
        index.jsp
    Source Packages
        <default package>
    Generated Sources (jax-ws)
        myWebservice
            Address.java
            AddressService.java
            AddressWebService.java
            GetAddress.java
            GetAddressResponse.java
            InitializeJdbc.java
            InitializeJdbcResponse.java
            ObjectFactory.java
            StoreAddress.java
            StoreAddressResponse.java
            package-info.java
    Libraries
    Web Service References
        AddressWebService
            AddressWebService
                AddressServicePort
    Configuration Files
```

FIGURE 41.14 Address.java is automatically copied to the Web service client reference package.

Create a JSP named `TestAddressWebService` in the `AddressWebServiceClientProject` project, as shown in Listing 41.4.

LISTING 41.4 `TestAddressWebService.jsp`

```
<!-- TestAddressWebService.jsp -->
<%@ page import = "myWebservice.Address" %>
<%@ page import = "myWebservice.AddressWebService" %>
<%@ page import = "myWebservice.AddressService" %>
<jsp:useBean id = "addressId"
  class = "myWebservice.Address" scope = "session"></jsp:useBean>
<jsp:setProperty name = "addressId" property = "*" />

```

```
<html>
<head>
  <title>Address Information</title>
</head>
<body>
  <form method = "post" action = "TestAddressWebService.jsp">
  Last Name <font color = "#FF0000">*</font>
  <input type = "text" name = "lastName"
    <%if (addressId.getLastName() != null) {
      out.print("value = \"" + addressId.getLastName() + "\"");}%>
    size = "20" />&nbsp;

  First Name <font color = "#FF0000">*</font>
  <input type = "text" name = "firstName"
    <%if (addressId.getFirstName() != null) {
      out.print("value = \"" + addressId.getFirstName() + "\"");}%>
    size = "20" />&nbsp;

  MI
  <input type = "text" name = "mi"
    <%if (addressId.getMi() != null) {
      out.print("value = \"" + addressId.getMi() + "\" "); } %>
    size = "3" />&nbsp;

  <p>Telephone
  <input type = "text" name = "telephone"
    <%if (addressId.getTelephone() != null) {
      out.print("value = \"" + addressId.getTelephone() + "\" ");}%>
    size = "20" />&nbsp;

   Email
   <input type = "text" name = "email"
     <%if (addressId.getEmail() != null) {
       out.print("value = \"" + addressId.getEmail() + "\" ");}%>
     size = "28" />&nbsp;
   </p>

   <p>Street
   <input type = "text" name = "street"
     <%if (addressId.getStreet() != null) {
       out.print("value = \"" + addressId.getStreet() + "\" ");}%>
     size = "50" />&nbsp;
   </p>

   <p>City
   <input type = "text" name = "city"
     <%if (addressId.getCity() != null) {
       out.print("value = \"" + addressId.getCity() + "\" ");}%>
   size = "23" />&nbsp;

  State
  <select size = "1" name = "state">
    <option value = "GA">Georgia-GA</option>
    <option value = "OK">Oklahoma-OK</option>
    <option value = "IN">Indiana-IN</option>
  </select>&nbsp;

  Zip
  <input type = "text" name = "zip"
    <%if (addressId.getZip() != null) {
      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) {
 82         AddressWebService addressWebService = new AddressWebService();
 83         AddressService proxy = addressWebService.getAddressServicePort();
 84 
 85         if (request.getParameter("Submit").equals("Store")) {
 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")) {
 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 `AddressWebServiceClientProject` 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).

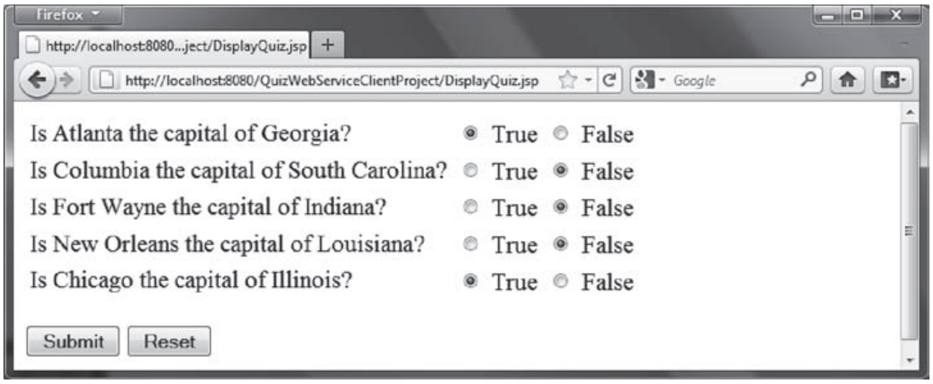
41.6 Web Service Session Tracking

You can use the `HttpSession` *interface to session tracking for Web.*

Key Point

Section 37.8.3, 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.

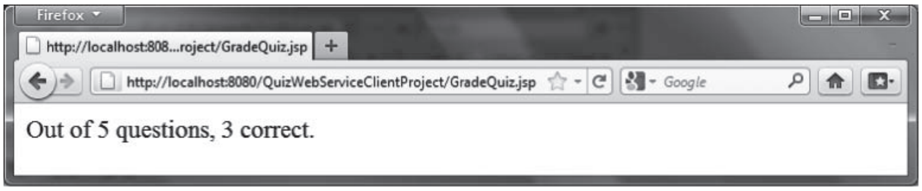
The Web client consists of two JSP pages: DisplayQuiz.jsp and GradeQuiz.jsp. The `DisplayQuiz` page invokes the service method `getQuestion()` to display the questions, as shown in Figure 41.15. When you click the *Submit* button, the program invokes the service


Firefox
http://localhost:8080...ject/DisplayQuiz.jsp
http://localhost:8080/QuizWebServiceClientProject/DisplayQuiz.jsp
Google

Is Atlanta the capital of Georgia? [✓ True] False
Is Columbia the capital of South Carolina? True [✓ False]
Is Fort Wayne the capital of Indiana? True [✓ False]
Is New Orleans the capital of Louisiana? True [✓ False]
Is Chicago the capital of Illinois? [✓ True] False

Submit Reset

FIGURE 41.15 The *Submit* button submits the answers for grading.


Firefox
http://localhost:808...roject/GradeQuiz.jsp
http://localhost:8080/QuizWebServiceClientProject/GradeQuiz.jsp
Google

Out of 5 questions, 3 correct.

FIGURE 41.16 The answers are graded and displayed.

method `gradeQuiz` to grade the answers. The result is displayed in the `GradeQuiz` page, as shown in Figure 41.16.

Why is session tracking needed for this project? Each time a client displays a quiz, it creates a 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.

For convenience, let us create the Web service class named `QuizService` in the `WebServiceProject` in package `chapter41`. Listing 41.5 gives the program.

LISTING 41.5 QuizService.java

```java
package chapter41;

import javax.jws.WebMethod;
import javax.jws.WebService;
import java.util.List;
import java.util.ArrayList;
import com.sun.xml.ws.developer.servlet.HttpSessionScope;

@HttpSessionScope
@WebService(name = "QuizService", serviceName = "QuizWebService")
public class QuizService {
  private ArrayList<Object[]> quiz = new ArrayList<Object[]>();

  public QuizService() {
    // Initialize questions and answers
    quiz.add(new Object[]{
      "Is Atlanta the capital of Georgia?", true});
    quiz.add(new Object[]{
      "Is Columbia the capital of South Carolina?", true});
    quiz.add(new Object[]{
      "Is Fort Wayne the capital of Indiana?", false});
22        quiz.add(new Object[]{
23          "Is New Orleans the capital of Louisiana?", false});
24        quiz.add(new Object[]{
25          "Is Chicago the capital of Illinois?", false});
26
27        // Shuffle to generate a random quiz for a client
28        java.util.Collections.shuffle(quiz);
29      }
30
31      @WebMethod(operationName = "getQuestions")
32      public java.util.List<String> getQuestions() {
33
34        // Extract questions from quiz
35        List<String> questions = new ArrayList<String>();
36        for  (int i = 0; i < quiz.size(); i++) {
37          questions.add((String)(quiz.get(i)[0]));
38        }
39
40        return questions; // Return questions in the quiz
41      }
42
43      @WebMethod(operationName = "gradeQuiz")
44      public List<Boolean> gradeQuiz(List<Boolean> answers) {
45        List<Boolean> result = new ArrayList<Boolean>();
46        for (int i = 0; i < quiz.size(); i++)
47          result.add(quiz.get(i)[1] == answers.get(i));
48
49        return result;
50      }
51    }
```

The Web service class named `QuizService` contains two methods `getQuestions` and `gradeQuiz`. The new Web service is named `QuizWebService` (line 10).

The annotation `@HttpSessionScope` (line 9) is new in JAX-WS 2.2, which enables the Web service automatically maintains a separate instance for each client session. To use this annotation, you have add JAX-WS 2.2 into your project's library. This can be done by clicking the **`Library`** node in the project and select **`Add Library`**.

Assume five True/False questions are available from the service. The quiz is stored in an `ArrayList` (lines 16–25).

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.

A new quiz is generated in the constructor and the quiz is shuffled using the shuffle method in the Collections class (line 28).

The `getQuestions` method (lines 31–40) returns questions in a list. The questions are extracted from the quiz (lines 34–37) and are returned (line 39).

The `gradeQuiz` method (lines 42–49) checks the `answers` from the client with the answers in the quiz. 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 44–46).

After creating and publishing the Web service, let us create a project for the client. The project named `QuizWebServiceClientProject` can be created as follows:

1. Choose **`File, New Project`** to display the New Web Application dialog box.
2. In the New Web Application dialog box, choose **`Java Web`** in the Categories pane and choose **`Web Application`** in the Projects pane. Click *Next* to display the Name and Location dialog box.

3. Enter `QuizWebServiceClientProject` as the project name, specify the location where you want the project to be stored, and uncheck the *Set as Main Project* check box. Click *Next* to display the Server and Settings dialog box.
4. Choose **`GlassFish Server 4`** in the Server field, and **`Java EE 7 Web`** as in the Java EE Version field, and click Finish to create the project.

To use `QuizWebService`, you need to create a Web service client as follows:

1. Right-click the `QuizWebServiceClientProject` 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 `myWebservice` in the Package field. 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 41.6 and 41.7 show DisplayQuiz.jsp and GradeQuiz.jsp.

LISTING 41.6 DisplayQuiz.jsp

```
<!-- DisplayQuiz.jsp -->
<%@ page import = "myWebservice.QuizWebService" %>
<%@ page import = "myWebservice.QuizService" %>
<jsp:useBean id = "quizWebService" scope = "session"
  class = "myWebservice.QuizWebService">
</jsp:useBean>

<html>
<body>
  <%
  QuizService proxy = quizWebService.getQuizServicePort();
  java.util.List<String> questions =
    (java.util.ArrayList<String>)(proxy.getQuestions());
  %>
  <form method = "post" action = "GradeQuiz.jsp">
  <table>
    <% for (int i = 0; i < questions.size(); i++) {%>
    <tr>
    <td>
      <label><%= questions.get(i) %></label>
    </td>
    <td>
      <input type = "radio" name = <%= "question" + i%>
        value = "True" /> True
    </td>
    <td>
      <input type = "radio" name = <%= "question" + i%>
      value = "False" /> False
    </td>
    </tr>
    <%}%>
  </table>
  <p><input type = "submit" name = "Submit" value = "Submit">
    <input type = "reset" value = "Reset">
```

```
35     </p>
36     </form>
37   </body>
38   </html>
```

This page generates a quiz by invoking the `getQuestions()` in lines 12–13. The questions are displayed in a table with radio buttons (lines 16–32). Clicking the *Submit* button invokes GradeQuiz.jsp.

LISTING 41.7 GradeQuiz.jsp

```
 1   <!-- GradeQuiz.jsp -->
 2   <%@ page import = "myWebservice.QuizWebService" %>
 3   <%@ page import = "myWebservice.QuizService" %>
 4   <jsp:useBean id = "quizWebService" scope = "session"
 5     class = "myWebservice.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 41.5.

Check Point

41.6.1 What is the annotation to specify a Web service? What is the annotation to specify a Web method?

41.6.2 How do you deploy a Web service in NetBeans?

41.6.3 Can you test a Web service from a client?

41.6.4 How do you create a Web service reference for a client?

41.6.5 What is WSDL? What is SOAP? What is a SOAP request? What is a SOAP response?

41.6.6 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?

41.6.7 How do you obtain an `HttpSession` object for tracking a Web session?

41.6.8 Can you create two Web service references in one package in the same project in NetBeans?

41.6.9 What happens if you don't clone the quiz in lines 40–41 in Listing 41.5, QuizService.java?

Key Terms

`@WebService`	41-3	publishing a Web service	41-2
`@WebMethod`	41-3	Web service	41-2
consuming a Web service	41-2	Web service client reference	41-15
proxy object	41-2	WSDL	41-6

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 client 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 the 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.

Quiz

Answer the quiz for this chapter online at the book Companion Website.

Programming Exercises

MyProgrammingLab™

***41.1** (*Get a score from a database table)* Suppose 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 41.1, ScoreService.java, to obtain a score for the specified name. Note your program does not need the `permission` column; ignore it. The next exercise will need the `permission` column.

***41.2** (*Permission to find scores)* Revise the preceding exercise so that the `findScore` method returns `–1` if permission is `false`. Add an 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.

***41.3** (*Compute loan*) You can compute a loan payment for a loan with the specified amount, the number of years, and the 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 the loan amount, the number of years, and the annual interest rate.

***41.4** (*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.

***41.5** (*Quiz*) The user needs to answer all five questions before clicking the *Submit* button in the Quiz application in Section 41.6, Web Service Session Tracking. A runtime error will occur if a radio button is not checked. Fix this problem.