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 object type of arguments in a Web service (§42.5).
• To discover how a client communicates with a Web service (§42.5).
• To describe what are SOAP request and SOAP response (§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

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, select **GlassFish V2** as the server, and **Java EE 5** as the Java EE version. Click Finish to create the project.

**<Side Remark: create Web service class>**
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.

**Listing 42.1 ScoreService.java**

```java
***PD: Please add line numbers in the following code***
***Layout: Please layout exactly. Don’t skip the space. This is true for all source code in the book. Thanks, AU.***

```

```java
package chapter42;

import java.util.HashMap;
import javax.jws.WebService; // For annotation @WebService
import javax.jws.WebMethod; // For annotation @WebMethod

@WebService(name = "ScoreService", serviceName = "ScoreWebService")
public class ScoreService {
    // Stores scores in a map indexed by name
    private HashMap<String, Double> scores = new HashMap<String, Double>();

    public ScoreService() {
        scores.put("John", 90.5);
        scores.put("Michael", 100.0);
        scores.put("Michelle", 98.5);
    }

    @WebMethod(operationName = "findScore")
    public double findScore(String name) {
        Double d = scores.get(name);
        if (d == null) {
            System.out.println("Student " + name + " is not found ");
            return -1;
        }
```

```java
```
```
```
else {
    System.out.println("Student " + name + ",
    + studentScore + " score is ");
    return studentScore;
}
}

<Side Remark: what is annotation?>
<Side Remark: boilerplate code>
Lines 4-5 imports 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
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

<Side Remark: 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 **Undeploy and 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

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 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.
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

<Side Remark: ipconfig>
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.

***END NOTE

<Side Remark: Windows firewall>
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 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.

***END NOTE

<Side Remark: keep server live>
NOTE: If your application server is launched from NetBeans and you stop NetBeans, the application server is also stopped. To keep application server live, you may launch it independently from Windows Start button by choosing All Programs, Sun Microsystems, Application Server PE 9, Start Default Server. To shut it down, choose Stop Default Server from the same location.

***END NOTE

42.4 Consuming Web Services

<Side Remark: 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 applications, or another Web service) and written in any language.
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.

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

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.
2. In the New Project dialog box, choose General 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, Uncheck the Create Main Class checkbook. Click Finish to create the project.

<Side Remark: 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 the New Web Service Client dialog box, as shown in Figure 42.6.
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.
Figure 42.6
The New Web Service Client dialog box creates a Web service reference.

Now you will see ScoreWebService create 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
The boilerplate code for Web services are automatically generated in the client’s project.

.Side Remark: what is WSDL?
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 xml-resources 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, etc.
The .wsdl file describes Web services to clients.

**Side Remark: refresh reference**

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 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`, as shown in Listing 42.2.

**Listing 42.2 FindScoreApplet.java**

```java
package chapter42;
import chapter42.ScoreWebService;
import chapter42.ScoreService;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

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

---

**Figure 42.8**

The .wsdl file describes Web services to clients.
private JButton jbtGetScore = new JButton("Get Score");
private JTextField jtfName = new JTextField();
private JTextField jtfScore = new JTextField();

public void init() {
    JPanel jPanel1 = new JPanel();
    jPanel1.setLayout(new GridLayout(2, 2));
    jPanel1.add(new JLabel("Name"));
    jPanel1.add(jtfName);
    jPanel1.add(new JLabel("Score"));
    jPanel1.add(jtfScore);
    add(jbtGetScore, BorderLayout.SOUTH);
    add(jPanel1, BorderLayout.CENTER);
}

jbtGetScore.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        getScore();
    }
});

private void getScore() {
    try {
        // Get student score
        double score = proxy.findScore(jtfName.getText().trim());

        // Display the result
        if (score < 0)
            jtfScore.setText("Not found");
        else
            jtfScore.setText(new Double(score).toString());
    } catch(Exception ex) {
        ex.printStackTrace();
    }
}

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

<Side Remark: SOAP>

In the preceding example, you created a Web service client that 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.

<Side Remark: SOAP request>
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.

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

```xml
<nsl:findScore>
  <arg0>Michael</arg0>
</nsl: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

```xml
<nsl:findScoreResponse>
  <return>100.0</return>
</nsl:findScoreResponse>
```
specifies that the method returns 100.0.

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

![Figure 42.10](image)

A proxy object sends SOAP requests and receives SOAP responses.

<Side Remark: XML serialization>
<Side Remark: XML deserialization>

Can you pass an argument of any type between a client and a Web service? No. SOAP only supports 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/deserialization 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:

<Side Remark: no-arg constructor>

1. The class must have a no-arg constructor.

<Side Remark: get and set methods>

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 a `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 37.12, `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.

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

```
package chapter42;

import chapter39.Address;
import java.sql.*;
import javax.jws.WebMethod;
import javax.jws.WebService;

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

**Listing 42.3 AddressService.java**

***PD: Please add line numbers in the following code***

***Layout: Please layout exactly. Don’t skip the space.***

This is true for all source code in the book. Thanks, AU.
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() {
    try {
        Class.forName("com.mysql.jdbc.Driver");

        // Connect to the sample database
        Connection connection = DriverManager.getConnection("jdbc:mysql://localhost/javabook", "scott", "tiger");

        statement1 = connection.prepareStatement("select * from Address where firstName = ? and lastName = ?");
        statement2 = connection.prepareStatement("insert into Address (lastName, firstName, mi, telephone, email, street, city, state, zip) values (?, ?, ?, ?, ?, ?, ?, ?)");
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
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 29-39). 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:**

*Side Remark: database driver*

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 `Undeploy and 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.

**Figure 42.11**
The `TestAddressWebService` page allows the user to search and store addresses.
Figure 42.12
The Search button finds and displays an address.

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. Enter chapter42 in the Package field.
4. 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.
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.

```jsp
<%@ page import = "chapter42.Address" %>
<%@ page import = "chapter42.AddressWebService" %>
<%@ page import = "chapter42.AddressService" %>
<jsp:useBean id = "addressId" class = "chapter42.Address" scope = "session">
<jsp:setProperty name = "addressId" property = "*"/>
<html>
<head>
<title>Address Information</title>
</head>
<body>
<form method = "post" action = "TestAddressWebService.jsp">
  Last Name <input type = "text" name = "lastName">
</form>
```

Figure 42.14

The Address.java is automatically copied to the Web service client reference package.
<%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;

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>

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

Zip
<input type="text" name="zip"
    <%if (addressId.getZip() != null) {
        out.print("value = " + addressId.getZip() + ",";'>
    size = "9" /&nbsp;
</p>
<p><input type = "submit" name = "Submit" value = "Search">
<input type = "submit" name = "Submit" value = "Store">
<input type = "reset" value = "Reset">
</p></form>
<p><font color = "#FF0000">* required fields</font></p>
<% if (request.getParameter("Submit") != null) {
    AddressWebService addressWebService = new AddressWebService();
    AddressService proxy = addressWebService.getAddressServicePort();

    if (request.getParameter("Submit").equals("Store")) {
        proxy.storeAddress(addressId);
        out.println(addressId.getFirstName() + " " + addressId.getLastName() + " has been added to the database");
    }
        else if (request.getParameter("Submit").equals("Search")) {
            Address address = proxy.getAddress(addressId.getFirstName(), addressId.getLastName());
            if (address == null)
                out.println(addressId.getFirstName() + " " + addressId.getLastName() + " is not in the database");
            else
                addressId = address;
    }
  }
%></body></html>

Lines 2-4 import the classes for the JSP page. The Address class (line 2) was created in the WebServicePorject and was automatically copied to the liangweb 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

<Side Remark: 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.

![Figure 42.15](image_url)

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

![Figure 42.16](image_url)

**Figure 42.16**
The answers are graded and displayed.

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
again 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 chapter42.
Listing 42.5 gives the program.

```java
/*PD: Please add line numbers in the following code*/
/*Layout: Please layout exactly. Don’t skip the space.*/
This is true for all source code in the book. Thanks, AU.

<Side Remark line 13: define service name>
<Side Remark line 15: @Resource annotation>
<Side Remark line 16: HttpSession>
<Side Remark line 17: quiz>
<Side Remark line 21: initialize quiz>
<Side Remark line 33: define remote method>
<Side Remark line 34: generateQuiz>
```

1722
package chapter42;

import javax.annotation.Resource;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import javax.xml.ws.WebServiceContext;
import javax.xml.ws.handler.MessageContext;
import java.util.List;
import java.util.ArrayList;

@WebService(name = "QuizService", serviceName = "QuizWebService")
public class QuizService {
    private @Resource WebServiceContext webServiceContext;
    private HttpSession session;
    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
        });
        quiz.add(new Object[] {
            "Is New Orleans the capital of Louisiana?", false
        });
        quiz.add(new Object[] {
            "Is Chicago the capital of Illinois?", false
        });
    }

    @WebMethod(operationName = "generateQuiz")
    public void generateQuiz() {
        MessageContext messageContext = webServiceContext.getMessageContext();
        session = ((HttpServletRequest) messageContext.get(MessageContext.SERVLET_REQUEST)).getSession();
        java.util.ArrayList cloneQuiz = (java.util.ArrayList)(quiz.clone());
        // Shuffle to generate a random quiz for a client
java.util.Collections.shuffle(cloneQuiz);

// Store the quiz in the session
session.setAttribute("quiz", quiz);
}

@WebMethod(operationName = "getQuestions")
public java.util.List<String> getQuestions() {
    List<Object[]> quiz = (List<Object[]>)session.getAttribute("quiz");

    // Extract questions from quiz
    List<String> questions = new ArrayList<String>();
    for (int i = 0; i < quiz.size(); i++) {
        questions.add((String)(quiz.get(i)[0]));
    }

    return questions; // Return questions in the quiz
}

@WebMethod(operationName = "gradeQuiz")
public List<Boolean> gradeQuiz(List<Boolean> answers) {
    // Retrieve the quiz from the session
    List<Object[]> quiz = (List<Object[]>)session.getAttribute("quiz");

    List<Boolean> result = new ArrayList<Boolean>();
    for (int i = 0; i < quiz.size(); i++) {
        result.add(quiz.get(i)[1] == answers.get(i));
    }

    return result;
}

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.

**<Side Remark: generateQuiz>**

**<Side Remark: randomly shuffling>**

The generateQuiz method (lines 33-45) 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 40-42. A new quiz is generated by randomly shuffling the original quiz (line 44). The new quiz is stored in the session (line 47).
<Side Remark: getQuestions>
The getQuestions method (lines 50-62) returns questions in a list. Line 52-53 obtains the quiz from the session. The questions are extracted from the quiz (lines 56-59) and are returned (line 61).

<Side Remark: gradeQuiz>
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).

<Side Remark: create Web service client>
After creating and publishing the Web service, you can create DisplayQuiz.jsp and GradeQuiz.jsp in the liangweb project. The liangweb project 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

```jsp
<%@ page import = "chapter42Quiz.QuizWebService" %>

<!-- DisplayQuiz.jsp -->
```
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

***PD: Please add line numbers in the following code***
***Layout: Please layout exactly. Don’t skip the space. This is true for all source code in the book. Thanks, AU.***

```jsp
<%@ page import = "chapter42Quiz.QuizService" %>
<jsp:useBean id = "quizWebService" scope = "session"
    class = "chapter42Quiz.QuizWebService">
</jsp:useBean>

<html>
<body>
<%
    QuizService proxy = quizWebService.getQuizServicePort();
    proxy.generateQuiz();
    java.util.List<String> questions =
    (java.util.ArrayList<String>)(proxy.getQuestions());
    %>
<form method = "post" action = "GradeQuiz.jsp">
<table>
    <jsp: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>
    </jsp:for>
</table>
<p><input type = "submit" name = "Submit" value = "Submit" />
    <input type = "reset" value = "Reset" />
</p>
</form>
</body>
</html>
```
This page collects the answers passed from the HTML form from the DisplayQuiz page (lines 15-22), invokes the gradeQuiz method to grade the quiz (line 25), finds the correct count (lines 28-32), and display the result (line 35).

NOTE:
You need to answer all the 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

**PD: Please place terms in two or three columns like this: (I will insert the page number in the proof)**

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>@WebService</td>
<td></td>
</tr>
<tr>
<td>@WebMethod</td>
<td></td>
</tr>
<tr>
<td>consuming Web service</td>
<td></td>
</tr>
<tr>
<td>proxy object</td>
<td></td>
</tr>
<tr>
<td>publishing Web service</td>
<td></td>
</tr>
<tr>
<td>Web service</td>
<td></td>
</tr>
<tr>
<td>Web service client reference</td>
<td></td>
</tr>
<tr>
<td>WSDL</td>
<td></td>
</tr>
</tbody>
</table>

### Chapter Summary

- Web services enable a Java program on one system to invoke a method in an object on another system.
- Web service is platform and language independent. You can develop and use Web services using any languages.
- Web services run on the Web using HTTP. SOAP is a popular protocol for implementing Web services.
- 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.
- You need to use Java annotation @WebService to annotate a Web service and use annotation @WebMethod to annotate a remote method.
- A Web service class may have unlimited number of remote methods.
- After a Web service is published, you can write a client program to use it. You have to first create Web client reference. From the reference, you create a proxy object for facilitating the communications between a server and a client.
- 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, etc.
- The message between the client and server is described in XML. SOAP request describes the information that is sent to the Web service and SOAP response describes the information that is received from the Web service.
The objects passed between client and Web service are serialized in XML. Not all object types are supported by SOAP.

You can track session 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 Does Web services support callback, i.e., a Web service can call a method from a client’s program?

42.4 What is SOAP? What is to publish a Web service? What is 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, QuizService.java?

Programming Exercises

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

```java
create table Scores (name varchar(20),
                    score number, permission number);
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 a 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 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 the 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.