Chapter 28: Web Services in Java

Internet & World Wide Web
How to Program, 5/e

Note: This chapter is a copy of Chapter 31 of our book *Java How to Program, 9/e*. For that reason, we simply copied the PowerPoint slides for this chapter and *did not* re-numb er them

OBJECTIVES

In this chapter you will learn:

- What a web service is.
- How to publish and consume web services in NetBeans.
- How XML, JSON, XML-Based Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) Architecture enable Java web services.
- How to create client desktop and web applications that consume web services.
- How to use session tracking in web services to maintain client state information.
- How to connect to databases from web services.
- How to pass objects of user-defined types to and return them from a web service.

31.1 Introduction

- A web service is a software component stored on one computer that can be accessed by an application (or other software component) on another computer over a network.
- Web services communicate using such technologies as XML, JSON and HTTP.
- In this chapter, we use two Java APIs that facilitate web services.
  - JAX-WS is based on the Simple Object Access Protocol (SOAP)—an XML-based protocol that allows web services and clients to communicate, even if the client and the web service are written in different languages.
  - JAX-RS uses Representational State Transfer (REST)—a network architecture that uses the web's traditional request/response mechanisms such as `GET` and `POST` requests.

31.1 Introduction (cont.)

- Business-to-Business Transactions
  - Rather than relying on proprietary applications, businesses can conduct transactions via standardized, widely available web services.
  - This has important implications for business-to-business (B2B) transactions.
  - Web services are platform and language independent, enabling companies to collaborate without worrying about the compatibility of their hardware, software and communications technologies.

31.2 Web Service Basics

- The machine on which a web service resides is referred to as a web service host.
- The client application sends a request over a network to the web service host, which processes the request and returns a response over the network to the application.
- In Java, a web service is implemented as a class.
- The class that represents the web service resides on a server—it's not part of the client application.
- Making a web service available to receive client requests is known as publishing a web service; using a web service from a client application is known as consuming a web service.

31.3 Simple Object Access Protocol (SOAP)

- The Simple Object Access Protocol (SOAP) is a platform-independent protocol that uses XML to interact with web services, typically over HTTP.
- Each request and response is packaged in a SOAP message.
  - Written in XML so that they are computer readable, human readable and platform independent.
- Most firewalls allow HTTP traffic to pass through, so that clients can browse the web by sending requests to and receiving responses from web servers.
  - Thus, SOAP-based services can send and receive SOAP messages over HTTP connections with few limitations.

31.3 Simple Object Access Protocol (SOAP) (cont.)

- The wire format used to transmit requests and responses must support all types passed between the applications.
- When a program invokes a method of a SOAP web service, the request and all relevant information are packaged in a SOAP message enclosed in a SOAP envelope and sent to the server on which the web service resides.
- When the web service receives this SOAP message, it parses the XML representing the message, then processes the message's contents.
- The message specifies the method that the client wishes to execute and the arguments the client passed to that method.
- The web service calls the method with the specified arguments (if any) and sends the response back to the client in another SOAP message.
- The client parses the response to retrieve the method's result.

31.4 Representational State Transfer (REST)

- Representational State Transfer (REST) refers to an architectural style for implementing web services.
  - Often called RESTful web services.
- RESTful web services are implemented using web standards.
- Each method in a RESTful web service is identified by a unique URL.
- Thus, when the server receives a request, it immediately knows what operation to perform.
  - Can be used in a program or directly from a web browser.
  - The results of a particular operation may be cached locally by the browser when the service is invoked with a `GET` request.

31.5 JavaScript Object Notation (JSON)

- JavaScript Object Notation (JSON) is an alternative to XML for representing data.
  - Text-based data-interchange format used to represent objects in JavaScript as collections of name/value pairs represented as `String`s.
  - Commonly used in Ajax applications.
  - Makes objects easy to read, create and parse
  - Much less verbose than XML, so it allows programs to transmit data efficiently across the Internet

31.5 JavaScript Object Notation (JSON) (cont.)

- Each JSON object is represented as a list of property names and values contained in curly braces:
  - `{` *propertyName1* `:` *value1*`,` *propertyName2* `:` *value2* `}`
- Arrays are represented with square brackets:
  - `[` *value1*`,` *value2*`,` *value3* `]`
  - Each value in an array can be a string, a number, a JSON object, `true`, `false` or `null`.
- Representation of an array of address-book entries:

```
[ { first: 'Cheryl', last: 'Black' },
  { first: 'James', last: 'Blue' },
  { first: 'Mike', last: 'Brown' },
  { first: 'Meg', last: 'Gold' } ]
```

31.6 Publishing and Consuming SOAP-Based Web Services

- This section presents our first example of publishing (enabling for client access) and consuming (using) a web service.
- We begin with a SOAP-based web service.

31.6.1 Creating a Web Application Project and Adding a Web Service Class in NetBeans

- To create a web service in NetBeans, you first create a Web Application project.
- To create a web application, perform the following steps:
  1. Select File > New Project… to open the New Project dialog.
  2. Select Java Web from the dialog's Categories list, then select Web Application from the Projects list. Click Next >.
  3. Specify the name of your project in the Project Name field and specify where you'd like to store the project in the Project Location field. You can click the Browse button to select the location. Click Next >.
  4. Select GlassFish Server 3 from the Server drop-down list and Java EE 6 from the Java EE Version drop-down list.
  5. Click Finish to create the project.

31.6.1 Creating a Web Application Project and Adding a Web Service Class in NetBeans (cont.)

- Perform the following steps to add a web service class to the project:
  - 1. In the Projects tab in NetBeans, right click the project's node and select New > Web Service… to open the New Web Service dialog.
  - 2. Specify the name of the service in the Web Service Name field.
  - 3. Specify the package name in the Package field.
  - 4. Click Finish to create the web service class.
- The IDE generates a sample web service class with the name from *Step 2* in the package from *Step 3*.
- You can find this class in your project's Web Services node.
- In this class, you'll define the methods that your web service makes available to client applications.

31.6.2 Defining the `WelcomeSOAP` Web Service in NetBeans

- Figure 31.1 contains the `WelcomeSOAPService` code.
- By default, each new web service class created with the JAX-WS APIs is a POJO (plain old Java object)
  - You do *not* need to extend a class or implement an interface to create a web service.

```
1  // Fig. 31.1: WelcomeSOAP.java
2  // Web service that returns a welcome message via SOAP.
3  package com.deitel.welcomesoap;
4
5  import javax.jws.WebService; // program uses the annotation @WebService
6  import javax.jws.WebMethod; // program uses the annotation @WebMethod
7  import javax.jws.WebParam; // program uses the annotation @WebParam
8
9  @WebService() // annotates the class as a web service
10 public class WelcomeSOAP
11 {
12    // WebMethod that returns welcome message
13    @WebMethod( operationName = "welcome" )
14    public String welcome( @WebParam( name = "name" ) String name )
15    {
16       return "Welcome to JAX-WS web services with SOAP, " + name + "!";
17    } // end method welcome
18 } // end class WelcomeSOAP
```

Fig. 31.1 | Web service that returns a welcome message via SOAP.

31.6.2 Defining the `WelcomeSOAP` Web Service in NetBeans (cont.)

- When you deploy a web application containing a class that uses the `@WebService` annotation, the server recognizes that the class implements a web service and creates all the server-side artifacts that support the web service
  - Framework that allows the web service to wait for client requests and respond to those requests once it is deployed on an application server.

31.6.2 Defining the `WelcomeSOAP` Web Service in NetBeans (cont.)

- The `@WebService` annotation indicates that a class implements a web service.The annotation is followed by parentheses containing optional annotation attributes.
- The `name` attribute specifies the name of the service endpoint interface class that will be generated for the client.
- A service endpoint interface (SEI) class (sometimes called a proxy class) is used to interact with the web service
  - a client application consumes the web service by invoking methods on the service endpoint interface object.
- The `serviceName` attribute specifies the service name, which is also the name of the class that the client uses to obtain a service endpoint interface object.
  - If not specified, the web service's name is assumed to be the java class name followed by the word `Service`.

31.6.2 Defining the `WelcomeSOAP` Web Service in NetBeans (cont.)

- A method is tagged with the `@WebMethod` annotation to indicate that it can be called remotely.
  - Any methods that are not tagged with `@WebMethod` are not accessible to clients that consume the web service.
- An `@WebMethod` annotation's `operationName` attribute to specifies the method name that is exposed to the web service's client.
  - If the `operationName` is not specified, it is set to the actual Java method's name.
- A parameter is annotated with the `@WebParam` annotation.
- The optional `@WebParam` attribute `name` indicates the parameter name that is exposed to the web service's clients.
  - If you don't specify the name, the actual parameter name is used.

Common Programming Error 31.1

Failing to expose a method as a web method by declaring it with the `@WebMethod` *annotation prevents clients of the web service from accessing the method. There's one exception—if none of the class's methods are declared with the* `@WebMethod` *annotation, then all the* `public` *methods of the class will be exposed as web methods.*

Common Programming Error 31.2

Methods with the `@WebMethod` *annotation cannot be* `static`*. An object of the web service class must exist for a client to access the service's web methods.*

31.6.2 Defining the `WelcomeSOAP` Web Service in NetBeans (cont.)

- To define the `WelcomeSOAP` class's `welcome` method, perform the following steps:
  - 1. In the project's **Web Services** node, double click `WelcomeSOAP` to open the file `WelcomeSOAPService.java` in the code editor.
  - 2. Click the **Design** button at the top of the code editor to show the web service design view (Fig. 31.2).

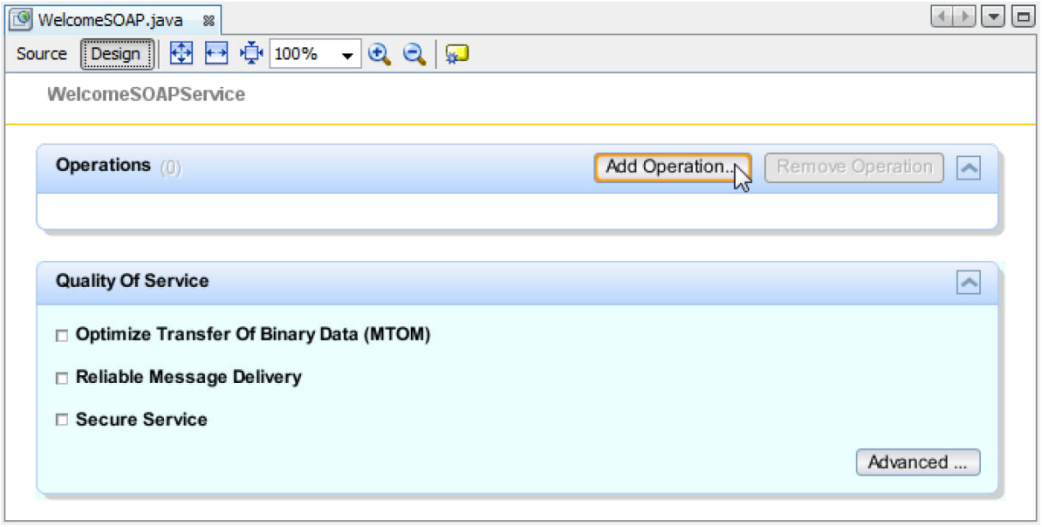


Fig. 31.2 | Web service design view.

31.6.2 Defining the `WelcomeSOAP` Web Service in NetBeans (cont.)

- 3. Click the Add Operation… button to display the Add Operation… dialog (Fig. 31.3).
- 4. Specify the method name `welcome` in the Name field. The default Return Type (`String`) is correct for this example.
- 5. Add the method's `name` parameter by clicking the Add button to the right of the Parameters tab then entering name in the Name field. The parameter's default Type (`String`) is correct for this example.
- 6. Click OK to create the `welcome` method. The design view should now appear as shown in Fig. 31.3.
- 7. At the top of the design view, click the Source button to display the class's source code and add the code line 18 of Fig. 31.1 to the body of method `welcome`.

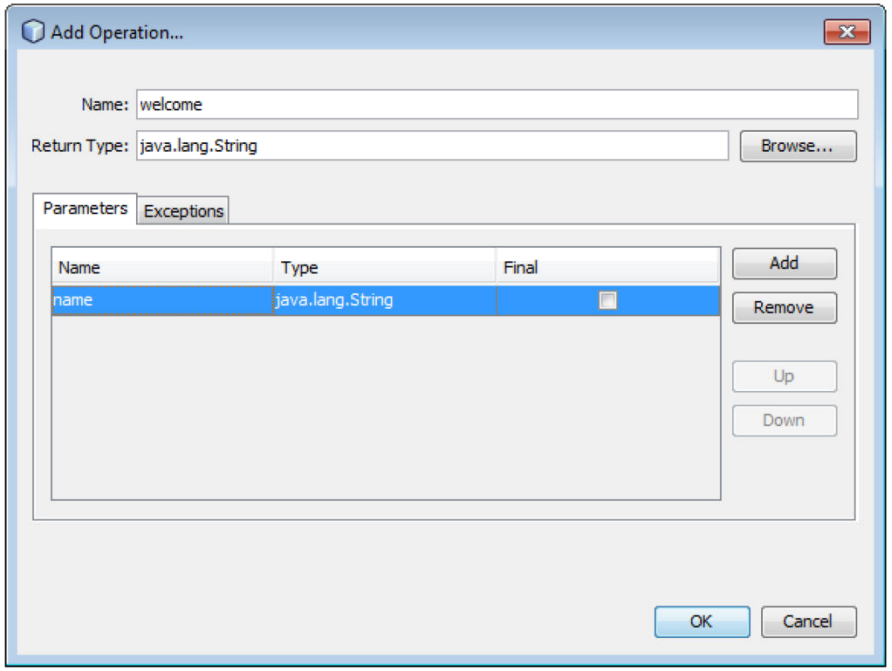


Fig. 31.3 | Adding an operation to a web service.

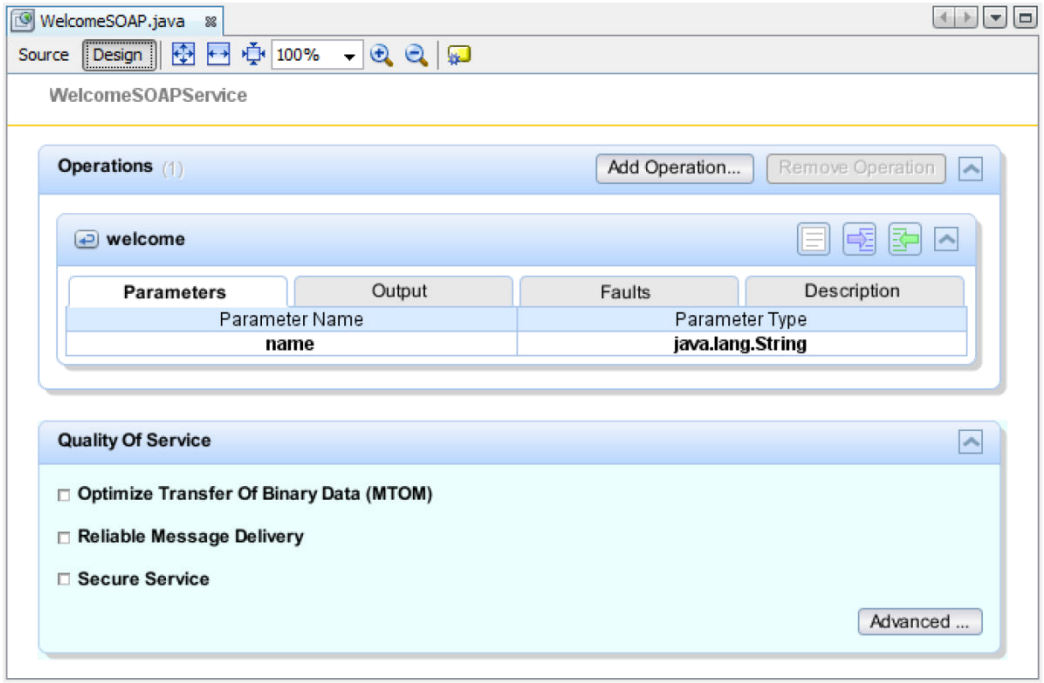


Fig. 31.4 | Web service design view after new operation is added.

31.6.3 Publishing the WelcomeSOAP Web Service from NetBeans

- NetBeans handles all the details of building and deploying a web service for you.
  - This includes creating the framework required to support the web service.
- Right click the project name `WelcomeSoap` in the Projects tab and select Deploy to build and deploy the web application in the GlassFish server.

31.6.4 Testing the `WelcomeSOAP` Web Service with GlassFish Application Server's `Tester` Web Page

- The GlassFish application server can dynamically create a web page for testing a web service's methods from a web browser.
- Expand the project's **Web Services** in the NetBeans **Projects** tab.
- Right click the web service class name and select **Test Web Service**.
- The GlassFish application server builds the `Tester` web page and loads it into your web browser.
- Figure 31.5 shows the `Tester` web page for the `WelcomeSOAP` web service.

Fig. 31.5 | `Tester` web page created by GlassFish for the `WelcomeSOAP` web service.

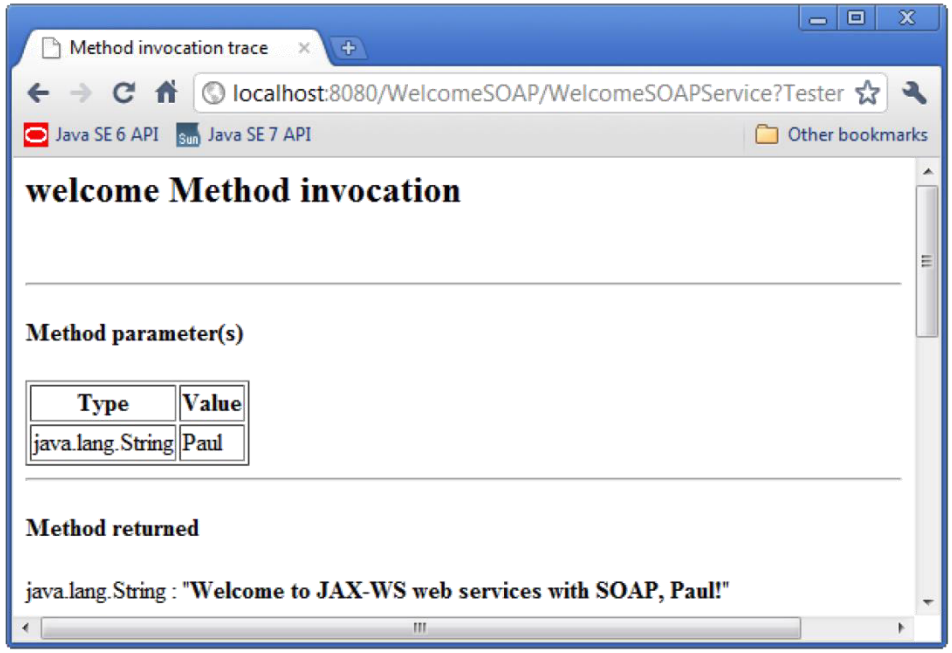


Fig. 31.6 | Testing `WelcomeSOAP`'s `welcome` method.

31.6.4 Testing the WelcomeSOAP Web Service with GlassFish Application Server's Tester Web Page (cont.)

- If your computer is connected to a network and allows HTTP requests, then you can test the web service from another computer on the network by typing the following URL (where *host* is the hostname or IP address of the computer on which the web service is deployed) into a browser on another computer:
  - `http://`*host*`:8080/WelcomeSoap/WelcomeSoapService?Tester`

31.6.5 Describing a Web Service with the Web Service Description Language (WSDL)

- To consume a web service, a client must determine its functionality and how to use it.
- Web services normally contain a service description.
  - Web Service Description Language (WSDL)—an XML vocabulary that defines the methods a web service makes available and how clients interact with them.
  - A WSDL document also specifies lower-level information that clients might need, such as the required formats for requests and responses.
- WSDL documents help applications determine how to interact with the web services described in the documents.
- GlassFish generates a web service's WSDL dynamically for you.

31.6.5 Describing a Web Service with the Web Service Description Language (WSDL) (cont.)

- To access the `WelcomeSOAP` web service, the client code will need the following WSDL URL:
  - `http://localhost:8080/WelcomeSoap/WelcomeSoapService?Tester`
- Eventually, you'll want clients on other computers to use your web service.
- Such clients need access to the web service's WSDL, which they would access with the following URL:
  - `http://`*host*`:8080/WelcomeSOAP/WelcomeSOAPService?WSDL`

  where *host* is the hostname or IP address of the server that hosts the web service.

31.6.6 Creating a Client to Consume the WelcomeSOAP Web Service

- You enable a Java-based client application to consume a web service by adding a web service reference to the client application.
  - Defines the service endpoint interface class that allows the client to access the web service.
- An application that consumes a web service consists of
  - an object of a service endpoint interface (SEI) class that's used to interact with the web service
  - a client application that consumes the web service by invoking methods on the service endpoint interface object
- The service endpoint interface object handles the details of passing method arguments to and receiving return values from the web service on the client's behalf.
- Figure 31.5 depicts the interactions among the client code, the SEI object and the web service.

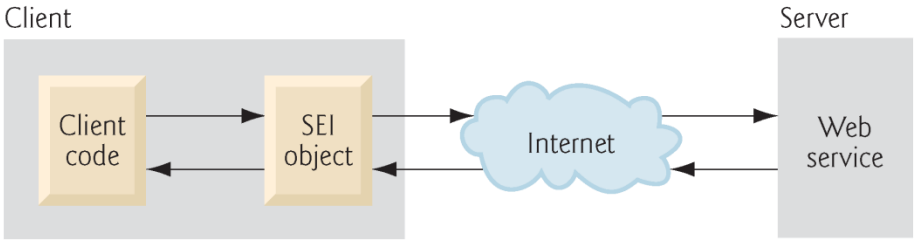


Fig. 31.7 | Interaction between a web service client and a web service.

31.6.6 Creating a Client to Consume the WelcomeSOAP Web Service (cont.)

- Requests to and responses from web services created with JAX-WS are typically transmitted via SOAP.
- Any client capable of generating and processing SOAP messages can interact with a web service, regardless of the language in which the web service is written.
- We now use NetBeans to create a client Java desktop GUI application.
- When you add a web service reference, the IDE creates and compiles the client-side artifacts
  - framework of Java code that supports the client-side service endpoint interface class.
- The client then calls methods on an object of the service endpoint interface class, which uses the rest of the artifacts to interact with the web service.

31.6.6 Creating a Client to Consume the `WelcomeSOAP` Web Service (cont.)

- Perform the following steps to create a client Java desktop application in NetBeans:
  - 1. Select **File > New Project…** to open the **New Project** dialog.
  - 2. Select **Java** from the **Categories** list and **Java Application** from the **Projects** list, then click **Next >**.
  - 3. Specify the name in the **Project Name** field and uncheck the **Create Main Class** checkbox if you intend to create a your own class that contains `main`.
  - 4. Click **Finish** to create the project.

31.6.6 Creating a Client to Consume the `WelcomeSOAP` Web Service (cont.)

- To add a web service reference, perform the following steps.
  - 1. Right click the project name (`WelcomeSOAPClient`) in the NetBeans **Projects** tab and select **New > Web Service Client…** from the pop-up menu to display the **New Web Service Client** dialog.
  - 2. In the **WSDL URL** field, specify the URL `http://localhost:8080/WelcomeSoap/WelcomeSoapService?WSDL` (Fig. 31.8). The IDE uses this WSDL to generate the client-side artifacts.
  - 3. For the other options, leave the default settings, then click **Finish** to create the web service reference and dismiss the **New Web Service Client** dialog.
- In the NetBeans **Projects** tab, the project now contains a **Web Service References** folder with the web service's service endpoint interface (Fig. 31.9).

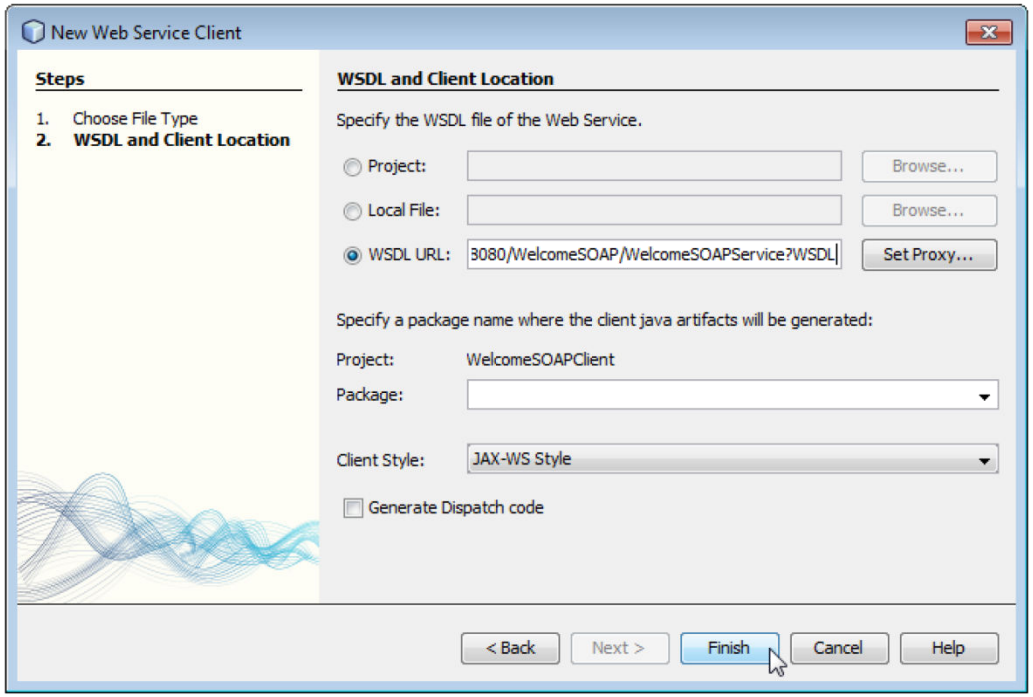


Fig. 31.8 | **New Web Service Client** dialog.

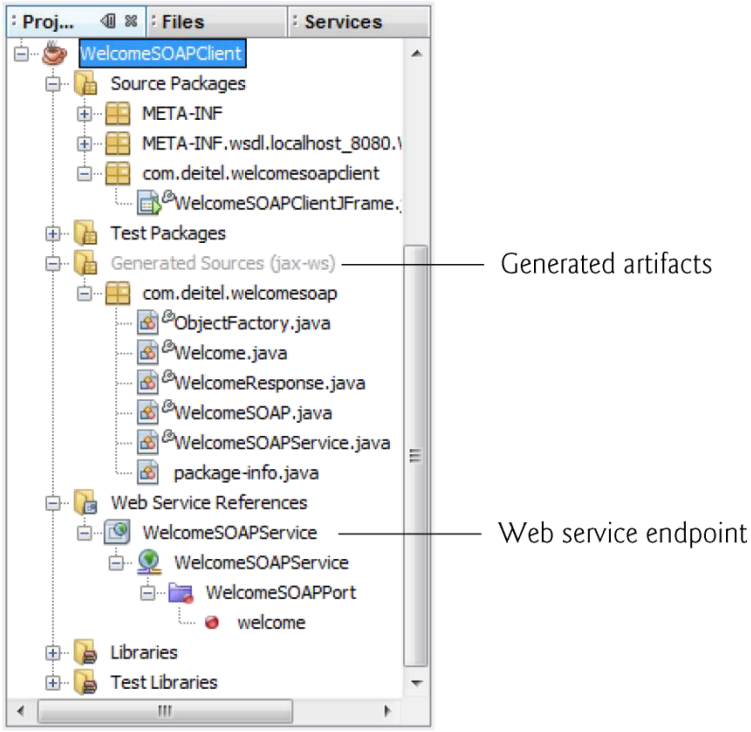


Fig. 31.9 | NetBeans **Project** tab after adding a web service reference to the project.

31.6.8 Consuming the `WelcomeSOAP` Web Service

- For this example, we use a GUI application to interact with the `WelcomeSOAP` web service.

```
1  // Fig. 31.10: WelcomeSOAPClientJFrame.java
2  // Client desktop application for the WelcomeSOAP web service.
3  package com.deitel.welcomesoapclient;
4
5  import com.deitel.welcomesoap.WelcomeSOAP;
6  import com.deitel.welcomesoap.WelcomeSOAPService;
7  import javax.swing.JOptionPane;
8
```

Fig. 31.10 | Client desktop application for the `WelcomeSOAP` web service. (Part 1 of 5.)

```
 9  public class WelcomeSOAPClientJFrame extends javax.swing.JFrame
10  {
11     // references the service endpoint interface object (i.e., the proxy)
12     private WelcomeSOAP welcomeSOAPProxy;
13
14     // no-argument constructor
15     public WelcomeSOAPClientJFrame()
16     {
17        initComponents();
18
19        try
20        {
21           // create the objects for accessing the WelcomeSOAP web service
22           WelcomeSOAPService service = new WelcomeSOAPService();
23           welcomeSOAPProxy = service.getWelcomeSOAPPort();
24        } // end try
25        catch ( Exception exception )
26        {
27           exception.printStackTrace();
28           System.exit( 1 );
29        } // end catch
30     } // end WelcomeSOAPClientJFrame constructor
31
```

Fig. 31.10 | Client desktop application for the `WelcomeSOAP` web service. (Part 2 of 5.)

```
32      // The initComponents method is autogenerated by NetBeans and is called
33      // from the constructor to initialize the GUI. This method is not shown
34      // here to save space. Open WelcomeSOAPClientJFrame.java in this
35      // example's folder to view the complete generated code.
36
37      // call the web service with the supplied name and display the message
38      private void submitJButtonActionPerformed(
39         java.awt.event.ActionEvent evt )
40      {
41         String name = nameJTextField.getText(); // get name from JTextField
42
43         // retrieve the welcome string from the web service
44         String message = welcomeSOAPProxy.welcome( name );
45         JOptionPane.showMessageDialog( this, message,
46            "Welcome", JOptionPane.INFORMATION_MESSAGE );
47      } // end method submitJButtonActionPerformed
48
```

Fig. 31.10 | Client desktop application for the `WelcomeSOAP` web service. (Part 3 of 5.)

```
49      // main method begins execution
50      public static void main( String args[] )
51      {
52         java.awt.EventQueue.invokeLater(
53            new Runnable()
54            {
55               public void run()
56               {
57                  new WelcomeSOAPClientJFrame().setVisible( true );
58               } // end method run
59            } // end anonymous inner class
60         ); // end call to java.awt.EventQueue.invokeLater
61      } // end main
62
63      // Variables declaration - do not modify
64      private javax.swing.JLabel nameJLabel;
65      private javax.swing.JTextField nameJTextField;
66      private javax.swing.JButton submitJButton;
67      // End of variables declaration
68   } // end class WelcomeSOAPClientJFrame
```

Fig. 31.10 | Client desktop application for the `WelcomeSOAP` web service. (Part 4 of 5.)

Fig. 31.10 | Client desktop application for the `WelcomeSOAP` web service. (Part 5 of 5.)

31.7 Publishing and Consuming REST-Based XML Web Services

- Now, we access a Java web service using the REST architecture.
- We recreate the `WelcomeSOAP` example to return data in plain XML format.

31.7.1 Creating a REST-Based XML Web Service

- The RESTful Web Services plug-in for NetBeans provides templates for creating RESTful web services, including ones that can interact with databases on the client's behalf.
- To create a RESTful web service:
  - 1. Right-click the WelcomeRESTXML node in the Projects tab, and select New > Other… to display the New File dialog.
  - 2. Select Web Services under Categories, then select RESTful Web Services from Patterns and click Next >.
  - 3. Under Select Pattern, ensure Simple Root Resource is selected, and click Next >.
  - 4. For this example, set the Resource Package to `com.deitel.welcomerestxml`, the Path to `welcome` and the Class Name to `welcomeRESTXMLResource`. Leave the MIME Type and Representation Class set to `application/xml` and `java.lang.String`, respectively. The correct configuration is shown in Fig. 31.11.
  - 5. Click Finish to create the web service.

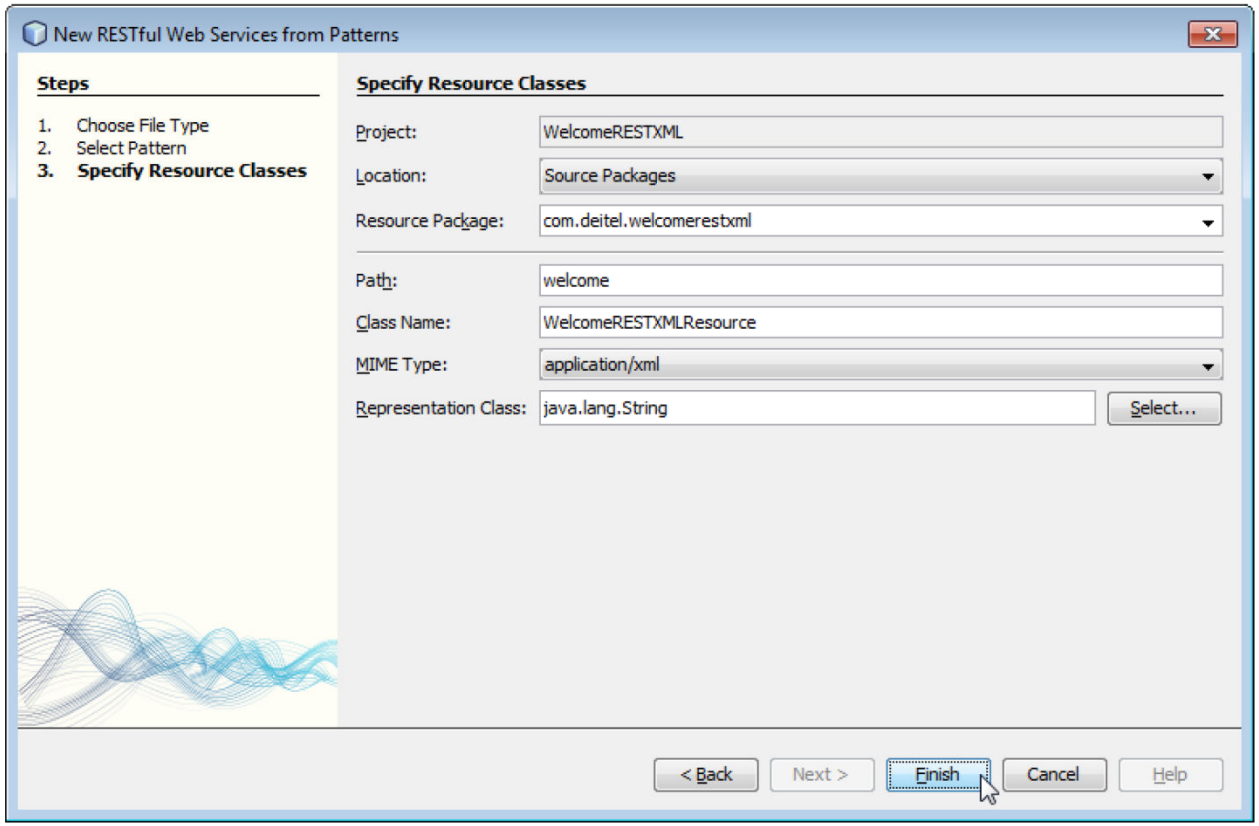


Fig. 31.11 | Creating the `WelcomeRESTXML` RESTful web service.

31.7.1 Creating a REST-Based XML Web Service (cont.)

- NetBeans generates the class and sets up the proper annotations.
- The class is placed in the project's RESTful Web Services folder.
- The code for the completed service is shown in Fig. 31.12.
- We removed some of the code generated by NetBeans that was unnecessary for this simple web service.
- The `@Path` annotation on the `WelcomeRESTXMLResource` class indicates the URI for accessing the web service.
  - This is appended to the web application project's URL to invoke the service.

```
1  // Fig. 31.12: WelcomeRESTXMLResource.java
2  // REST web service that returns a welcome message as XML.
3  package com.deitel.welcomerestxml;
4
5  import java.io.StringWriter;
6  import javax.ws.rs.GET; // annotation to indicate method uses HTTP GET
7  import javax.ws.rs.Path; // annotation to specify path of resource
8  import javax.ws.rs.PathParam; // annotation to get parameters from URI
9  import javax.ws.rs.Produces; // annotation to specify type of data
10 import javax.xml.bind.JAXB; // utility class for common JAXB operations
11
12 @Path( "welcome" ) // URI used to access the resource
```

Fig. 31.12 | REST web service that returns a welcome message as XML. (Part 1 of 2.)

```
13  public class WelcomeRESTXMLResource
14  {
15     // retrieve welcome message
16     @GET // handles HTTP GET requests
17     @Path( "{name}" ) // URI component containing parameter
18     @Produces( "application/xml" ) // response formatted as XML
19     public String getXml( @PathParam( "name" ) String name )
20     {
21        String message = "Welcome to JAX-RS web services with REST and " +
22           "XML, " + name + "!"; // our welcome message
23        StringWriter writer = new StringWriter();
24        JAXB.marshal( message, writer ); // marshal String as XML
25        return writer.toString(); // return XML as String
26     } // end method getXml
27  } // end class WelcomeRESTXMLResource
```

Fig. 31.12 | REST web service that returns a welcome message as XML. (Part 2 of 2.)

31.7.1 Creating a REST-Based XML Web Service (cont.)

- Methods of the class can also use the `@Path` annotation.
  - Parts of the path specified in curly braces indicate parameters—they are placeholders for values that are passed to the web service as part of the path.
- Arguments in a URL can be used as arguments to a web service method.
  - To do so, you bind the parameters specified in the `@Path` specification to parameters of the web service method with the `@PathParam` annotation.
  - When the request is received, the server passes the argument(s) in the URL to the appropriate parameter(s) in the web service method.

31.7.1 Creating a REST-Based XML Web Service (cont.)

- The `@GET` annotation denotes that this method is accessed via an HTTP `GET` request.
- The `@Produces` annotation denotes the content type returned to the client.
  - It is possible to have multiple methods with the same HTTP method and path but different `@Produces` annotations, and JAX-RS will call the method matching the content type requested by the client.
- The `@Consumes` annotation restricts the content type that the web service will accept from a client.

31.7.1 Creating a REST-Based XML Web Service (cont.)

- `JAXB` class from package `javax.xml.bind`.
  - JAXB (Java Architecture for XML Binding) is a set of classes for converting POJOs to and from XML.
  - `JAXB` class contains easy-to-use wrappers for common operations.
- `JAXB` `static` method `marshal` converts its argument to XML format.

31.7.1 Creating a REST-Based XML Web Service (cont.)

- GlassFish does not provide a testing facility for RESTful services, but NetBeans automatically generates a test page that can be accessed by right clicking the WelcomeRESTXML node in the Projects tab and selecting Test RESTful Web Services.
- On the test page (Fig. 31.12), expand the welcome element in the left column and select {name}.
- The right side of the page displays a form that allows you to choose the MIME type of the data (`application/xml` by default) and lets you enter the `name` parameter's value.
- Click the Test button to invoke the web service and display the returned XML

Error-Prevention Tip 31.1

At the time of this writing, the test page did not work in Google's Chrome web browser. If this is your default web browser, copy the test page's URL from Chrome's address field and paste it into another web browser's address field. Fig. 31.13 shows the test page in Mozilla Firefox.

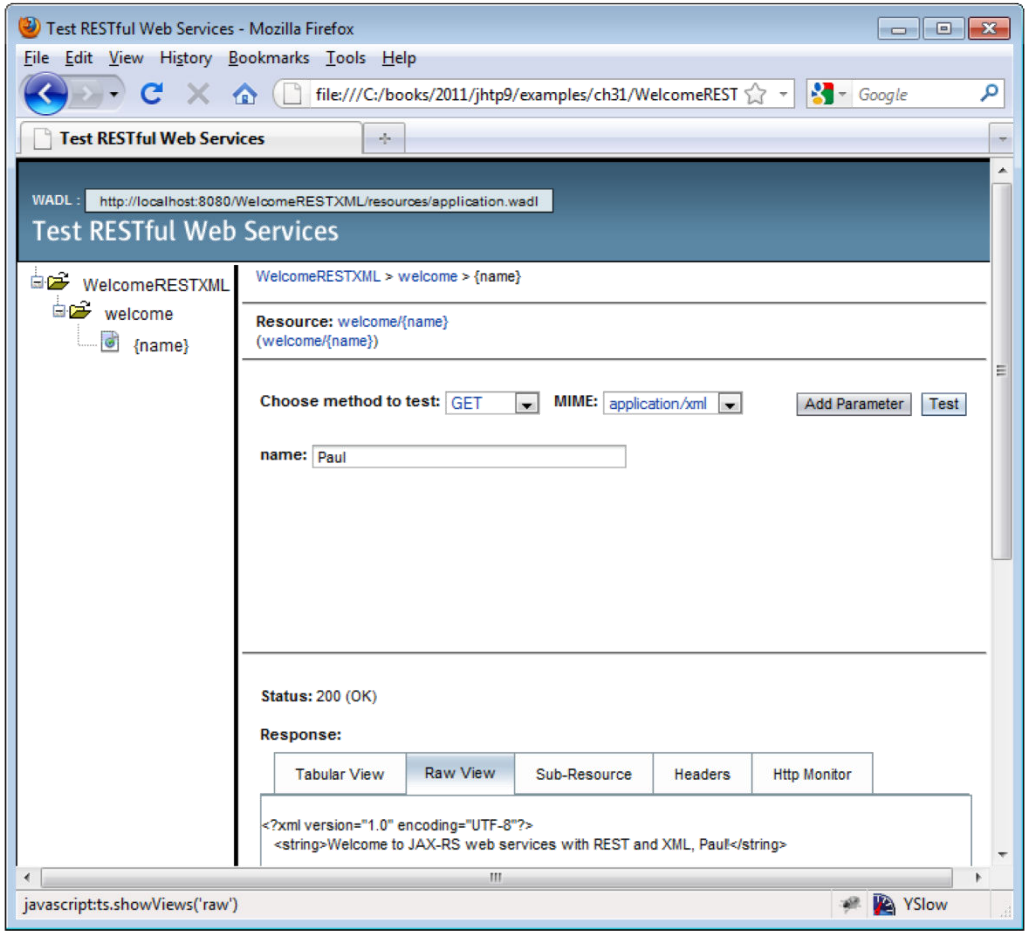


Fig. 31.13 | Test page for the `WelcomeRESTXML` web service.

31.7.1 Creating a REST-Based XML Web Service (cont.)

- The test page shows several tabs containing the results and various other information.
- The Raw View tab shows the actual XML response.
- The Headers tab shows the HTTP headers returned by the server.
- The Http Monitor tab shows a log of the HTTP transactions that took place to complete the request and response.
- The test page provides its functionality by reading a WADL file from the server.
  - WADL (Web Application Description Language) has similar design goals to WSDL, but describes RESTful services instead of SOAP services.

31.7.2 Consuming a REST-Based XML Web Service

- RESTful web services do not require web service references.
- As in the RESTful XML web service, we use the JAXB library.
- JAXB `static` method `unmarshal` takes as arguments a file name or URL as a `String`, and a `Class<T>` object indicating the Java class to which the XML will be converted.

```
1  // Fig. 31.14: WelcomeRESTXMLClientJFrame.java
2  // Client that consumes the WelcomeRESTXML service.
3  package com.deitel.welcomerestxmlclient;
4
5  import javax.swing.JOptionPane;
6  import javax.xml.bind.JAXB; // utility class for common JAXB operations
7
8  public class WelcomeRESTXMLClientJFrame extends javax.swing.JFrame
9  {
10    // no-argument constructor
11    public WelcomeRESTXMLClientJFrame()
12    {
13       initComponents();
14    } // end constructor
15
16    // The initComponents method is autogenerated by NetBeans and is called
17    // from the constructor to initialize the GUI. This method is not shown
18    // here to save space. Open WelcomeRESTXMLClientJFrame.java in this
19    // example's folder to view the complete generated code.
20
```

Fig. 31.14 | Client that consumes the `WelcomeRESTXML` service. (Part 1 of 4.)

```
21      // call the web service with the supplied name and display the message
22      private void submitJButtonActionPerformed(
23         java.awt.event.ActionEvent evt)
24      {
25         String name = nameJTextField.getText(); // get name from JTextField
26
27         // the URL for the REST service
28         String url =
29            "http://localhost:8080/WelcomeRESTXML/resources/welcome/" + name;
30
31         // read from URL and convert from XML to Java String
32         String message = JAXB.unmarshal( url, String.class );
33
34         // display the message to the user
35         JOptionPane.showMessageDialog( this, message,
36               "Welcome", JOptionPane.INFORMATION_MESSAGE );
37      } // end method submitJButtonActionPerformed
38
```

Fig. 31.14 | Client that consumes the `WelcomeRESTXML` service. (Part 2 of 4.)

```
39      // main method begins execution
40      public static void main( String args[] )
41      {
42         java.awt.EventQueue.invokeLater(
43            new Runnable()
44            {
45               public void run()
46               {
47                  new WelcomeRESTXMLClientJFrame().setVisible( true );
48               } // end method run
49            } // end anonymous inner class
50         ); // end call to java.awt.EventQueue.invokeLater
51      } // end main
52
53      // Variables declaration - do not modify
54      private javax.swing.JLabel nameJLabel;
55      private javax.swing.JTextField nameJTextField;
56      private javax.swing.JButton submitJButton;
57      // End of variables declaration
58   } // end class WelcomeRESTXMLClientJFrame
```

Fig. 31.14 | Client that consumes the `WelcomeRESTXML` service. (Part 3 of 4.)

Fig. 31.14 | Client that consumes the `WelcomeRESTXML` service. (Part 4 of 4.)

31.8 Publishing and Consuming REST-Based JSON Web Services

- XML was designed primarily as a document interchange format.
- JSON is designed as a *data exchange format*.
- Data structures in most programming languages do not map directly to XML constructs.
- JSON is a subset of the JavaScript programming language, and its components—objects, arrays, strings, numbers—can be easily mapped to constructs in Java and other programming languages.
- There are many open-source JSON libraries for Java and other languages; you can find a list of them at `json.org`.
- We use the Gson library from `code.google.com/p/google-gson/`.

31.8.1 Creating a REST-Based JSON Web Service

- You must download the Gson library's JAR file, then add it to the project as a library.
- To do so, right click your project's Libraries folder, select Add JAR/Folder… locate the downloaded Gson JAR file and click Open.
- Note that the argument to the `@Produces` attribute is `"application/json"`.
- JSON does not permit strings or numbers to stand on their own—they must be encapsulated in a composite data type.
  - So, we created class `TextMessage` to encapsulate the `String` representing the message.
- `Gson` (from package `com.google.gson.Gson`) method `toJson` converts an object into its JSON `String` representation.
- RESTful services returning JSON can be tested in the same way as those returning XML.

```java
// Fig. 31.15: WelcomeRESTJSONResource.java
// REST web service that returns a welcome message as JSON.
package com.deitel.welcomerestjson;

import com.google.gson.Gson; // converts POJO to JSON and back again
import javax.ws.rs.GET; // annotation to indicate method uses HTTP GET
import javax.ws.rs.Path; // annotation to specify path of resource
import javax.ws.rs.PathParam; // annotation to get parameters from URI
import javax.ws.rs.Produces; // annotation to specify type of data

@Path( "welcome" ) // path used to access the resource
public class WelcomeRESTJSONResource
{
   // retrieve welcome message
   @GET // handles HTTP GET requests
   @Path( "{name}" ) // takes name as a path parameter
   @Produces( "application/json" ) // response formatted as JSON
   public String getJson( @PathParam( "name" ) String name )
   {
      // add welcome message to field of TextMessage object
      TextMessage message = new TextMessage(); // create wrapper object
      message.setMessage( String.format( "%s, %s!",
         "Welcome to JAX-RS web services with REST and JSON", name ) );
```

Fig. 31.15 | REST web service that returns a welcome message as JSON. (Part 1 of 2.)

```
24
25          return new Gson().toJson( message ); // return JSON-wrapped message
26       } // end method getJson
27    } // end class WelcomeRESTJSONResource
28
29    // private class that contains the message we wish to send
30    class TextMessage
31    {
32       private String message; // message we're sending
33
34       // returns the message
35       public String getMessage()
36       {
37          return message;
38       } // end method getMessage
39
40       // sets the message
41       public void setMessage( String value )
42       {
43          message = value;
44       } // end method setMessage
45    } // end class TextMessage
```

Fig. 31.15 | REST web service that returns a welcome message as JSON. (Part 2 of 2.)

31.8.2 Consuming a REST-Based JSON Web Service

- `URL` method `openStream` invokes the web service and obtains an `InputStream` from which the client can read the response.
- We wrap the `InputStream` in an `InputStreamReader` so it can be passed as the first argument to the `Gson` class's `fromJson` method.
  - The method we use takes as arguments a `Reader` from which to read a JSON `String` and a `Class<T>` object indicating the Java class to which the JSON `String` will be converted.

```
1  // Fig. 31.16: WelcomeRESTJSONClientJFrame.java
2  // Client that consumes the WelcomeRESTJSON service.
3  package com.deitel.welcomerestjsonclient;
4
5  import com.google.gson.Gson; // converts POJO to JSON and back again
6  import java.io.InputStreamReader;
7  import java.net.URL;
8  import javax.swing.JOptionPane;
9
10 public class WelcomeRESTJSONClientJFrame extends javax.swing.JFrame
11 {
12    // no-argument constructor
13    public WelcomeRESTJSONClientJFrame()
14    {
15       initComponents();
16    } // end constructor
17
```

Fig. 31.16 | Client that consumes the `WelcomeRESTJSON` service. (Part 1 of 5.)

```
18      // The initComponents method is autogenerated by NetBeans and is called
19      // from the constructor to initialize the GUI. This method is not shown
20      // here to save space. Open WelcomeRESTJSONClientJFrame.java in this
21      // example's folder to view the complete generated code.
22
23      // call the web service with the supplied name and display the message
24      private void submitJButtonActionPerformed(
25         java.awt.event.ActionEvent evt )
26      {
27         String name = nameJTextField.getText(); // get name from JTextField
28
```

Fig. 31.16 | Client that consumes the `WelcomeRESTJSON` service. (Part 2 of 5.)

```
29          // retrieve the welcome string from the web service
30          try
31          {
32             // the URL of the web service
33             String url = "http://localhost:8080/WelcomeRESTJSON/" +
34                "resources/welcome/" + name;
35
36             // open URL, using a Reader to convert bytes to chars
37             InputStreamReader reader =
38                new InputStreamReader( new URL( url ).openStream() );
39
40             // parse the JSON back into a TextMessage
41             TextMessage message =
42                new Gson().fromJson( reader, TextMessage.class );
43
44             // display message to the user
45             JOptionPane.showMessageDialog( this, message.getMessage(),
46                "Welcome", JOptionPane.INFORMATION_MESSAGE );
47          } // end try
48          catch ( Exception exception )
49          {
50             exception.printStackTrace(); // show exception details
51          } // end catch
52       } // end method submitJButtonActionPerformed
```

Fig. 31.16 | Client that consumes the `WelcomeRESTJSON` service. (Part 3 of 5.)

```
53
54      // main method begin execution
55      public static void main( String args[] )
56      {
57         java.awt.EventQueue.invokeLater(
58            new Runnable()
59            {
60               public void run()
61               {
62                  new WelcomeRESTJSONClientJFrame().setVisible( true );
63               } // end method run
64            } // end anonymous inner class
65         ); // end call to java.awt.EventQueue.invokeLater
66      } // end main
67
68      // Variables declaration - do not modify
69      private javax.swing.JLabel nameJLabel;
70      private javax.swing.JTextField nameJTextField;
71      private javax.swing.JButton submitJButton;
72      // End of variables declaration
73   } // end class WelcomeRESTJSONClientJFrame
74
```

Fig. 31.16 | Client that consumes the `WelcomeRESTJSON` service. (Part 4 of 5.)

```
75  // private class that contains the message we are receiving
76  class TextMessage
77  {
78     private String message; // message we're receiving
79
80     // returns the message
81     public String getMessage()
82     {
83        return message;
84     } // end method getMessage
85
86     // sets the message
87     public void setMessage( String value )
88     {
89        message = value;
90     } // end method setMessage
91  } // end class TextMessage
```

Fig. 31.16 | Client that consumes the `WelcomeRESTJSON` service. (Part 5 of 5.)

31.9 Session Tracking in a SOAP-Based Web Service

- Section 29.8 described the advantages of using session tracking to maintain client-state information so you can personalize the users' browsing experiences.
- Now we'll incorporate session tracking into a web service.
- Storing session information also enables a web service to distinguish between clients.

31.9.1 Creating a `Blackjack` Web Service

- Our next example is a web service that assists you in developing a blackjack card game.
- The `Blackjack` web service (Fig. 31.17) provides web methods to shuffle a deck of cards, deal a card from the deck and evaluate a hand of cards.
- The web service (Fig. 31.17) stores each card as a `String` consisting of a number, `1`–`13`, representing the card's face (ace through king, respectively), followed by a space and a digit, `0`–`3`, representing the card's suit (hearts, diamonds, clubs or spades, respectively).
- For example, the jack of clubs is represented as `"11 2"` and the two of hearts as `"2 0"`.
- To create and deploy this web service, follow the steps that we presented in Sections 31.6.2–31.6.3 for the `WelcomeSOAP` service.

```
1  // Fig. 31.17: Blackjack.java
2  // Blackjack web service that deals cards and evaluates hands
3  package com.deitel.blackjack;
4
5  import com.sun.xml.ws.developer.servlet.HttpSessionScope;
6  import java.util.ArrayList;
7  import java.util.Random;
8  import javax.jws.WebMethod;
9  import javax.jws.WebParam;
10 import javax.jws.WebService;
11
12 @HttpSessionScope // enable web service to maintain session state
13 @WebService()
14 public class Blackjack
15 {
16    private ArrayList< String > deck; // deck of cards for one user session
17    private static final Random randomObject = new Random();
18
```

Fig. 31.17 | `Blackjack` web service that deals cards and evaluates hands. (Part 1 of 6.)

```
19      // deal one card
20      @WebMethod( operationName = "dealCard" )
21      public String dealCard()
22      {
23         String card = "";
24         card = deck.get( 0 ); // get top card of deck
25         deck.remove( 0 ); // remove top card of deck
26         return card;
27      } // end WebMethod dealCard
28
29      // shuffle the deck
30      @WebMethod( operationName = "shuffle" )
31      public void shuffle()
32      {
33         // create new deck when shuffle is called
34         deck = new ArrayList< String >();
35
```

Fig. 31.17 | `Blackjack` web service that deals cards and evaluates hands. (Part 2 of 6.)

```
36          // populate deck of cards
37          for ( int face = 1; face <= 13; face++ ) // loop through faces
38             for ( int suit = 0; suit <= 3; suit++ ) // loop through suits
39                deck.add( face + " " + suit ); // add each card to deck
40
41          String tempCard; // holds card temporarily during swapping
42          int index; // index of randomly selected card
43
44          for ( int i = 0; i < deck.size() ; i++ ) // shuffle
45          {
46             index = randomObject.nextInt( deck.size() - 1 );
47
48             // swap card at position i with randomly selected card
49             tempCard = deck.get( i );
50             deck.set( i, deck.get( index ) );
51             deck.set( index, tempCard );
52          } // end for
53       } // end WebMethod shuffle
54
```

Fig. 31.17 | `Blackjack` web service that deals cards and evaluates hands. (Part 3 of 6.)

```
55      // determine a hand's value
56      @WebMethod( operationName = "getHandValue" )
57      public int getHandValue( @WebParam( name = "hand" ) String hand )
58      {
59         // split hand into cards
60         String[] cards = hand.split( "\t" );
61         int total = 0; // total value of cards in hand
62         int face; // face of current card
63         int aceCount = 0; // number of aces in hand
64
65         for ( int i = 0; i < cards.length; i++ )
66         {
67            // parse string and get first int in String
68            face = Integer.parseInt(
69               cards[ i ].substring( 0, cards[ i ].indexOf( " " ) ) );
70
```

Fig. 31.17 | `Blackjack` web service that deals cards and evaluates hands. (Part 4 of 6.)

```
71            switch ( face )
72            {
73               case 1: // if ace, increment aceCount
74                  ++aceCount;
75                  break;
76               case 11: // jack
77               case 12: // queen
78               case 13: // king
79                  total += 10;
80                  break;
81               default: // otherwise, add face
82                  total += face;
83                  break;
84            } // end switch
85         } // end for
86
```

Fig. 31.17 | `Blackjack` web service that deals cards and evaluates hands. (Part 5 of 6.)

```
87          // calculate optimal use of aces
88          if ( aceCount > 0 )
89          {
90             // if possible, count one ace as 11
91             if ( total + 11 + aceCount - 1 <= 21 )
92                total += 11 + aceCount - 1;
93             else // otherwise, count all aces as 1
94                total += aceCount;
95           } // end if
96
97          return total;
98       } // end WebMethod getHandValue
99    } // end class Blackjack
```

Fig. 31.17 | `Blackjack` web service that deals cards and evaluates hands. (Part 6 of 6.)

31.9.1 Creating a `Blackjack` Web Service (cont.)

- To enable session tracking in a web service in JAX-WS 2.2, precede your web service class with the `@HttpSessionScope-` annotation.
  - Annotation is located in package `com.sun.xml.ws.developer.servlet`.
- Add the JAX-WS 2.2 library to your project.
  - Right click the Libraries node in your `Blackjack` web application project and select Add Library…
  - In the dialog that appears, locate and select JAX-WS 2.2, then click Add Library.
  - Once a web service is annotated with `@HttpSessionScope`, the server automatically maintains a separate instance of the class for each client session. The deck instance variable (line 16) will be maintained separately for each client.

31.9.2 Consuming the Blackjack Web Service

- The blackjack application in Fig. 31.18 keeps track of the player's and dealer's cards, and the web service tracks the cards that have been dealt.

```
1  // Fig. 31.18: BlackjackGameJFrame.java
2  // Blackjack game that uses the Blackjack Web Service.
3  package com.deitel.blackjackclient;
4
5  import com.deitel.blackjack.Blackjack;
6  import com.deitel.blackjack.BlackjackService;
7  import java.awt.Color;
8  import java.util.ArrayList;
9  import javax.swing.ImageIcon;
10 import javax.swing.JLabel;
11 import javax.swing.JOptionPane;
12 import javax.xml.ws.BindingProvider;
13
14 public class BlackjackGameJFrame extends javax.swing.JFrame
15 {
16    private String playerCards;
17    private String dealerCards;
18    private ArrayList<JLabel> cardboxes; // list of card image JLabels
19    private int currentPlayerCard; // player's current card number
20    private int currentDealerCard; // blackjackProxy's current card number
21    private BlackjackService blackjackService; // used to obtain proxy
22    private Blackjack blackjackProxy; // used to access the web service
23
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 1 of 24.)

```
24      // enumeration of game states
25      private enum GameStatus
26      {
27         PUSH, // game ends in a tie
28         LOSE, // player loses
29         WIN, // player wins
30         BLACKJACK // player has blackjack
31      } // end enum GameStatus
32
33      // no-argument constructor
34      public BlackjackGameJFrame()
35      {
36         initComponents();
37
38         // due to a bug in NetBeans, we must change the JFrame's background
39         // color here rather than in the designer
40         getContentPane().setBackground( new Color( 0, 180, 0 ) );
41
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 2 of 24.)

```
42          // initialize the blackjack proxy
43          try
44          {
45             // create the objects for accessing the Blackjack web service
46             blackjackService = new BlackjackService();
47             blackjackProxy = blackjackService.getBlackjackPort();
48
49             // enable session tracking
50             ( (BindingProvider) blackjackProxy ).getRequestContext().put(
51                BindingProvider.SESSION_MAINTAIN_PROPERTY, true );
52          } // end try
53          catch ( Exception e )
54          {
55             e.printStackTrace();
56          } // end catch
57
58          // add JLabels to cardBoxes ArrayList for programmatic manipulation
59          cardboxes = new ArrayList<JLabel>();
60
61          cardboxes.add( dealerCard1JLabel );
62          cardboxes.add( dealerCard2JLabel );
63          cardboxes.add( dealerCard3JLabel );
64          cardboxes.add( dealerCard4JLabel );
65          cardboxes.add( dealerCard5JLabel );
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 3 of 24.)

```
66          cardboxes.add( dealerCard6JLabel );
67          cardboxes.add( dealerCard7JLabel );
68          cardboxes.add( dealerCard8JLabel );
69          cardboxes.add( dealerCard9JLabel );
70          cardboxes.add( dealerCard10JLabel );
71          cardboxes.add( dealerCard11JLabel );
72          cardboxes.add( playerCard1JLabel );
73          cardboxes.add( playerCard2JLabel );
74          cardboxes.add( playerCard3JLabel );
75          cardboxes.add( playerCard4JLabel );
76          cardboxes.add( playerCard5JLabel );
77          cardboxes.add( playerCard6JLabel );
78          cardboxes.add( playerCard7JLabel );
79          cardboxes.add( playerCard8JLabel );
80          cardboxes.add( playerCard9JLabel );
81          cardboxes.add( playerCard10JLabel );
82          cardboxes.add( playerCard11JLabel );
83       } // end constructor
84
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 4 of 24.)

```
85      // play the dealer's hand
86      private void dealerPlay()
87      {
88         try
89         {
90            // while the value of the dealers's hand is below 17
91            // the dealer must continue to take cards
92            String[] cards = dealerCards.split( "\t" );
93
94            // display dealer's cards
95            for ( int i = 0; i < cards.length; i++ )
96            {
97               displayCard( i, cards[i] );
98            }
99
100           while ( blackjackProxy.getHandValue( dealerCards ) < 17 )
101           {
102              String newCard = blackjackProxy.dealCard(); // deal new card
103              dealerCards += "\t" + newCard; // deal new card
104              displayCard( currentDealerCard, newCard );
105              ++currentDealerCard;
106              JOptionPane.showMessageDialog( this, "Dealer takes a card",
107                 "Dealer's turn", JOptionPane.PLAIN_MESSAGE );
108           } // end while
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 5 of 24.)

```
109
110          int dealersTotal = blackjackProxy.getHandValue( dealerCards );
111          int playersTotal = blackjackProxy.getHandValue( playerCards );
112
113          // if dealer busted, player wins
114          if ( dealersTotal > 21 )
115          {
116             gameOver( GameStatus.WIN );
117             return;
118          } // end if
119
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 6 of 24.)

```
120          // if dealer and player are below 21
121          // higher score wins, equal scores is a push
122          if ( dealersTotal > playersTotal )
123          {
124             gameOver( GameStatus.LOSE );
125          }
126          else if ( dealersTotal < playersTotal )
127          {
128             gameOver( GameStatus.WIN );
129          }
130          else
131          {
132             gameOver( GameStatus.PUSH );
133          }
134       } // end try
135       catch ( Exception e )
136       {
137          e.printStackTrace();
138       } // end catch
139    } // end method dealerPlay
140
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 7 of 24.)

```
141     // displays the card represented by cardValue in specified JLabel
142     private void displayCard( int card, String cardValue )
143     {
144        try
145        {
146           // retrieve correct JLabel from cardBoxes
147           JLabel displayLabel = cardboxes.get( card );
148
149           // if string representing card is empty, display back of card
150           if ( cardValue.equals( "" ) )
151           {
152              displayLabel.setIcon( new ImageIcon( getClass().getResource(
153                 "/com/deitel/java/blackjackclient/" +
154                 "blackjack_images/cardback.png" ) ) );
155              return;
156           } // end if
157
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 8 of 24.)

```
158          // retrieve the face value of the card
159          String face = cardValue.substring( 0, cardValue.indexOf( " " ) );
160
161          // retrieve the suit of the card
162          String suit =
163             cardValue.substring( cardValue.indexOf( " " ) + 1 );
164
165          char suitLetter; // suit letter used to form image file
166
167          switch ( Integer.parseInt( suit ) )
168          {
169             case 0: // hearts
170                suitLetter = 'h';
171                break;
172             case 1: // diamonds
173                suitLetter = 'd';
174                break;
175             case 2: // clubs
176                suitLetter = 'c';
177                break;
178             default: // spades
179                suitLetter = 's';
180                break;
181          } // end switch
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 9 of 24.)

```
182
183           // set image for displayLabel
184           displayLabel.setIcon( new ImageIcon( getClass().getResource(
185              "/com/deitel/java/blackjackclient/blackjack_images/" +
186              face + suitLetter + ".png" ) ) );
187        } // end try
188        catch ( Exception e )
189        {
190           e.printStackTrace();
191        } // end catch
192     } // end method displayCard
193
194     // displays all player cards and shows appropriate message
195     private void gameOver( GameStatus winner )
196     {
197        String[] cards = dealerCards.split( "\t" );
198
199        // display blackjackProxy's cards
200        for ( int i = 0; i < cards.length; i++ )
201        {
202           displayCard( i, cards[i] );
203        }
204
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 10 of 24.)

```
205         // display appropriate status image
206         if ( winner == GameStatus.WIN )
207         {
208            statusJLabel.setText( "You win!" );
209         }
210         else if ( winner == GameStatus.LOSE )
211         {
212            statusJLabel.setText( "You lose." );
213         }
214         else if ( winner == GameStatus.PUSH )
215         {
216            statusJLabel.setText( "It's a push." );
217         }
218         else // blackjack
219         {
220            statusJLabel.setText( "Blackjack!" );
221         }
222
223         // display final scores
224         int dealersTotal = blackjackProxy.getHandValue( dealerCards );
225         int playersTotal = blackjackProxy.getHandValue( playerCards );
226         dealerTotalJLabel.setText( "Dealer: " + dealersTotal );
227         playerTotalJLabel.setText( "Player: " + playersTotal );
228
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 11 of 24.)

```
229         // reset for new game
230         standJButton.setEnabled( false );
231         hitJButton.setEnabled( false );
232         dealJButton.setEnabled( true );
233      } // end method gameOver
234
235      // The initComponents method is autogenerated by NetBeans and is called
236      // from the constructor to initialize the GUI. This method is not shown
237      // here to save space. Open BlackjackGameJFrame.java in this
238      // example's folder to view the complete generated code
239
240      // handles dealJButton click
241      private void dealJButtonActionPerformed(
242         java.awt.event.ActionEvent evt )
243      {
244         String card; // stores a card temporarily until it's added to a hand
245
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 12 of 24.)

```
246         // clear card images
247         for ( int i = 0; i < cardboxes.size(); i++ )
248         {
249            cardboxes.get( i ).setIcon( null );
250         }
251
252         statusJLabel.setText( "" );
253         dealerTotalJLabel.setText( "" );
254         playerTotalJLabel.setText( "" );
255
256         // create a new, shuffled deck on remote machine
257         blackjackProxy.shuffle();
258
259         // deal two cards to player
260         playerCards = blackjackProxy.dealCard(); // add first card to hand
261         displayCard( 11, playerCards ); // display first card
262         card = blackjackProxy.dealCard(); // deal second card
263         displayCard( 12, card ); // display second card
264         playerCards += "\t" + card; // add second card to hand
265
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 13 of 24.)

```
266         // deal two cards to blackjackProxy, but only show first
267         dealerCards = blackjackProxy.dealCard(); // add first card to hand
268         displayCard( 0, dealerCards ); // display first card
269         card = blackjackProxy.dealCard(); // deal second card
270         displayCard( 1, "" ); // display back of card
271         dealerCards += "\t" + card; // add second card to hand
272
273         standJButton.setEnabled( true );
274         hitJButton.setEnabled( true );
275         dealJButton.setEnabled( false );
276
277         // determine the value of the two hands
278         int dealersTotal = blackjackProxy.getHandValue( dealerCards );
279         int playersTotal = blackjackProxy.getHandValue( playerCards );
280
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 14 of 24.)

```
281         // if hands both equal 21, it is a push
282         if ( playersTotal == dealersTotal && playersTotal == 21 )
283         {
284            gameOver( GameStatus.PUSH );
285         }
286         else if ( dealersTotal == 21 ) // blackjackProxy has blackjack
287         {
288            gameOver( GameStatus.LOSE );
289         }
290         else if ( playersTotal == 21 ) // blackjack
291         {
292            gameOver( GameStatus.BLACKJACK );
293         }
294
295         // next card for blackjackProxy has index 2
296         currentDealerCard = 2;
297
298         // next card for player has index 13
299         currentPlayerCard = 13;
300      } // end method dealJButtonActionPerformed
301
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 15 of 24.)

```
302     // handles standJButton click
303     private void hitJButtonActionPerformed(
304        java.awt.event.ActionEvent evt )
305     {
306        // get player another card
307        String card = blackjackProxy.dealCard(); // deal new card
308        playerCards += "\t" + card; // add card to hand
309
310        // update GUI to display new card
311        displayCard( currentPlayerCard, card );
312        ++currentPlayerCard;
313
314        // determine new value of player's hand
315        int total = blackjackProxy.getHandValue( playerCards );
316
317        if ( total > 21 ) // player busts
318        {
319           gameOver( GameStatus.LOSE );
320        }
321        else if ( total == 21 ) // player cannot take any more cards
322        {
323           hitJButton.setEnabled( false );
324           dealerPlay();
325        } // end if
326     } // end method hitJButtonActionPerformed
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 16 of 24.)

```
327
328    // handles standJButton click
329    private void standJButtonActionPerformed(
330       java.awt.event.ActionEvent evt )
331    {
332       standJButton.setEnabled( false );
333       hitJButton.setEnabled( false );
334       dealJButton.setEnabled( true );
335       dealerPlay();
336    } // end method standJButtonActionPerformed
337
338    // begins application execution
339    public static void main( String args[] )
340    {
341       java.awt.EventQueue.invokeLater(
342          new Runnable()
343          {
344             public void run()
345             {
346                new BlackjackGameJFrame().setVisible( true );
347             }
348          }
349       ); // end call to java.awt.EventQueue.invokeLater
350    } // end main
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 17 of 24.)

```
351
352     // Variables declaration - do not modify
353     private javax.swing.JButton dealJButton;
354     private javax.swing.JLabel dealerCard10JLabel;
355     private javax.swing.JLabel dealerCard11JLabel;
356     private javax.swing.JLabel dealerCard1JLabel;
357     private javax.swing.JLabel dealerCard2JLabel;
358     private javax.swing.JLabel dealerCard3JLabel;
359     private javax.swing.JLabel dealerCard4JLabel;
360     private javax.swing.JLabel dealerCard5JLabel;
361     private javax.swing.JLabel dealerCard6JLabel;
362     private javax.swing.JLabel dealerCard7JLabel;
363     private javax.swing.JLabel dealerCard8JLabel;
364     private javax.swing.JLabel dealerCard9JLabel;
365     private javax.swing.JLabel dealerJLabel;
366     private javax.swing.JLabel dealerTotalJLabel;
367     private javax.swing.JButton hitJButton;
368     private javax.swing.JLabel playerCard10JLabel;
369     private javax.swing.JLabel playerCard11JLabel;
370     private javax.swing.JLabel playerCard1JLabel;
371     private javax.swing.JLabel playerCard2JLabel;
372     private javax.swing.JLabel playerCard3JLabel;
373     private javax.swing.JLabel playerCard4JLabel;
374     private javax.swing.JLabel playerCard5JLabel;
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 18 of 24.)

```
375     private javax.swing.JLabel playerCard6JLabel;
376     private javax.swing.JLabel playerCard7JLabel;
377     private javax.swing.JLabel playerCard8JLabel;
378     private javax.swing.JLabel playerCard9JLabel;
379     private javax.swing.JLabel playerJLabel;
380     private javax.swing.JLabel playerTotalJLabel;
381     private javax.swing.JButton standJButton;
382     private javax.swing.JLabel statusJLabel;
383     // End of variables declaration
384 } // end class BlackjackGameJFrame
```

Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 19 of 24.)

a) Dealer and player hands after the user clicks the **Deal** `JButton`

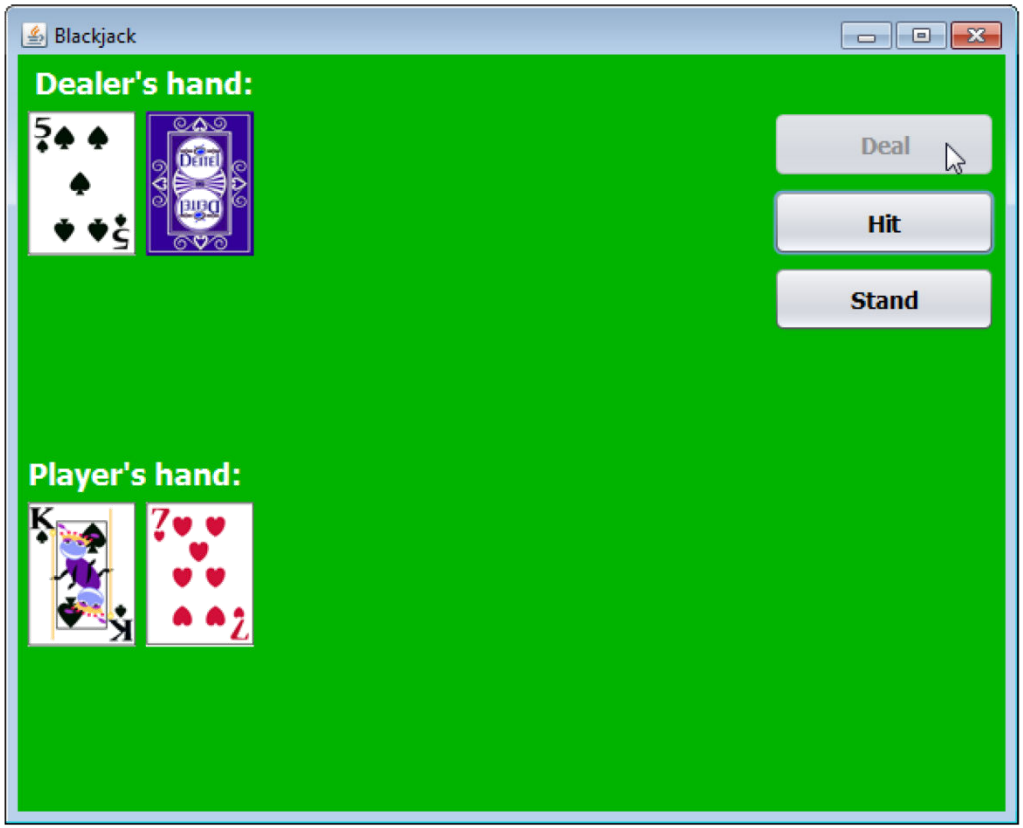


Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 20 of 24.)

b) Dealer and player hands after the user clicks **Stand**. In this case, the result is a push

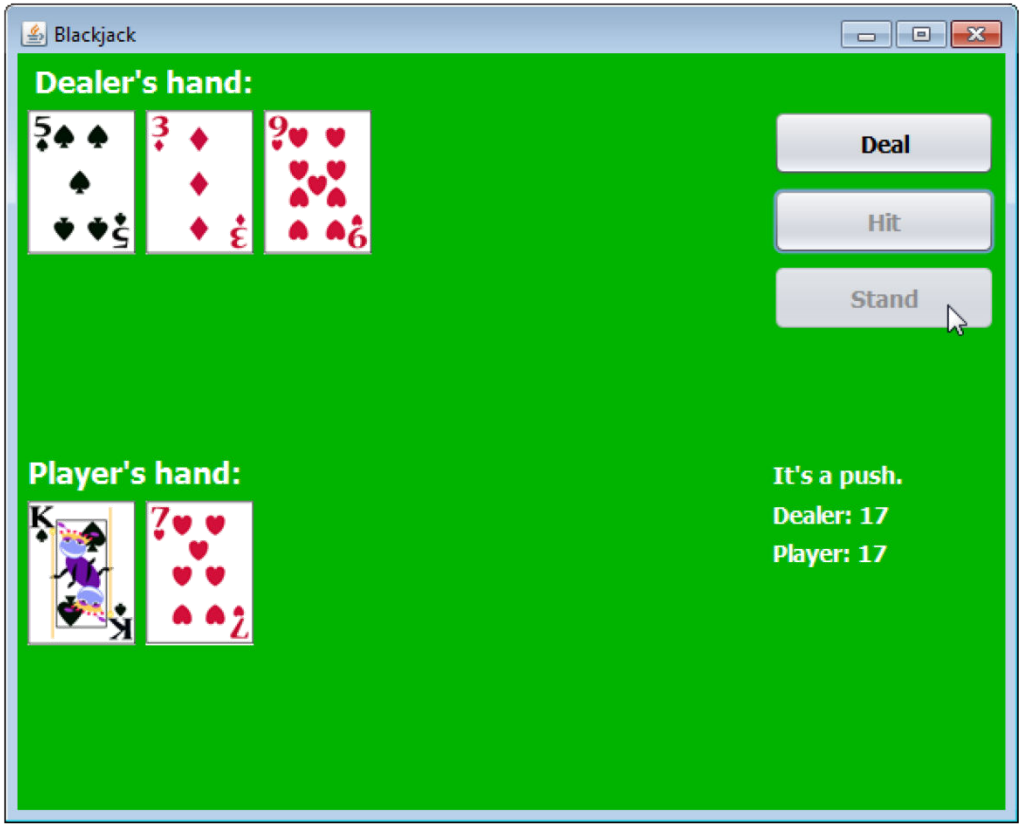


Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 21 of 24.)

c) Dealer and player hands after the user clicks **Hit** and draws 21. In this case, the player wins

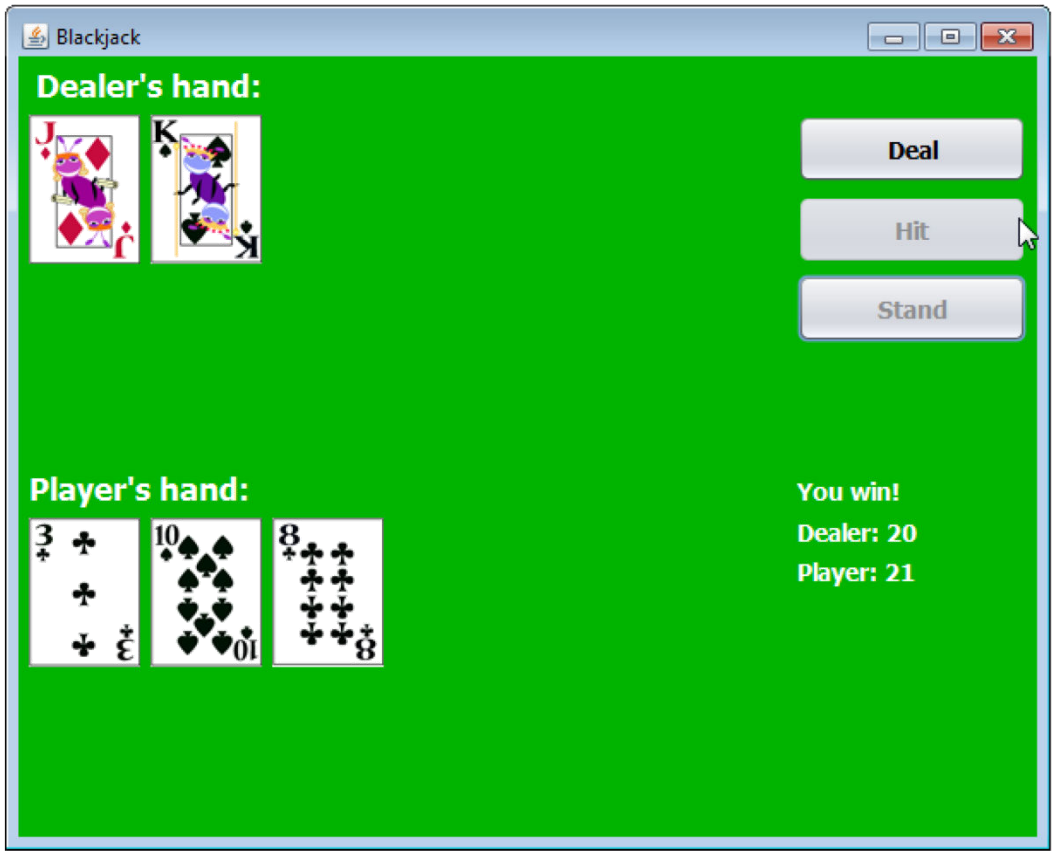


Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 22 of 24.)

d) Dealer and player hands after the player is dealt blackjack

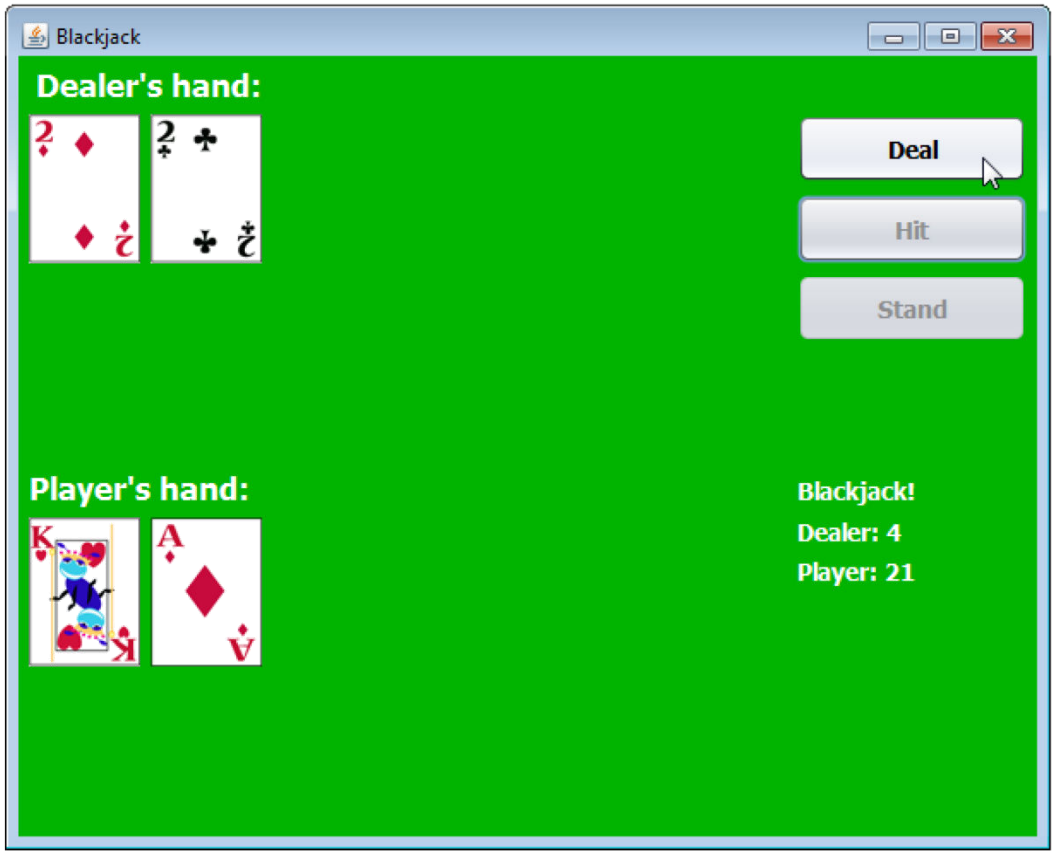


Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 23 of 24.)

e) Dealer and player hands after the dealer is dealt blackjack

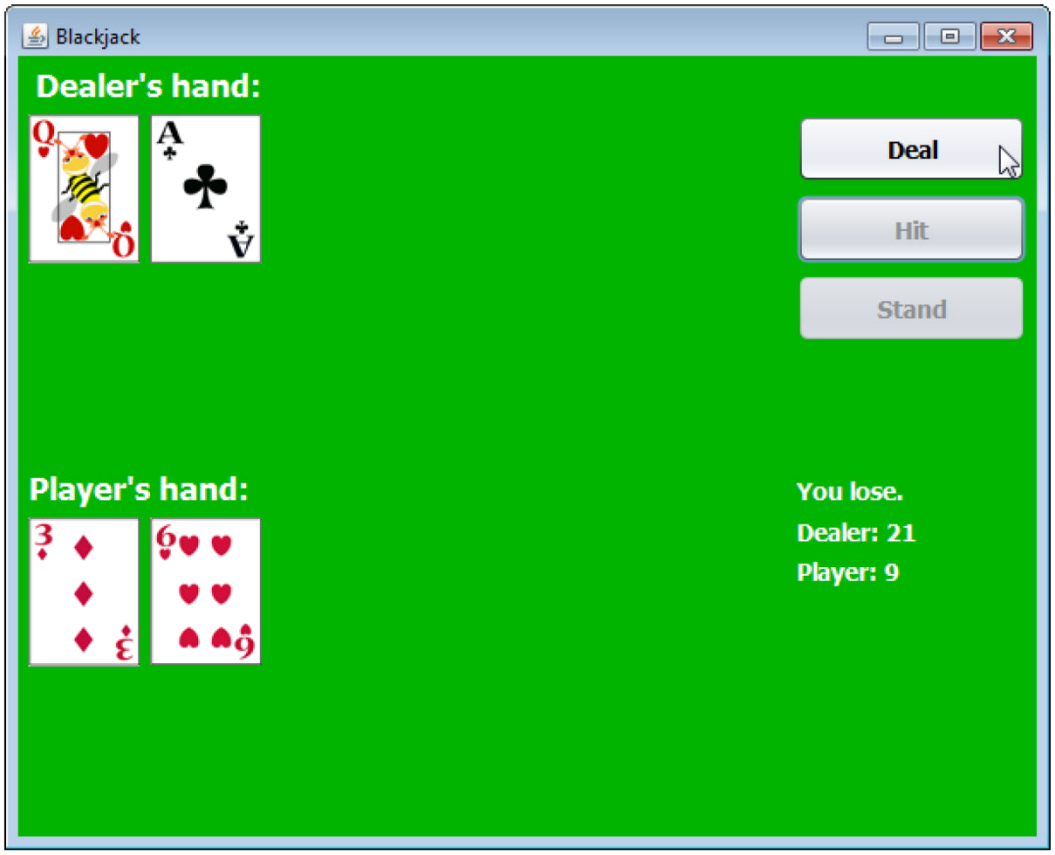


Fig. 31.18 | Blackjack game that uses the `Blackjack` web service. (Part 24 of 24.)

31.9.2 Consuming the `Blackjack` Web Service (cont.)

- When interacting with a JAX-WS web service that performs session tracking, the client application must indicate whether it wants to allow the web service to maintain session information.
- We first cast the service endpoint interface object to interface type `BindingProvider`.
- A `BindingProvider` enables the client to manipulate the request information that will be sent to the server.
- This information is stored in an object that implements interface `RequestContext`.
- The `BindingProvider` and `RequestContext` are part of the framework that is created by the IDE when you add a web service client to the application.

31.9.2 Consuming the `Blackjack` Web Service (cont.)

- Next, we invoke the `BindingProvider`'s `getRequestContext` method to obtain the `RequestContext` object.
- Then we call the `RequestContext`'s `put` method to set the property
  `BindingProvider.SESSION_MAINTAIN_PROPERTY` to `true`.
- This enables the client side of the session-tracking mechanism, so that the web service knows which client is invoking the service's web methods.

31.10 Consuming a Database-Driven SOAP-Based Web Service

- In this section, we present an airline reservation web service that receives information regarding the type of seat a customer wishes to reserve and makes a reservation if such a seat is available.
- Later in the section, we present a web application that allows a customer to specify a reservation request, then uses the airline reservation web service to attempt to execute the request.

31.10.1 Creating the `Reservation` Database

- Our web service uses a `reservation` database containing a single table named `Seats` to locate a seat matching a client's request.
- The sample data is shown in Fig. 31.19.

number	location	class	taken
1	Aisle	Economy	0
2	Aisle	Economy	0
3	Aisle	First	0
4	Middle	Economy	0
5	Middle	Economy	0
6	Middle	First	0
7	Window	Economy	0
8	Window	Economy	0
9	Window	First	0
10	Window	First	0

Fig. 31.19 | Data from the `seats` table.

Software Engineering Observation 31.1

Using `PreparedStatement`*s to create SQL statements is highly recommended to secure against so-called SQL injection attacks in which executable code is inserted into SQL code. The site* `www.owasp.org/index.php/Preventing_SQL_Injection_in_Java` *provides a summary of SQL injection attacks and ways to mitigate against them.*

```
1  // Fig. 31.20: Reservation.java
2  // Airline reservation web service.
3  package com.deitel.reservation;
4
5  import java.sql.Connection;
6  import java.sql.PreparedStatement;
7  import java.sql.ResultSet;
8  import java.sql.SQLException;
9  import javax.annotation.Resource;
10 import javax.jws.WebMethod;
11 import javax.jws.WebParam;
12 import javax.jws.WebService;
13 import javax.sql.DataSource;
14
```

Fig. 31.20 | Airline reservation web service. (Part 1 of 5.)

```
15  @WebService()
16  public class Reservation
17  {
18     // allow the server to inject the DataSource
19     @Resource( name="jdbc/reservation" )
20     DataSource dataSource;
21
22     // a WebMethod that can reserve a seat
23     @WebMethod( operationName = "reserve" )
24     public boolean reserve( @WebParam( name = "seatType" ) String seatType,
25        @WebParam( name = "classType" ) String classType )
26     {
27        Connection connection = null;
28        PreparedStatement lookupSeat = null;
29        PreparedStatement reserveSeat = null;
30
```

Fig. 31.20 | Airline reservation web service. (Part 2 of 5.)

```
31          try
32          {
33             connection = DriverManager.getConnection(
34                DATABASE_URL, USERNAME, PASSWORD );
35             lookupSeat = connection.prepareStatement(
36                "SELECT \"number\" FROM \"seats\" WHERE (\"taken\" = 0) " +
37                "AND (\"location\" = ?) AND (\"class\" = ?)" );
38             lookupSeat.setString( 1, seatType );
39             lookupSeat.setString( 2, classType );
40             ResultSet resultSet = lookupSeat.executeQuery();
41
42             // if requested seat is available, reserve it
43             if ( resultSet.next() )
44             {
45                int seat = resultSet.getInt( 1 );
46                reserveSeat = connection.prepareStatement(
47                   "UPDATE \"seats\" SET \"taken\"=1 WHERE \"number\"=?" );
48                reserveSeat.setInt( 1, seat );
49                reserveSeat.executeUpdate();
50                return true;
51             } // end if
52
53             return false;
54          } // end try
```

Fig. 31.20 | Airline reservation web service. (Part 3 of 5.)

```
55          catch ( SQLException e )
56          {
57             e.printStackTrace();
58             return false;
59          } // end catch
60          catch ( Exception e )
61          {
62             e.printStackTrace();
63             return false;
64          } // end catch
65          finally
66          {
67             try
68             {
69                lookupSeat.close();
70                reserveSeat.close();
71                connection.close();
72             } // end try
73             catch ( Exception e )
74             {
75                e.printStackTrace();
76                return false;
77             } // end catch
78          } // end finally
```

Fig. 31.20 | Airline reservation web service. (Part 4 of 5.)

```
79      } // end WebMethod reserve
80  } // end class Reservation
```

Fig. 31.20 | Airline reservation web service. (Part 5 of 5.)

31.10.2 Creating a Web Application to Interact with the `Reservation` Service

- This section presents a `ReservationClient` JSFweb application that consumes the `Reservation` web service.
- The application allows users to select `"Aisle"`, `"Middle"` or `"Window"` seats in `"Economy"` or `"First"` class, then submit their requests to the airline reservation web service.
- If the database request is not successful, the application instructs the user to modify the request and try again.

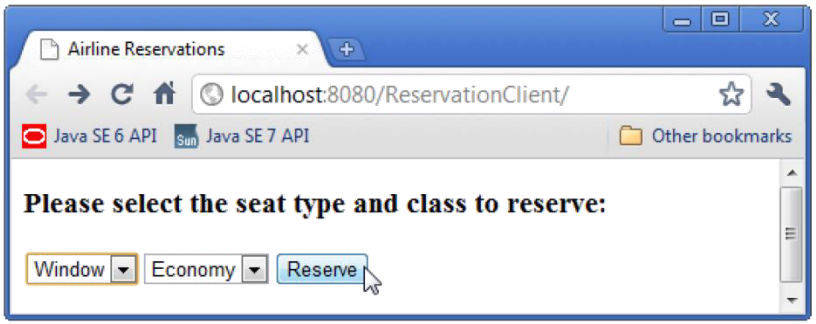
```
1  <?xml version='1.0' encoding='UTF-8' ?>
2
3  <!-- Fig. 31.21: index.xhtml -->
4  <!-- Facelets page that allows a user to select a seat -->
5  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
6     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
7  <html xmlns="http://www.w3.org/1999/xhtml"
8     xmlns:h="http://java.sun.com/jsf/html"
9     xmlns:f="http://java.sun.com/jsf/core">
10    <h:head>
11       <title>Airline Reservations</title>
12    </h:head>
13    <h:body>
14       <h:form>
15          <h3>Please select the seat type and class to reserve:</h3>
16          <h:selectOneMenu value="#{reservationBean.seatType}">
17             <f:selectItem itemValue="Aisle" itemLabel="Aisle" />
18             <f:selectItem itemValue="Middle" itemLabel="Middle" />
19             <f:selectItem itemValue="Window" itemLabel="Window" />
20          </h:selectOneMenu>
21          <h:selectOneMenu value="#{reservationBean.classType}">
22             <f:selectItem itemValue="Economy" itemLabel="Economy" />
23             <f:selectItem itemValue="First" itemLabel="First" />
24          </h:selectOneMenu>
```

Fig. 31.21 | Facelets page that allows a user to select a seat. (Part 1 of 4.)

```
25            <h:commandButton value="Reserve"
26               action="#{reservationBean.reserveSeat}"/>
27         </h:form>
28         <h3>#{reservationBean.result}</h3>
29      </h:body>
30   </html>
```

Fig. 31.21 | Facelets page that allows a user to select a seat. (Part 2 of 4.)

a) Selecting a seat



b) Seat reserved successfully

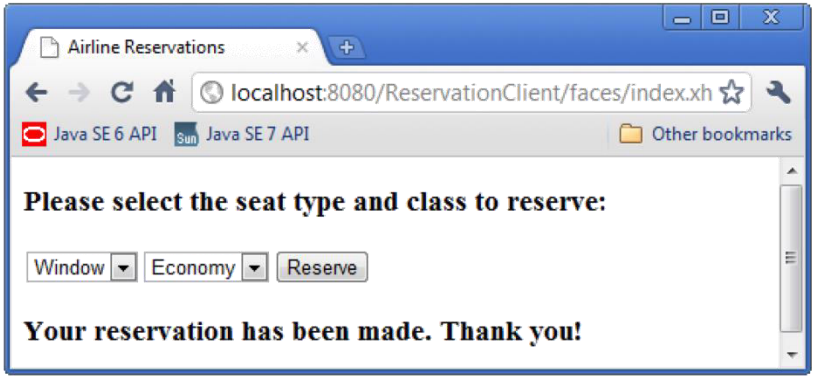
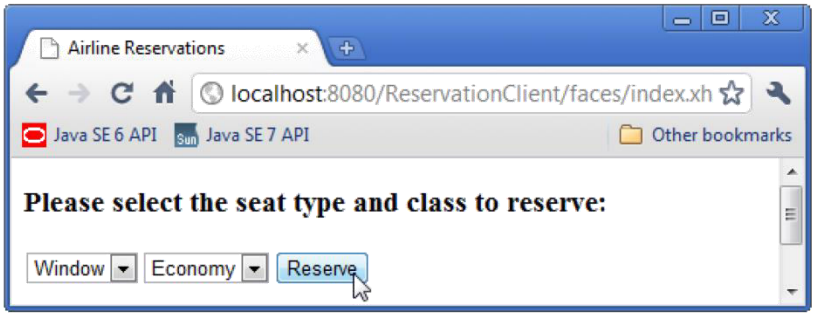


Fig. 31.21 | Facelets page that allows a user to select a seat. (Part 3 of 4.)

c) Attempting to reserve another window seat in economy when there are no such seats available



d) No seats match the requested seat type and class

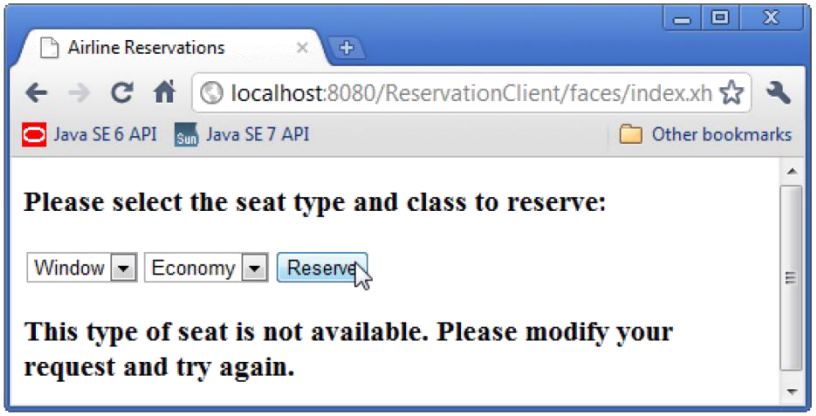


Fig. 31.21 | Facelets page that allows a user to select a seat. (Part 4 of 4.)

```
1  // Fig. 31.22: ReservationBean.java
2  // Bean for seat reservation client.
3  package reservationclient;
4
5  import com.deitel.reservation.Reservation;
6  import com.deitel.reservation.ReservationService;
7  import javax.faces.bean.ManagedBean;
8
9  @ManagedBean( name = "reservationBean" )
10 public class ReservationBean
11 {
12    // references the service endpoint interface object (i.e., the proxy)
13    private Reservation reservationServiceProxy; // reference to proxy
14    private String seatType; // type of seat to reserve
15    private String classType; // class of seat to reserve
16    private String result; // result of reservation attempt
17
```

Fig. 31.22 | Page bean for seat reservation client. (Part 1 of 4.)

```
18      // no-argument constructor
19      public ReservationBean()
20      {
21         // get service endpoint interface
22         ReservationService reservationService = new ReservationService();
23         reservationServiceProxy = reservationService.getReservationPort();
24      } // end constructor
25
26      // return classType
27      public String getClassType()
28      {
29         return classType;
30      } // end method getClassType
31
32      // set classType
33      public void setClassType( String classType )
34      {
35         this.classType = classType;
36      } // end method setClassType
37
```

Fig. 31.22 | Page bean for seat reservation client. (Part 2 of 4.)

```
38      // return seatType
39      public String getSeatType()
40      {
41         return seatType;
42      } // end method getSeatType
43
44      // set seatType
45      public void setSeatType( String seatType )
46      {
47         this.seatType = seatType;
48      } // end method setSeatType
49
50      // return result
51      public String getResult()
52      {
53         return result;
54      } // end method getResult
55
```

Fig. 31.22 | Page bean for seat reservation client. (Part 3 of 4.)

```
56      // invoke the web service when the user clicks Reserve button
57      public void reserveSeat()
58      {
59         try
60         {
61            boolean reserved = reservationServiceProxy.reserve(
62               getSeatType(), getClassType() );
63
64            if ( reserved )
65               result = "Your reservation has been made. Thank you!";
66            else
67               result = "This type of seat is not available. " +
68                  "Please modify your request and try again.";
69         } // end try
70         catch ( Exception e )
71         {
72            e.printStackTrace();
73         } // end catch
74      } // end method reserveSeat
75   } // end class ReservationBean
```

Fig. 31.22 | Page bean for seat reservation client. (Part 4 of 4.)

31.11 Equation Generator: Returning User-Defined Types

- Most of the web services we've demonstrated received and returned primitive-type instances.
- It's also possible to process instances of class types in a web service.
- This section presents a RESTful `EquationGenerator` web service that generates random arithmetic equations of type `Equation`.
- The client is a math-tutoring application that accepts information about the mathematical question that the user wishes to attempt (addition, subtraction or multiplication) and the skill level of the user (1 specifies equations using numbers from 1 through 9, 2 specifies equations involving numbers from 10 through 99, and 3 specifies equations containing numbers from 100 through 999).
- The web service then generates an equation consisting of random numbers in the proper range.
- The client application receives the `Equation` and displays the sample question to the user.

31.11 Equation Generator: Returning User-Defined Types (cont.)

- We define class `Equation` in Fig. 31.23.
- The only requirement for serialization and deserialization to work with the JAXB and Gson classes is that class `Equation` must have the same `public` properties on both the server and the client.
- Such properties can be public instance variables or private instance variables that have corresponding *set* and *get* methods.

```
1  // Fig. 31.23: Equation.java
2  // Equation class that contains information about an equation.
3  package com.deitel.equationgeneratorxml;
4
5  public class Equation
6  {
7     private int leftOperand;
8     private int rightOperand;
9     private int result;
10    private String operationType;
11
12    // required no-argument constructor
13    public Equation()
14    {
15       this( 0, 0, "add" );
16    } // end no-argument constructor
17
```

Fig. 31.23 | `Equation` class that contains information about an equation. (Part 1 of 6.)

```
18      // constructor that receives the operands and operation type
19      public Equation( int leftValue, int rightValue, String type )
20      {
21         leftOperand = leftValue;
22         rightOperand = rightValue;
23
24         // determine result
25         if ( type.equals( "add" ) ) // addition
26         {
27            result = leftOperand + rightOperand;
28            operationType = "+";
29         } // end if
30         else if ( type.equals( "subtract" ) ) // subtraction
31         {
32            result = leftOperand - rightOperand;
33            operationType = "-";
34         } // end if
35         else // multiplication
36         {
37            result = leftOperand * rightOperand;
38            operationType = "*";
39         } // end else
40      } // end three argument constructor
```

Fig. 31.23 | `Equation` class that contains information about an equation. (Part 2 of 6.)

```
41
42      // gets the leftOperand
43      public int getLeftOperand()
44      {
45         return leftOperand;
46      } // end method getLeftOperand
47
48      // required setter
49      public void setLeftOperand( int value )
50      {
51         leftOperand = value;
52      } // end method setLeftOperand
53
54      // gets the rightOperand
55      public int getRightOperand()
56      {
57         return rightOperand;
58      } // end method getRightOperand
59
```

Fig. 31.23 | `Equation` class that contains information about an equation. (Part 3 of 6.)

```
60      // required setter
61      public void setRightOperand( int value )
62      {
63         rightOperand = value;
64      } // end method setRightOperand
65
66      // gets the resultValue
67      public int getResult()
68      {
69         return result;
70      } // end method getResult
71
72      // required setter
73      public void setResult( int value )
74      {
75         result = value;
76      } // end method setResult
77
```

Fig. 31.23 | `Equation` class that contains information about an equation. (Part 4 of 6.)

```
78      // gets the operationType
79      public String getOperationType()
80      {
81         return operationType;
82      } // end method getOperationType
83
84      // required setter
85      public void setOperationType( String value )
86      {
87         operationType = value;
88      } // end method setOperationType
89
90      // returns the left hand side of the equation as a String
91      public String getLeftHandSide()
92      {
93         return leftOperand + " " + operationType + " " + rightOperand;
94      } // end method getLeftHandSide
95
```

Fig. 31.23 | `Equation` class that contains information about an equation. (Part 5 of 6.)

```
96      // returns the right hand side of the equation as a String
97      public String getRightHandSide()
98      {
99         return "" + result;
100     } // end method getRightHandSide
101
102     // returns a String representation of an Equation
103     public String toString()
104     {
105        return getLeftHandSide() + " = " + getRightHandSide();
106     } // end method toString
107  } // end class Equation
```

Fig. 31.23 | `Equation` class that contains information about an equation. (Part 6 of 6.)

31.11.1 Creating the `EquationGeneratorXML` Web Service

- Figure 31.24 presents the `EquationGeneratorXML` web service's class for creating randomly generated `Equation`s.
- Method `getXml` (lines 19–38) takes two parameters—a `String` representing the mathematical operation (`"add"`, `"subtract"` or `"multiply"`) and an `int` representing the difficulty level.
- JAX-RS automatically converts the arguments to the correct type and will return a "not found" error to the client if the argument cannot be converted from a `String` to the destination type.
- Supported types for conversion include integer types, floating-point types, `boolean` and the corresponding type-wrapper classes.

```
1  // Fig. 31.24: EquationGeneratorXMLResource.java
2  // RESTful equation generator that returns XML.
3  package com.deitel.equationgeneratorxml;
4
5  import java.io.StringWriter;
6  import java.util.Random;
7  import javax.ws.rs.PathParam;
8  import javax.ws.rs.Path;
9  import javax.ws.rs.GET;
10 import javax.ws.rs.Produces;
11 import javax.xml.bind.JAXB; // utility class for common JAXB operations
12
13 @Path( "equation" )
14 public class EquationGeneratorXMLResource
15 {
16    private static Random randomObject = new Random();
17
```

Fig. 31.24 | RESTful equation generator that returns XML. (Part 1 of 2.)

```
18      // retrieve an equation formatted as XML
19      @GET
20      @Path( "{operation}/{level}" )
21      @Produces( "application/xml" )
22      public String getXml( @PathParam( "operation" ) String operation,
23         @PathParam( "level" ) int level )
24      {
25         // compute minimum and maximum values for the numbers
26         int minimum = ( int ) Math.pow( 10, level - 1 );
27         int maximum = ( int ) Math.pow( 10, level );
28
29         // create the numbers on the left-hand side of the equation
30         int first = randomObject.nextInt( maximum - minimum ) + minimum;
31         int second = randomObject.nextInt( maximum - minimum ) + minimum;
32
33         // create Equation object and marshal it into XML
34         Equation equation = new Equation( first, second, operation );
35         StringWriter writer = new StringWriter(); // XML output here
36         JAXB.marshal( equation, writer ); // write Equation to StringWriter
37         return writer.toString(); // return XML string
38      } // end method getXml
39   } // end class EquationGeneratorXMLResource
```

Fig. 31.24 | RESTful equation generator that returns XML. (Part 2 of 2.)

31.11.2 Consuming the `EquationGeneratorXML` Web Service

- The `EquationGeneratorXMLClient` application (Fig. 31.24) retrieves an `Equation` object formatted as XML from the `EquationGeneratorXML` web service.
- The client application then displays the left-hand side of the `Equation` and waits for user to evaluate the expression and enter the result.

```
1  // Fig. 31.25: EquationGeneratorXMLClientJFrame.java
2  // Math-tutoring program using REST and XML to generate equations.
3  package com.deitel.equationgeneratorxmlclient;
4
5  import javax.swing.JOptionPane;
6  import javax.xml.bind.JAXB; // utility class for common JAXB operations
7
8  public class EquationGeneratorXMLClientJFrame extends javax.swing.JFrame
9  {
10    private String operation = "add"; // operation user is tested on
11    private int difficulty = 1; // 1, 2, or 3 digits in each number
12    private int answer; // correct answer to the question
13
14    // no-argument constructor
15    public EquationGeneratorXMLClientJFrame()
16    {
17       initComponents();
18    } // end no-argument constructor
19
```

Fig. 31.25 | Math-tutoring program using REST and XML to generate equations. (Part 1 of 8.)

```
20      // The initComponents method is autogenerated by NetBeans and is called
21      // from the constructor to initialize the GUI. This method is not shown
22      // here to save space. Open EquationGeneratorXMLClientJFrame.java in
23      // this example's folder to view the complete generated code.
24
25      // determine if the user answered correctly
26      private void checkAnswerJButtonActionPerformed(
27         java.awt.event.ActionEvent evt)
28      {
29         if ( answerJTextField.getText().equals( "" ) )
30         {
31            JOptionPane.showMessageDialog(
32               this, "Please enter your answer." );
33         } // end if
34
```

Fig. 31.25 | Math-tutoring program using REST and XML to generate equations. (Part 2 of 8.)

```
35          int userAnswer = Integer.parseInt( answerJTextField.getText() );
36
37          if ( userAnswer == answer )
38          {
39             equationJLabel.setText( "" ); // clear label
40             answerJTextField.setText( "" ); // clear text field
41             checkAnswerJButton.setEnabled( false );
42             JOptionPane.showMessageDialog( this, "Correct! Good Job!",
43                "Correct", JOptionPane.PLAIN_MESSAGE );
44          } // end if
45          else
46          {
47             JOptionPane.showMessageDialog( this, "Incorrect. Try again.",
48                "Incorrect", JOptionPane.PLAIN_MESSAGE );
49          } // end else
50       } // end method checkAnswerJButtonActionPerformed
51
```

Fig. 31.25 | Math-tutoring program using REST and XML to generate equations. (Part 3 of 8.)

```
52      // retrieve equation from web service and display left side to user
53      private void generateJButtonActionPerformed(
54         java.awt.event.ActionEvent evt)
55      {
56         try
57         {
58            String url = String.format( "http://localhost:8080/" +
59               "EquationGeneratorXML/resources/equation/%s/%d",
60               operation, difficulty );
61
62            // convert XML back to an Equation object
63            Equation equation = JAXB.unmarshal( url, Equation.class );
64
65            answer = equation.getResult();
66            equationJLabel.setText( equation.getLeftHandSide() + " =" );
67            checkAnswerJButton.setEnabled( true );
68         } // end try
69         catch ( Exception exception )
70         {
71            exception.printStackTrace();
72         } // end catch
73      } // end method generateJButtonActionPerformed
74
```

Fig. 31.25 | Math-tutoring program using REST and XML to generate equations. (Part 4 of 8.)

```
75      // obtains the mathematical operation selected by the user
76      private void operationJComboBoxItemStateChanged(
77         java.awt.event.ItemEvent evt)
78      {
79         String item = ( String ) operationJComboBox.getSelectedItem();
80
81         if ( item.equals( "Addition" ) )
82            operation = "add"; // user selected addition
83         else if ( item.equals( "Subtraction" ) )
84            operation = "subtract"; // user selected subtraction
85         else
86            operation = "multiply"; // user selected multiplication
87      } // end method operationJComboBoxItemStateChanged
88
89      // obtains the difficulty level selected by the user
90      private void levelJComboBoxItemStateChanged(
91         java.awt.event.ItemEvent evt)
92      {
93         // indices start at 0, so add 1 to get the difficulty level
94         difficulty = levelJComboBox.getSelectedIndex() + 1;
95      } // end method levelJComboBoxItemStateChanged
96
```

Fig. 31.25 | Math-tutoring program using REST and XML to generate equations. (Part 5 of 8.)

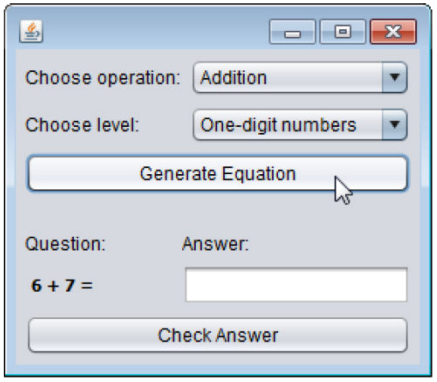
```
97      // main method begins execution
98      public static void main(String args[])
99      {
100        java.awt.EventQueue.invokeLater(
101           new Runnable()
102           {
103              public void run()
104              {
105                 new EquationGeneratorXMLClientJFrame().setVisible( true );
106              } // end method run
107           } // end anonymous inner class
108        ); // end call to java.awt.EventQueue.invokeLater
109     } // end main
110
```

Fig. 31.25 | Math-tutoring program using REST and XML to generate equations. (Part 6 of 8.)

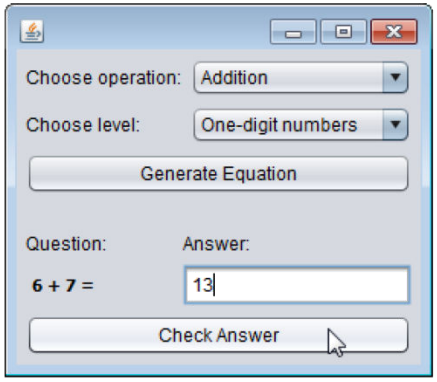
```
111     // Variables declaration - do not modify
112     private javax.swing.JLabel answerJLabel;
113     private javax.swing.JTextField answerJTextField;
114     private javax.swing.JButton checkAnswerJButton;
115     private javax.swing.JLabel equationJLabel;
116     private javax.swing.JButton generateJButton;
117     private javax.swing.JComboBox levelJComboBox;
118     private javax.swing.JLabel levelJLabel;
119     private javax.swing.JComboBox operationJComboBox;
120     private javax.swing.JLabel operationJLabel;
121     private javax.swing.JLabel questionJLabel;
122     // End of variables declaration
123 } // end class EquationGeneratorXMLClientJFrame
```

Fig. 31.25 | Math-tutoring program using REST and XML to generate equations. (Part 7 of 8.)

a) Generating a simple equation.



b) Sumbitting the answer.



c) Dialog indicating correct answer.

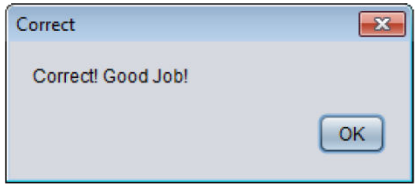


Fig. 31.25 | Math-tutoring program using REST and XML to generate equations. (Part 8 of 8.)

31.11.2 Consuming the `EquationGeneratorXML` Web Service (cont.)

- The event handler for `generateJButton` constructs the URL to invoke the web service, then passes this URL to the `unmarshal` method, along with an instance of `Class<Equation>`, so that JAXB can convert the XML into an `Equation` object.

31.11.3 Creating the `EquationGeneratorXML` Web Service

- As you saw in Section 31.8, RESTful web services can return data formatted as JSON as well.
- Figure 31.26 is a reimplementation of the `EquationGeneratorXML` service that returns an `Equation` in JSON format.
- The logic implemented here is the same as the XML version except that we use Gson to convert the `Equation` object into JSON instead of using JAXB to convert it into XML.
- Note that the `@Produces` annotation has also changed to reflect the JSON data format.

```
1  // Fig. 31.26: EquationGeneratorJSONResource.java
2  // RESTful equation generator that returns JSON.
3  package com.deitel.equationgeneratorjson;
4
5  import com.google.gson.Gson; // converts POJO to JSON and back again
6  import java.util.Random;
7  import javax.ws.rs.GET;
8  import javax.ws.rs.Path;
9  import javax.ws.rs.PathParam;
10 import javax.ws.rs.Produces;
11
12 @Path( "equation" )
13 public class EquationGeneratorJSONResource
14 {
15    static Random randomObject = new Random(); // random number generator
16
```

Fig. 31.26 | RESTful equation generator that returns JSON. (Part 1 of 2.)

```
17      // retrieve an equation formatted as JSON
18      @GET
19      @Path( "{operation}/{level}" )
20      @Produces( "application/json" )
21      public String getJson( @PathParam( "operation" ) String operation,
22         @PathParam( "level" ) int level )
23      {
24         // compute minimum and maximum values for the numbers
25         int minimum = ( int ) Math.pow( 10, level - 1 );
26         int maximum = ( int ) Math.pow( 10, level );
27
28         // create the numbers on the left-hand side of the equation
29         int first = randomObject.nextInt( maximum - minimum ) + minimum;
30         int second = randomObject.nextInt( maximum - minimum ) + minimum;
31
32         // create Equation object and return result
33         Equation equation = new Equation( first, second, operation );
34         return new Gson().toJson( equation ); // convert to JSON and return
35      } // end method getJson
36   } // end class EquationGeneratorJSONResource
```

Fig. 31.26 | RESTful equation generator that returns JSON. (Part 2 of 2.)

```
1  // Fig. 31.27: EquationGeneratorJSONClientJFrame.java
2  // Math-tutoring program using REST and JSON to generate equations.
3  package com.deitel.equationgeneratorjsonclient;
4
5  import com.google.gson.Gson; // converts POJO to JSON and back again
6  import java.io.InputStreamReader;
7  import java.net.URL;
8  import javax.swing.JOptionPane;
9
10 public class EquationGeneratorJSONClientJFrame extends javax.swing.JFrame
11 {
12    private String operation = "add"; // operation user is tested on
13    private int difficulty = 1; // 1, 2, or 3 digits in each number
14    private int answer; // correct answer to the question
15
16    // no-argument constructor
17    public EquationGeneratorJSONClientJFrame()
18    {
19       initComponents();
20    } // end no-argument constructor
21
```

Fig. 31.27 | Math-tutoring program using REST and JSON to generate equations. (Part 1 of 8.)

```
22      // The initComponents method is autogenerated by NetBeans and is called
23      // from the constructor to initialize the GUI. This method is not shown
24      // here to save space. Open EquationGeneratorJSONClientJFrame.java in
25      // this example's folder to view the complete generated code.
26
27      // determine if the user answered correctly
28      private void checkAnswerJButtonActionPerformed(
29         java.awt.event.ActionEvent evt)
30      {
31         if ( answerJTextField.getText().equals( "" ) )
32         {
33            JOptionPane.showMessageDialog(
34               this, "Please enter your answer." );
35         } // end if
36
```

Fig. 31.27 | Math-tutoring program using REST and JSON to generate equations. (Part 2 of 8.)

```
37          int userAnswer = Integer.parseInt( answerJTextField.getText() );
38
39          if ( userAnswer == answer )
40          {
41             equationJLabel.setText( "" ); // clear label
42             answerJTextField.setText( "" ); // clear text field
43             checkAnswerJButton.setEnabled( false );
44             JOptionPane.showMessageDialog( this, "Correct! Good Job!",
45                "Correct", JOptionPane.PLAIN_MESSAGE );
46          } // end if
47          else
48          {
49             JOptionPane.showMessageDialog( this, "Incorrect. Try again.",
50                "Incorrect", JOptionPane.PLAIN_MESSAGE );
51          } // end else
52       } // end method checkAnswerJButtonActionPerformed
53
```

Fig. 31.27 | Math-tutoring program using REST and JSON to generate equations. (Part 3 of 8.)

```
54      // retrieve equation from web service and display left side to user
55      private void generateJButtonActionPerformed(
56         java.awt.event.ActionEvent evt)
57      {
58         try
59         {
60            // URL of the EquationGeneratorJSON service, with parameters
61            String url = String.format( "http://localhost:8080/" +
62               "EquationGeneratorJSON/resources/equation/%s/%d",
63               operation, difficulty );
64
65            // open URL and create a Reader to read the data
66            InputStreamReader reader =
67               new InputStreamReader( new URL( url ).openStream() );
68
69            // convert the JSON back into an Equation object
70            Equation equation =
71               new Gson().fromJson( reader, Equation.class );
72
```

Fig. 31.27 | Math-tutoring program using REST and JSON to generate equations. (Part 4 of 8.)

```
73            // update the internal state and GUI to reflect the equation
74            answer = equation.getResult();
75            equationJLabel.setText( equation.getLeftHandSide() + " =" );
76            checkAnswerJButton.setEnabled( true );
77         } // end try
78         catch ( Exception exception )
79         {
80            exception.printStackTrace();
81         } // end catch
82      } // end method generateJButtonActionPerformed
83
```

Fig. 31.27 | Math-tutoring program using REST and JSON to generate equations. (Part 5 of 8.)

```
84      // obtains the mathematical operation selected by the user
85      private void operationJComboBoxItemStateChanged(
86         java.awt.event.ItemEvent evt)
87      {
88         String item = ( String ) operationJComboBox.getSelectedItem();
89
90         if ( item.equals( "Addition" ) )
91            operation = "add"; // user selected addition
92         else if ( item.equals( "Subtraction" ) )
93            operation = "subtract"; // user selected subtraction
94         else
95            operation = "multiply"; // user selected multiplication
96      } // end method operationJComboBoxItemStateChanged
97
98      // obtains the difficulty level selected by the user
99      private void levelJComboBoxItemStateChanged(
100        java.awt.event.ItemEvent evt)
101     {
102        // indices start at 0, so add 1 to get the difficulty level
103        difficulty = levelJComboBox.getSelectedIndex() + 1;
104     } // end method levelJComboBoxItemStateChanged
105
```

Fig. 31.27 | Math-tutoring program using REST and JSON to generate equations. (Part 6 of 8.)

```
106     // main method begins execution
107     public static void main( String args[] )
108     {
109        java.awt.EventQueue.invokeLater(
110           new Runnable()
111           {
112              public void run()
113              {
114                 new EquationGeneratorJSONClientJFrame().setVisible( true );
115              } // end method run
116           } // end anonymous inner class
117        ); // end call to java.awt.EventQueue.invokeLater
118     } // end main
119
```

Fig. 31.27 | Math-tutoring program using REST and JSON to generate equations. (Part 7 of 8.)

```
120     // Variables declaration - do not modify
121     private javax.swing.JLabel answerJLabel;
122     private javax.swing.JTextField answerJTextField;
123     private javax.swing.JButton checkAnswerJButton;
124     private javax.swing.JLabel equationJLabel;
125     private javax.swing.JButton generateJButton;
126     private javax.swing.JComboBox levelJComboBox;
127     private javax.swing.JLabel levelJLabel;
128     private javax.swing.JComboBox operationJComboBox;
129     private javax.swing.JLabel operationJLabel;
130     private javax.swing.JLabel questionJLabel;
131     // End of variables declaration
132 } // end class EquationGeneratorJSONClientJFrame
```

Fig. 31.27 | Math-tutoring program using REST and JSON to generate equations. (Part 8 of 8.)

31.11.4 Consuming the `EquationGeneratorJSON` Web Service

- The program in Fig. 31.27 consumes the `EquationGeneratorJSON` service and performs the same function as `EquationGeneratorXMLClient`—the only difference is in how the `Equation` object is retrieved from the web service.
- We use the `URL` class and an `InputStreamReader` to invoke the web service and read the response.
- The retrieved JSON is deserialized using Gson and converted back into an `Equation` object.