Lecture 15b

RESTful Web Services

OBJECTIVES

In this lecture you will learn:

- How to publish and consume web services in NetBeans.
- How JSON and Representational State Transfer (REST) Architecture enable Java web services.
- How to create client desktop and web applications that consume web services.
- How to pass objects of user defined types to and return them from a web service.

References:

P. Deitel, H. Deitel, "Java How to Program (Early objects)", Prentice Hall 10th ed. 2014, ISBN-10: 0133807800, ISBN-13: 9780133807806 (ch. 32)

15.1 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 lecture, we use 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.

15.4 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.

15.5 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

15.5 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' } ]
```

15.6 Publishing and Consuming REST-Based XML Web Services

Now, we access a Java web service using the REST architecture.

We recreate the `WelcomeSOAP` example from **Lecture 6** to return data in plain XML format.

15.6 Publishing and Consuming REST-Based XML Web Services

- Everything in REST is considered as a resource.
- Every resource is identified by an URI.
- Uses uniform interfaces. Resources are handled using POST, GET, PUT, DELETE operations which are similar to Create, Read, update and Delete(CRUD) operations.
- Be stateless. Every request is an independent request. Each request from client to server must contain all the information necessary to understand the request.
- Communications are done via representations. e.g. XML, JSON

15.6.1 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.

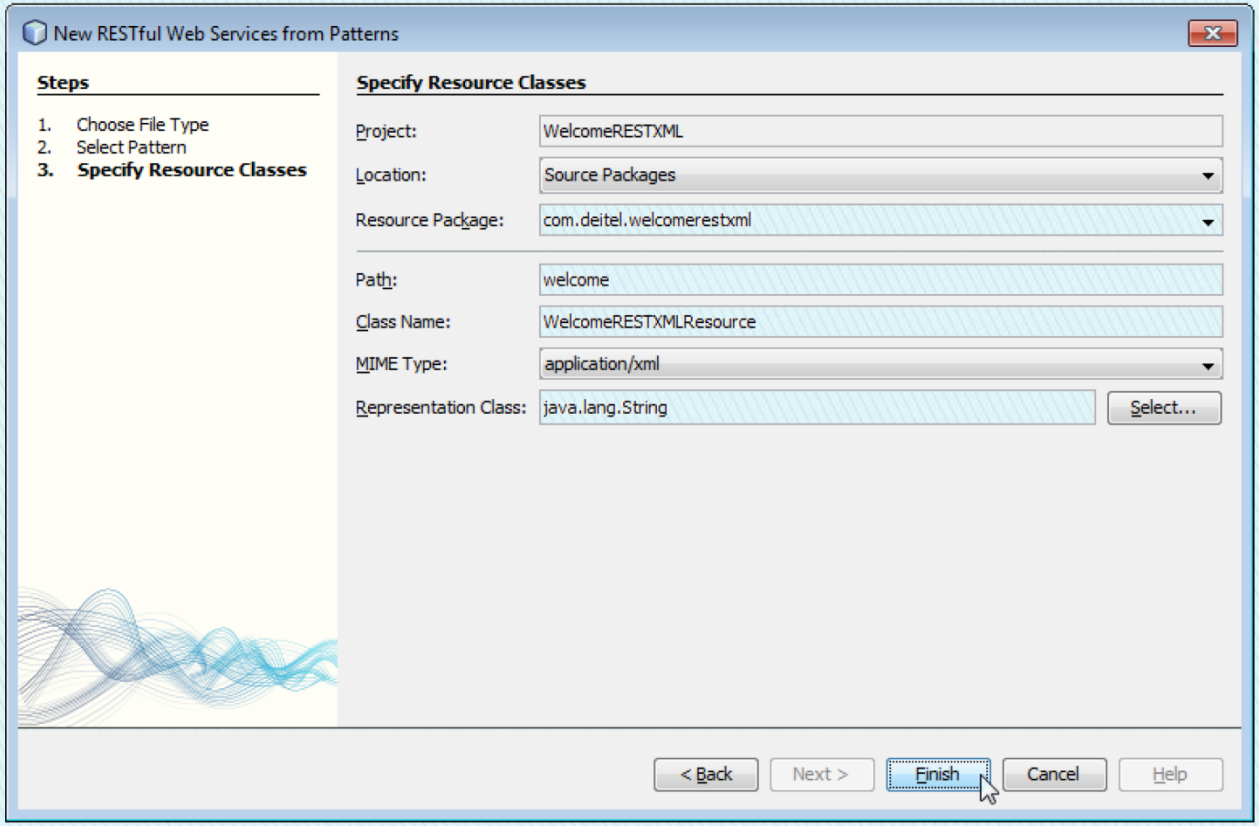


Fig. 31.11 | Creating the `WelcomeRESTXML` RESTful web service.

15.6.1 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. 15.12.

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.)

15.6.1 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.

15.6.1 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.

15.6.1 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.

15.6.1 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. 15.12), 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.

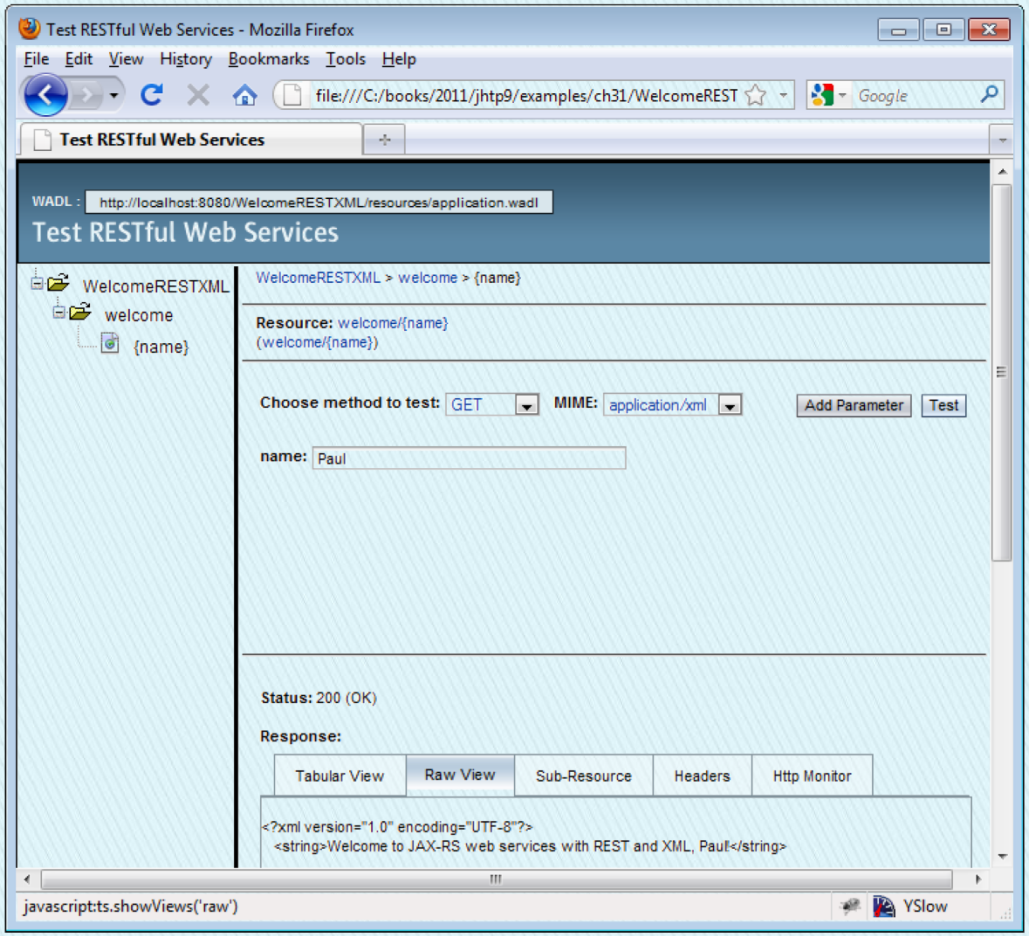


Fig. 31.13 | Test page for the `WelcomeRESTXML` web service.

15.7.1 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.

15.6.2 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.)

15.7 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/`.

15.7.1 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.

```
1  // Fig. 31.15: WelcomeRESTJSONResource.java
2  // REST web service that returns a welcome message as JSON.
3  package com.deitel.welcomerestjson;
4
5  import com.google.gson.Gson; // converts POJO to JSON and back again
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
11 @Path( "welcome" ) // path used to access the resource
12 public class WelcomeRESTJSONResource
13 {
14    // retrieve welcome message
15    @GET // handles HTTP GET requests
16    @Path( "{name}" ) // takes name as a path parameter
17    @Produces( "application/json" ) // response formatted as JSON
18    public String getJson( @PathParam( "name" ) String name )
19    {
20       // add welcome message to field of TextMessage object
21       TextMessage message = new TextMessage(); // create wrapper object
22       message.setMessage( String.format( "%s, %s!",
23          "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.)

15.7.2 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.)

15.11 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.

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

We define class `Equation` in Fig. 15.23.

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.)

15.8.1 Creating the EquationGeneratorXML Web Service

Figure 15.24 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.)

15.8.2 Consuming the EquationGeneratorXML Web Service

The `EquationGeneratorXMLClient` application (Fig. 15.24) 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.)

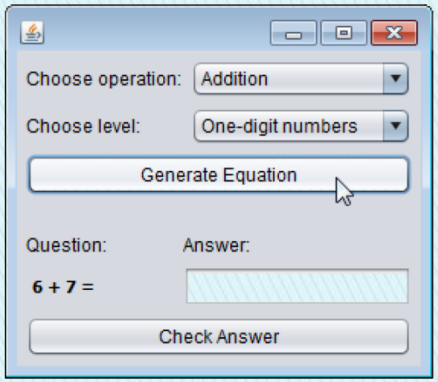
```
97      // main method begins execution
98      public static void main(String args[])
99      {
100        java.awt.EventQueue.invokeLater(
101           new Runnable()
102           {
103              public void run()
104              {
105                 new EquationGeneratorXMLClientJFrame().setVisible( 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.)

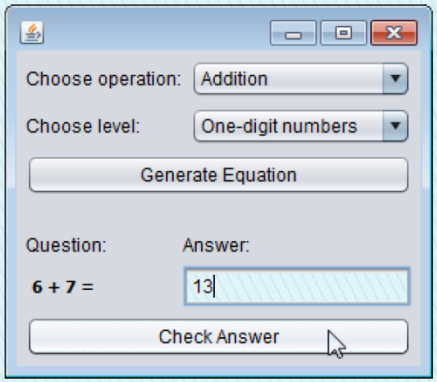
```
111     // Variables declaration - do not modify
112     private javax.swing.JLabel answerJLabel;
113     private javax.swing.JTextField answerJTextField;
114     private javax.swing.JButton checkAnswerJButton;
115     private javax.swing.JLabel equationJLabel;
116     private javax.swing.JButton 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.

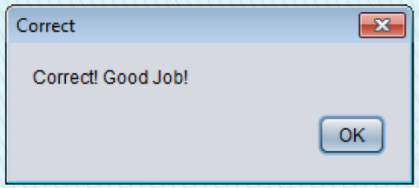


Fig. 31.25 | Math-tutoring program using REST and XML to generate equations. (Part 8 of 8.)

15.8.2 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.

15.8.3 Creating the EquationGeneratorXML Web Service

RESTful web services can return data formatted as JSON as well.

Figure 15.26 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.)

15.8.4 Consuming the EquationGeneratorJSON Web Service

The program in Fig. 15.27 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.

Problems

Solve the problems described in `tutorials.zip`.

Watch:

`http://www.youtube.com/watch?v=cDdfVMro99s`

`http://www.youtube.com/watch?v=_c-CCVy4_Eo`