95-702 Distributed Systems

Project 3

Assigned: Sunday, February 23, 2014

Due Friday March 21, 11:59 PM

For this project, you may work in groups of size one or two. The project submission must clearly contain your name(s). Only one person on the team should provide a submission to Blackboard. All the members of the team should contribute to each solution. All the members of the team must understand all of the code submitted. On future exams, it will be assumed that you have worked through all of this material.

Project Topics: Web Service Design Styles

Web Services are used to provide interoperable client and server interaction. The client may be a stand-alone program or a program running within a browser. Web services are an important foundation for the construction of service-oriented architectures.

In this project, we will build several small systems that illustrate various ways to design web services.

Your first task will be to build, deploy, and test a simple JAX-WS web service and web service client. Your solution will employ SOAP documents holding the procedure name and a list of typed parameters. As was discussed in class, this is a tightly coupled approach. It has the benefit of being very simple to build and deploy.

Your second task will be to build a JAX-WS web service using a single message argument. This is a less tightly coupled approach – allowing clients to change – but it is more of a challenge to build. You will have to take care of the XML messages that are placed within the SOAP documents. Note that when an XML message arrives on the server as a request or on the client as a response, you will need to parse it with an XML parser. It is recommended that you spend some time writing some small Java examples that read and process XML. In my solution, I used DOM processing and the following code to build the DOM tree:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder;
Document spyDoc = null;
try  {
```

```
    builder = factory.newDocumentBuilder();
    spyDoc = builder.parse( new InputSource( new StringReader( xmlString ) ));
  }
  catch (Exception e) {
      e.printStackTrace(); }
```

Your third task will be to utilize JAX-RS. This is a REST-style API and Netbeans provides some solid support.

In each of these assignments we will be making use of the two classes provided below. One class (Spy.java) holds information on a single spy. The class (SpyList.java) is a singleton (may only be created once) and maintains a list of spies. Feel free to use and modify this code as you see fit. It may need to change depending on which Task you are working on.

Task 1 A Tightly Coupled JAX-WS Web Service With Typed Parameters

In this task, you will create two Netbeans projects named WebServiceDesignStyles1Project and WebServiceDesignStyles1ProjectClient.

Be sure to take screen shots showing the testing platform running in the browser – showing the SOAP documents.

1) Write a JAX-WS web services (using SOAP) that allows a client to add a spy, delete a spy and list all of the spies in a spy list.

   The web service methods have the following signatures and pre- and post-conditions:

   Operation: addSpy

   Pre: A name, title, location, and password are provided as input.

   Post: The spy is added to the list of spies. The value returned

   is a simple String that says “Spy” <name> “was added to the list.”

```
@WebMethod(operationName = "addSpy")
public String addSpy(@WebParam(name = "name") String name,
                     @WebParam(name = "title") String title,
                     @WebParam(name = "location") String location,
```

```
                    @WebParam(name = "password") String password);
```

Operation: deleteSpy

Pre: A name that is in the list of spies is provided as input.

Post: The spy is deleted from the list of spies. The value returned is a simple String that says “Spy” <name> “was deleted from the list.”

```
@WebMethod(operationName = "deleteSpy")
public String deleteSpy(@WebParam(name = "name") String name)
```

Operation: getList

Pre: None

Post: A simple string is returned that contains all of the spy data on the list.

```
@WebMethod(operationName = "getList")
public String getList()
```

2) Write a JAX-WS web service client (using SOAP) with a main routine that tests the web service. The main routine must test each operation of the service. There does not have to be any user interaction. For example, my main routine looks like this (yours will be similar):

```
addSpy("joem", "spy", "Pittsburgh", "joe");
addSpy("mikem", "spy", "Philadelphia", "mike");
addSpy("jamesb", "spy", "Toronto", "james");
System.out.println("Three spies added to spy list");
System.out.println("The list contains: " + getList());
deleteSpy("mikem");
System.out.println("Mike was deleted");
System.out.println("The list now contains: " + getList());
```

Task 2 A Less Tightly Coupled JAX-WS Web Service Using a Single Message Argument

In this task, you will create two Netbeans projects named WebServiceDesignStyles2Project and WebServiceDesignStyles2ProjectClient.

Be sure to take screen shots showing the testing platform running in the browser – showing the SOAP documents.

1) In general, this is the same assignment as in Task 1. In this case, however, we are changing the style of the web service. Write a JAX-WS web services (using SOAP) that allows a client to add a spy, delete a spy and list all of the spies in a spy list.

   The web service methods have the following signatures and pre- and post-conditions:

   Operation: addSpy

   Pre: A name, title, location, and password are provided as an input string

   holding an XML document representing a spy. The format of the string is as

   follows (whitespace has been added for clarity but not found in the actual parameter):

```
<spy>
        <name>Michael</name>
        <location>Pittsburgh</location>
        <password>sesame</password>
        <title>spy master</title>
</spy>
```

   Post: The operation returns an XML representation of the spy that just added.

   The format of the XML representation is exactly the same as the input parameter.

```
@WebMethod(operationName = "addSpy")
public String addSpy(@WebParam(name = "spy") String spyXML)
```

Operation: deleteSpy

Pre: A name that is in the list of spies is provided as input. The name is marked up in XML in the following format:

<spy><name>Mike</name></spy>

Post: The spy is deleted from the list of spies. The value returned is a simple String that says “Spy” <name> “was deleted from the list.”

```
@WebMethod(operationName = "deleteSpy")
public String deleteSpy(@WebParam(name = "name") String nameXML);
```

2) Write a JAX-WS web service client (using SOAP) with a main routine that tests the web service. The main routine must test each operation of the service. There does not have to be any user interaction. For example, my main routine looks like this (yours will be similar):

```
String spy =
"<spy><name>Michael</name><location>Pittsburgh</location>" +
“<password>sesame</password><title>spy master</title></spy>";
addSpy(spy);
spy =
"<spy><name>Joe</name><location>Philadelphia</location>" +
“<password>xyz</password><title>spy agent</title></spy>";
addSpy(spy);
String list = getList();
System.out.println("Two Spies \n" + list);
String nameSpy = "<spy><name>Michael</name></spy>";
deleteSpy(nameSpy);
list = getList();
System.out.println("One Spy \n" + list);
nameSpy = "<spy><name>Joe</name></spy>";
deleteSpy(nameSpy);
```

```
list = getList();
System.out.println("Zero Spies \n" + list);
spy =
 "<spy><name>Michael</name><location>Pittsburgh</location> “+
 “<password>sesame</password><title>spy master</title></spy>";
addSpy(spy);
list = getList();
System.out.println("One Spy \n" + list);
System.out.println("List as XML");
Document doc = getDocument(list);  // See document builder code above
doc.getDocumentElement().normalize();
NodeList nl = doc.getElementsByTagName("name");
Node n = nl.item(0);
String name = n.getTextContent();
System.out.println("Should be the spy Mike " + name);
```

Task 3 A Less Tightly Coupled JAX-RS Web Service Using REST

In this task, you will create two Netbeans projects named "WebServiceDesignStyles3Project" and "WebServiceDesignStyles3ProjectClient".

Be sure to take screen shots showing the testing platform running in the browser – showing the options that may be selected.

1) Write a JAX-RS service that provides access to the list of spies. Your web service will be built from the "Restful Web Service from Patterns" option. Select the "Simple Root Resource" option. Name both the path and Class Name "SpyList".

With respect to the SpyList resource as a whole, you will automatically be provided with GET and PUT operations. These operations will operate on the SpyList resource. GET will return a list of spies (in XML) and PUT will add an individual spy to the list (the individual will also be represented in XML).

Add a DELETE operation that operates on a particular spy and a GET operation that operates on a particular spy. The DELETE operation will remove a spy from the spy list and a GET operation will retrieve a particular spy from the spy list.

The signature for the DELETE spy is shown here:

```
//Pre: spyName is is in the spy list
//Post: the spy is removed from the spy list and an XML representation
// of the spy deleted is returned to the caller.
@DELETE
@Path("{spyName}")
@Produces("application/xml")
public String deleteSpy(@PathParam("spyName") String spyName)
```

In summary, our service will provide for two different GET operations, one PUT operation and one DELETE.

Note: When you test your RESTFul web service, you will be prompted with a Configure REST Test client dialogue box. Choose the Web Test Client in Project option and select the WebServiceDesignStyles3Project.

Note: The JAX-RS services default to "per-request". This means that the service is created and destroyed on each visit. We want our service to remain in memory (and to hold the list of spies). So, just before the beginning of the class, mark the class as a singleton. This annotation is from the javax.inject package. This will allow the service to stay in memory between visits. The annotation looks like the following:

```
@Path("SpyList")
@Singleton
public class SpyList ...
```

2) Write a JAX-RS 2.0 web service client with a main routine that tests the REST web service. The main routine must test each operation of the service. There does not have to be any user interaction. Note that JAX-RS 2.0 provides support to the client side programmer. To do this task, you will need to research client side programming using JAX-RS 2.0.

Project 3 Summary

Be sure to review the grading rubric on the schedule. We will use that rubric in evaluating this project. Documentation is always required.

There will be 6 projects in Netbeans.

- WebServiceDesignStyles1Project
- WebServiceDesignStyles1ProjectClient
- WebServiceDesignStyles2Project
- WebServiceDesignStyles2ProjectClient
- WebServiceDesignStyles3Project
- WebServiceDesignStyles3ProjectClient

You should also have three screen shot folders:

- Project3Task1ScreenShots
- Project3Task2ScreenShots
- Project3Task3ScreenShots

Copy all of your Netbeans project folders and screenshot folders into a folder named with your id.

Zip that folder and submit it to Blackboard.

The submission should be a single zip file.

Project 3 Code

Spy.java

```
package edu.cmu.andrew.mm6;
class Spy {
    // instance data for spies
    private String name;
    private String title;
    private String location;
    private String password;

    public Spy() {
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getLocation() {
        return location;
    }

    public String getName() {
        return name;
    }

    public String getTitle() {
        return title;
```

```
}

public void setLocation(String location) {
    this.location = location;
}

public void setName(String name) {
    this.name = name;
}

public void setTitle(String title) {
    this.title = title;
}

public Spy(String name, String title, String location, String password) {
    this.name = name;
    this.title = title;
    this.location = location;
    this.password = password;
}

public Spy(String name, String title, String location) {
    this.name = name;
    this.title = title;
    this.location = location;
}

public String toXML() {
    StringBuffer xml = new StringBuffer();

    xml.append("<spy>");
    xml.append("<name>" + name + "</name>");
    xml.append("<title>" + title + "</title>");
```

```
        xml.append("<location>" + location + "</location>");
        xml.append("<password>" + password + "</password>");
        xml.append("</spy>");

        return xml.toString();

    }

    public static void main(String args[]) {
        Spy s = new Spy("james","spy", "Pittsburgh", "james");
        System.out.println(s);
    }
}
```

SpyList.java

```
package edu.cmu.andrew.mm6;
import java.util.Collection;
import java.util.Iterator;
import java.util.Map;
import java.util.TreeMap;

public class SpyList {

  private Map tree = new TreeMap();

    private static SpyList spyList = new SpyList();

    private SpyList() {
    }

    public static SpyList getInstance() {
        return spyList;
```

```
}
public void add(Spy s) {
    tree.put(s.getName(), s);
}
public void delete(Spy s) {
    tree.remove(s.getName());
}
public Spy get(String userID) {
    return (Spy) tree.get(userID);
}

public Collection getList() {
    return tree.values();
}

public String toString() {

    StringBuffer representation = new StringBuffer();
    Collection c = getList();
    Iterator sl = c.iterator();
    while(sl.hasNext()) {
        Spy spy = (Spy)sl.next();
        representation.append("Name: " + spy.getName()+" Title: " + spy.getTitle()+
                                    " Location: " + spy.getLocation());
    }
    return representation.toString();
}

public String toXML() {
    StringBuffer xml = new StringBuffer();
    xml.append("<spylist>\n");

    Collection c = getList();
```

```
        Iterator sl = c.iterator();
        while(sl.hasNext()) {
            Spy spy = (Spy)sl.next();
            xml.append(spy.toXML());
        }
        // Now, close
        xml.append("</spylist>");

        System.out.println("Spy list: " + xml.toString());
        return xml.toString();
    }
}
```