ASSIGNMENT NO. 7

Problem Statement:

To create a simple web service and write any distributed application to consume the web service.

Tools / Environment:

Java Programming Environment, JDK 8, Netbeans IDE with GlassFish Server

Related Theory:

Web Service:
A web service can be defined as a collection of open protocols and standards for exchanging information among systems or applications.
A service can be treated as a web service if:

- The service is discoverable through a simple lookup
- It uses a standard XML format for messaging
- It is available across internet/intranet networks.
- It is a self-describing service through a simple XML syntax
- The service is open to, and not tied to, any operating system/programming language

Types of Web Services:
There are two types of web services:

1. **SOAP**: SOAP stands for Simple Object Access Protocol. SOAP is an XML based industry standard protocol for designing and developing web services. Since it's XML based, it's platform and language independent. So, our server can be based on JAVA and the client can be on .NET, PHP etc. and vice versa.

2. **REST**: REST (Representational State Transfer) is an architectural style for developing web services. It's getting popular recently because it has a small learning curve when compared to SOAP. Resources are core concepts of Restful web services and they are uniquely identified by their URIs.

Web service architectures:
As part of a web service architecture, there exist three major roles.
Service Provider is the program that implements the service agreed for the web service and exposes the service over the internet/intranet for other applications to interact with.

Service Requestor is the program that interacts with the web service exposed by the Service Provider. It makes an invocation to the web service over the network to the Service Provider and exchanges information.

Service Registry acts as the directory to store references to the web services.
The following are the steps involved in a basic SOAP web service operational behavior:

1. The client program that wants to interact with another application prepares its request content as a SOAP message.

2. Then, the client program sends this SOAP message to the server web service as an HTTP POST request with the content passed as the body of the request.

3. The web service plays a crucial role in this step by understanding the SOAP request and converting it into a set of instructions that the server program can understand.

4. The server program processes the request content as programmed and prepares the output as the response to the SOAP request.

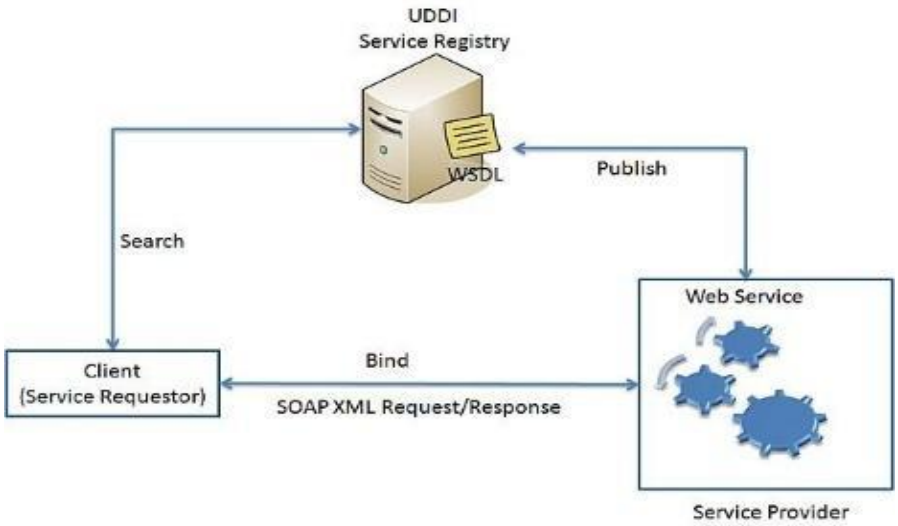
5. Then, the web service takes this response content as a SOAP message and reverts to the SOAP HTTP request invoked by the client program with this response.

6. The client program web service reads the SOAP response message to receive the outcome of the server program for the request content it sent as a request.

SOAP web services:
Simple Object Access Protocol (**SOAP**) is an XML-based protocol for accessing web services.
It is a W3C recommendation for communication between two applications, and it is a platform and language-independent technology in integrated distributed applications.

While XML and HTTP together make the basic platform for web services, the following are the key components of standard SOAP web services:
Universal Description, Discovery, and Integration (**UDDI**)*:* UDDI is an XML based framework for describing, discovering, and integrating web services. It acts as a directory of web service interfaces described in the WSDL language.

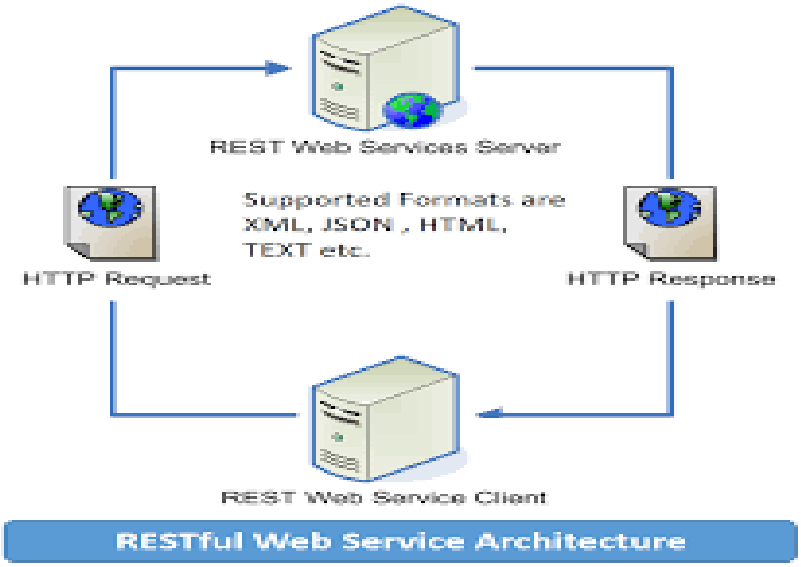
Web Services Description Language (**WSDL**)*:* WSDL is an XML document containing information about web services, such as the method name, method parameters, and how to invoke the service. WSDL is part of the UDDI registry. It acts as an interface between applications that want to interact based on web services. The following diagram shows the interaction between the UDDI, Service Provider, and service consumer in SOAP web services:



SOAP Web Services Architecture

RESTful web services

REST stands for **Representational State Transfer**. RESTful web services are considered a performance-efficient alternative to the SOAP web services. REST is an architectural style, not a protocol. Refer to the following diagram:



While both SOAP and RESTful support efficient web service development, the difference between these two technologies can be checked out in the following table:

	SOAP	REST
Bandwith usage	Uses more bandwith over the internet	Uses less bandwith
Client-server coupling	Tighter client-server coupling	Looser client server coupling
Security	Built in mechanism for security	No built in security
Data formats	Supports only XML	Supports multiple formats
Exposing business logic	Service interfaces	URIs
Failure handling	Retry logic built-in	Expects clients to retry
Caching data	Cannot be cached	Can be cached
Java API	JAX-WS	JAX-RS

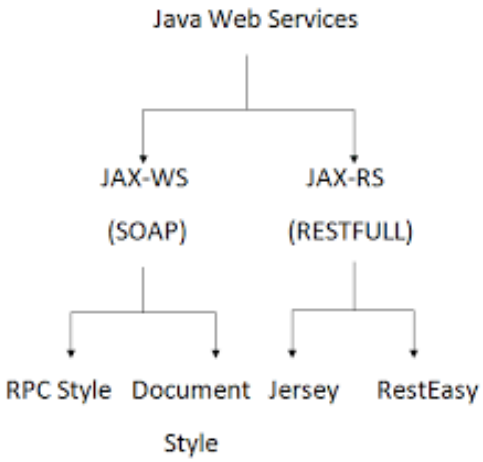
Designing the solution:

Java provides it's own API to create both SOAP as well as RESTful web services.

1. **JAX-WS**: JAX-WS stands for Java API for XML Web Services. JAX-WS is XML based Java API to build web services server and client application.

2. **JAX-RS**: Java API for RESTful Web Services (JAX-RS) is the Java API for creating REST web services. JAX-RS uses annotations to simplify the development and deployment of web services.

Both of these APIs are part of standard JDK installation, so no need to add any jars to work with them.



Implementing the solution:

Step 1: Choosing a container:

You can either deploy your web service in a web container.

- Choose **File** > **New Project** (Ctrl-Shift-N on Linux and Windows).
- Select **Web Application** from the Java **Web** category.
- Name the project *CalculatorWSApplication*.
- Select a location for the project. Click **Next**.
- Select the server **[Glassfish / Tomcat]** and **Java EE** version and click **Finish.**

Step 2: Creating a Web Service from a Java Class:

- Right-click the *CalculatorWSApplication* node and **choose New** > **Web Service**.
- Name the web service *CalculatorWS* and type *org.me.calculator* in Package.
- Keep "**Create Web Service from Scratch**" check box selected.
- If you are creating a Java EE project on GlassFish, select "**Implement Web Service as a Stateless Session Bean"**.
- Click **Finish**. The Projects window displays the structure of the new web service and the source code is shown in the editor area.

Step 3: Adding an Operation to the Web Service:

- Find the web service's node in the Projects window. Right-click that node. A context menu opens.
- Click **Add Operation** in either the visual designer or the context menu. The Add Operation dialog opens.
- In the upper part of the **Add Operation dialog box**, type **add** in **Name** and type **int** in the **Return Type** drop-down list.
- In the lower part of the Add Operation dialog box, click **Add** and create a parameter of type **int** named **i**. Click **Add** again and create a parameter of type **int** called **j**.
- Click **OK** at the bottom of the Add Operation dialog box. You return to the editor.
- Remove the **default hello operation**, either by deleting the hello() method in the **source code** or by selecting the hello operation in the **visual designer** and clicking **Remove Operation**.
- Click **Source** menu and view the generated code.
- In the editor, extend the skeleton **add operation.** Add the following:

```
int k = i + j;
return k; (instead of return 0)
```

Step 4: Deploying and Testing the Web Service:

Once you deploy a web service to a server, you can use the IDE to open the server's test client. The GlassFish server provides test clients whereas in Tomcat Web Server, there is no test client.

- Right-click the **project** and choose **Deploy**. The IDE starts the application server, builds the application, and deploys the application to the server.
- In the IDE's **Projects** tab, expand the **Web Services** node of the *CalculatorWSApplication* project. Right-click the *CalculatorWS* node, and choose **Test Web Service.**
- The IDE opens the tester page in the browser, if you deployed a web application to the GlassFish server.

Step 5: Consuming the Web Service:

Once the web service is deployed, you need to **create a client** to make use of the web service's **add** method. Here, you can create three types of clients: a Java class in a Java SE application, a servlet, and a JSP page in a web application.

Client 1: Java Class in Java SE Application

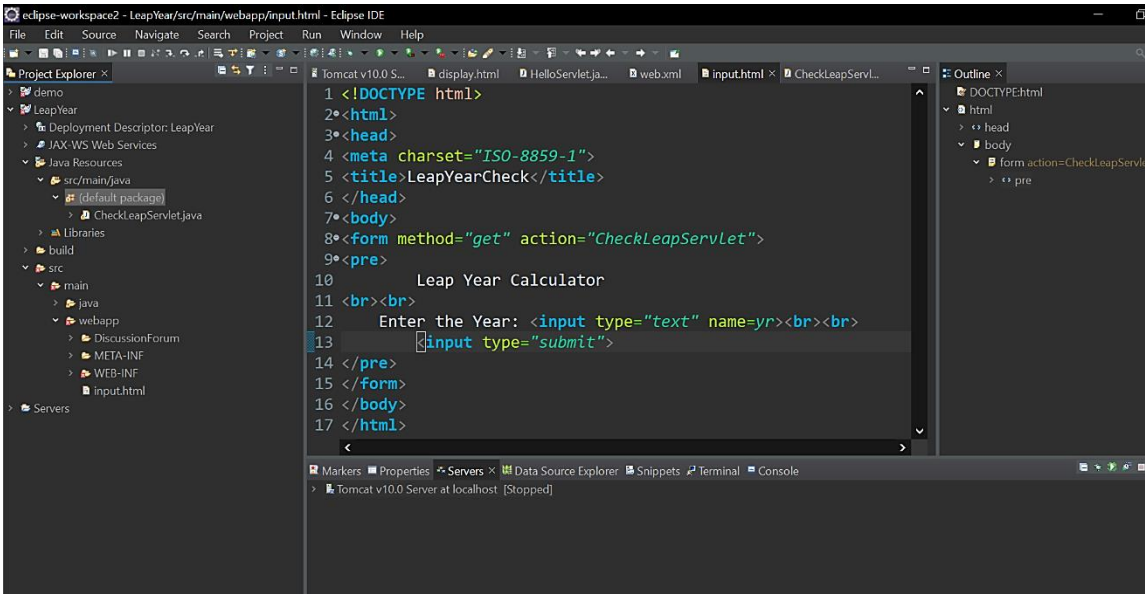
- Choose **File** > **New Project**.
- Select **Java Application** from the **Java** category.
- Name the project CalculatorWS_Client_Application.
- Keep "**Create Main Class selected**" and accept all other default settings. Click **Finish**.
- Right-click the CalculatorWS_Client_Application node and choose **New** > **Web Service Client**. The New Web Service Client wizard opens.
- Select "Project as the **…** source. Click on **Browse** button. Browse to the CalculatorWS web service in the **CalculatorWSApplication** project.
- When you have selected the web service, click **OK**.
- Do not select **a package** name. Leave this field empty. Keep the other settings at default and click **Finish**.
- The Projects window displays the new **web service client**, with a node for the **add** method that is created
- Double-click your **main class** so that it opens in the Source Editor. Drag the **add** node below the main() method.
- In the main() method body, **write the** code that initializes values for i and j, calls add(), and prints the result.

```
try
{
     int i=3;
     int j=4;
     int result = add(i,j);
     System.out.println("Result =" + result);
} catch (Exception ex) {
          System.out.println("Exception =" + ex);
```

Compiling and Executing the solution:

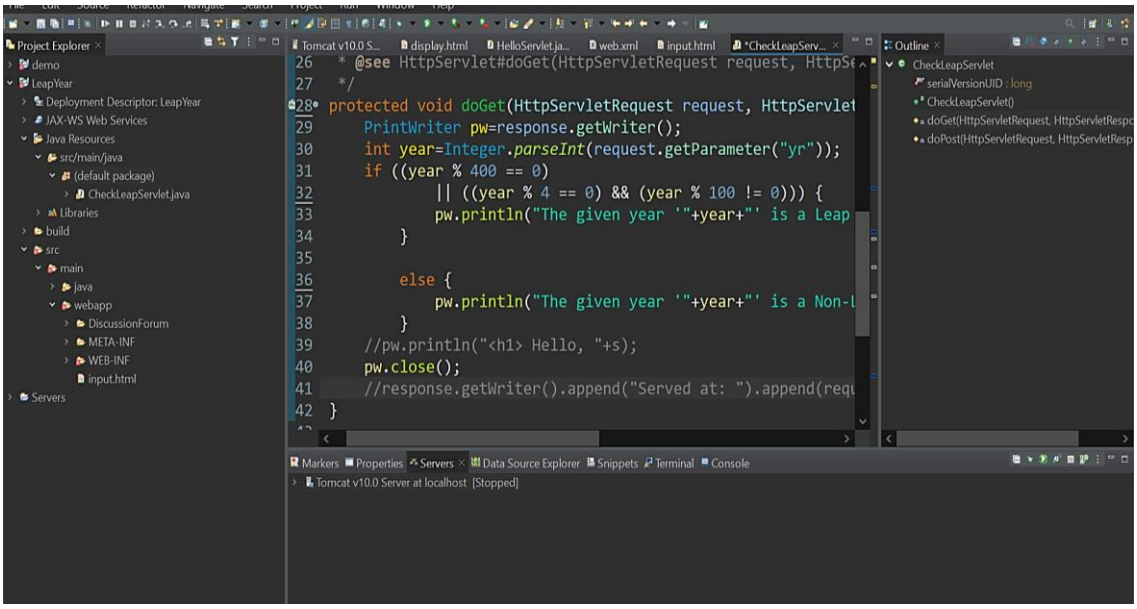
Right Click on the Project node and Choose Run.

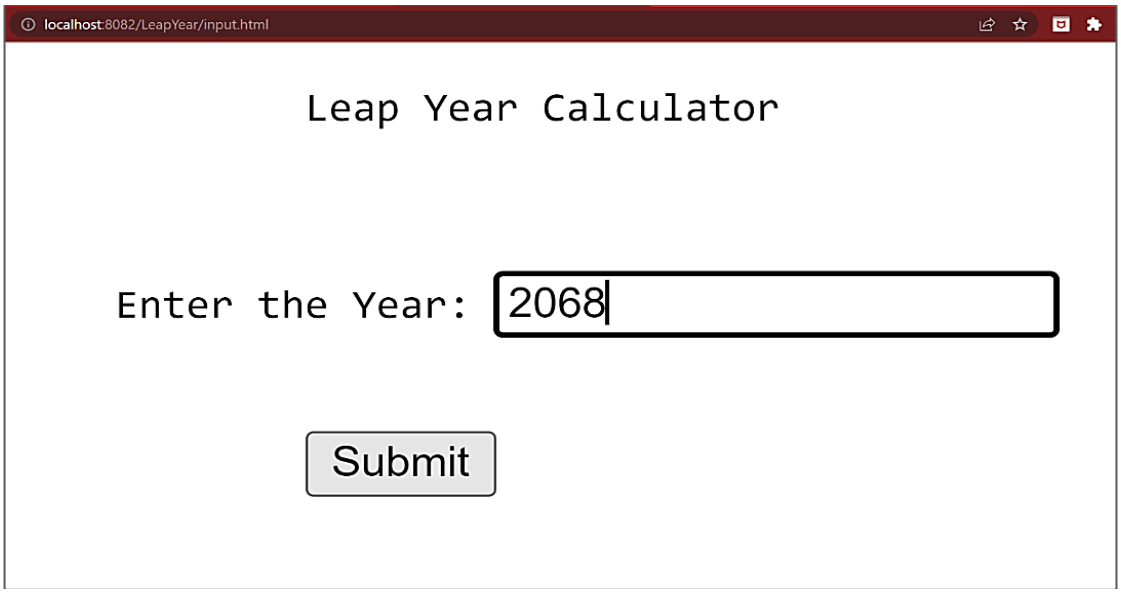
Output:

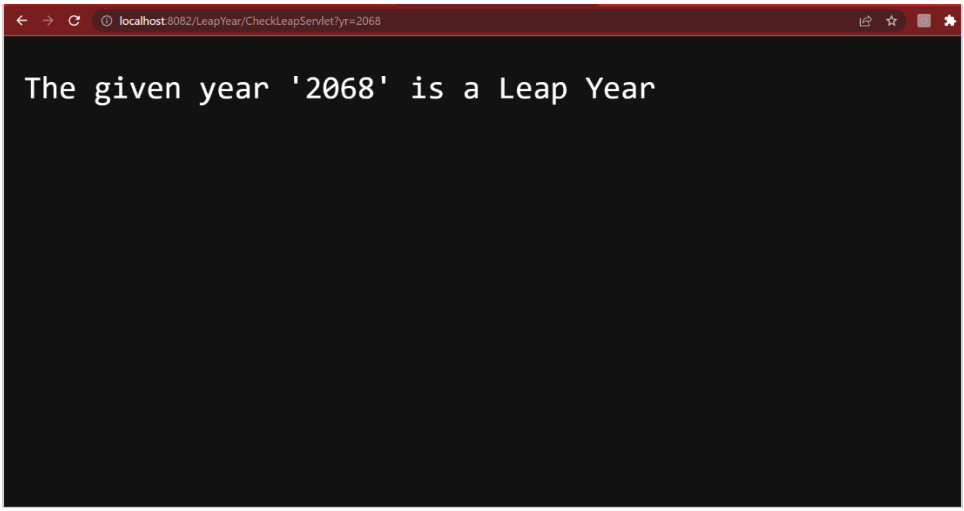

eclipse-workspace2 - LeapYear/src/main/webapp/input.html - Eclipse IDE
File Edit Source Navigate Search Project Run Window Help
Project Explorer
demo
LeapYear
Deployment Descriptor: LeapYear
JAX-WS Web Services
Java Resources
src/main/java
(default package)
CheckLeapServlet.java
Libraries
build
src
main
java
webapp
DiscussionForum
META-INF
WEB-INF
input.html
Servers
Tomcat v10.0 S...
display.html
HelloServlet.ja...
web.xml
input.html
CheckLeapServl...
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>LeapYearCheck</title>
6 </head>
7 <body>
8 <form method="get" action="CheckLeapServlet">
9 <pre>
10 Leap Year Calculator
11

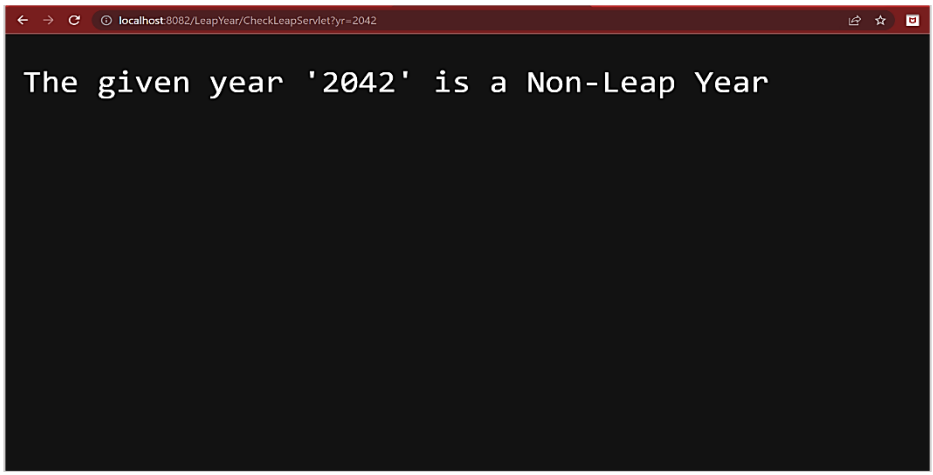
12 Enter the Year: <input type="text" name=yr>

13 <input type="submit">
14 </pre>
15 </form>
16 </body>
17 </html>
Outline
DOCTYPE:html
html
head
body
form action=CheckLeapServl
pre
Markers Properties Servers Data Source Explorer Snippets Terminal Console
Tomcat v10.0 Server at localhost [Stopped]


Project Explorer
demo
LeapYear
Deployment Descriptor: LeapYear
JAX-WS Web Services
Java Resources
src/main/java
(default package)
CheckLeapServlet.java
Libraries
build
src
main
java
webapp
DiscussionForum
META-INF
WEB-INF
input.html
Servers
Tomcat v10.0 S...
display.html
HelloServlet.ja...
web.xml
input.html
*CheckLeapServ...
26 * @see HttpServlet#doGet(HttpServletRequest request, HttpSe
27 */
28 protected void doGet(HttpServletRequest request, HttpServlet
29 PrintWriter pw=response.getWriter();
30 int year=Integer.parseInt(request.getParameter("yr"));
31 if ((year % 400 == 0)
32 || ((year % 4 == 0) && (year % 100 != 0))) {
33 pw.println("The given year '"+year+"' is a Leap
34 }
35
36 else {
37 pw.println("The given year '"+year+"' is a Non-L
38 }
39 //pw.println("<h1> Hello, "+s);
40 pw.close();
41 //response.getWriter().append("Served at: ").append(requ
42 }
Outline
CheckLeapServlet
serialVersionUID : long
CheckLeapServlet()
doGet(HttpServletRequest, HttpServletResp
doPost(HttpServletRequest, HttpServletResp
Markers Properties Servers Data Source Explorer Snippets Terminal Console
Tomcat v10.0 Server at localhost [Stopped]


localhost:8082/LeapYear/input.html
Leap Year Calculator
Enter the Year: 2068
Submit





Conclusion:
This assignment described the Web services approach to the Service Oriented Architecture concept. Also, described the Java APIs for programming Web services and demonstrated examples of their use by providing detailed step-by-step examples of how to program Web services in Java.

Submitted By: Shreya Mahajan

Roll No.: 4345

Class: B.E.I.T.

Staff Name: Ms. M. A. Rane/Mr. K. V. Patil

Staff Signature:

Date: