ASSIGNMENT No.7

TITLE: Create a simple web service and write any distributed application to consume the web service

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 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 popularity recently because it has 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:

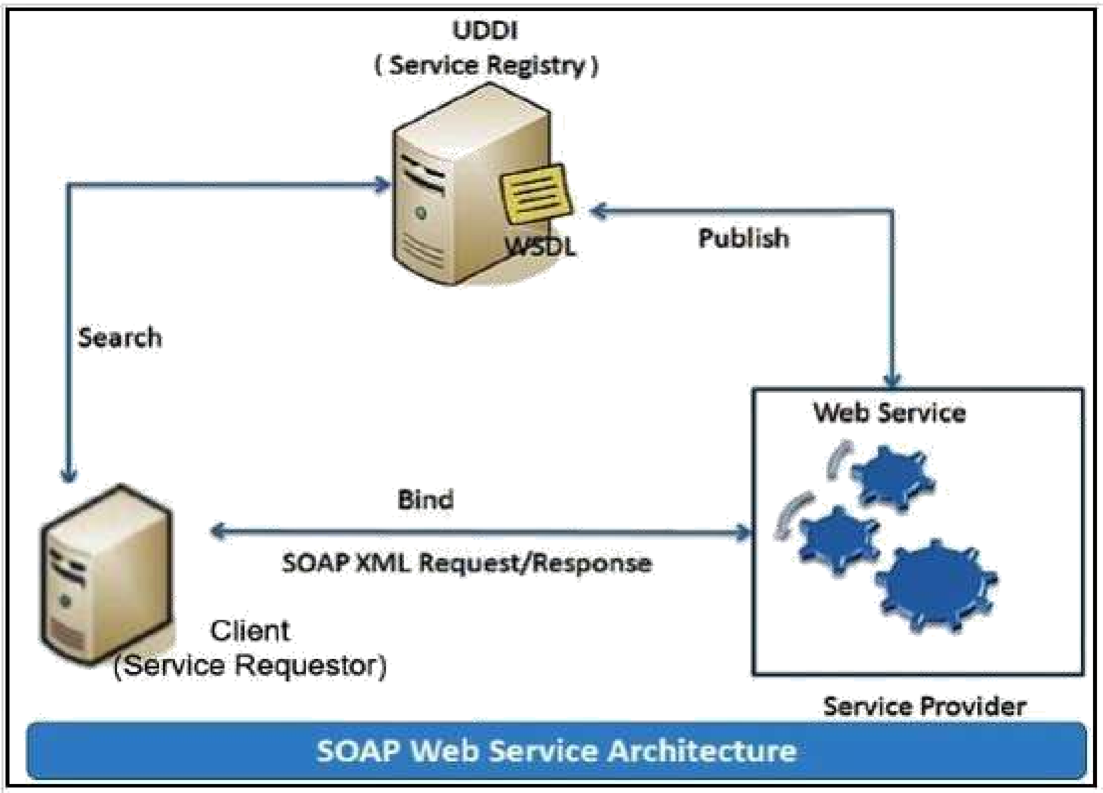
1. The client program that wants to interact with another application prepares its request content as a SOAP message.
2. Then, the client program sends this SOAP message to the server web service as an HTTP POST request with the content passed as the body of the request.
3. The web service plays a crucial 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:

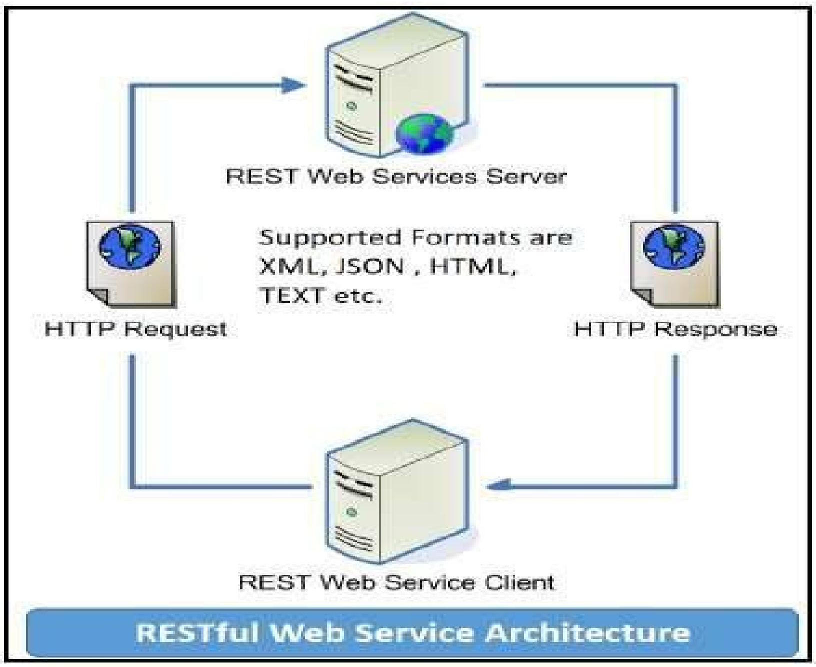
Universal Description, Discovery, and Integration (**UDDI**)*:* UDDI is an XMLbased framework for describing, discovering, and integrating web services. It acts as a directory of web service interfaces described in the WSDL language.

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:

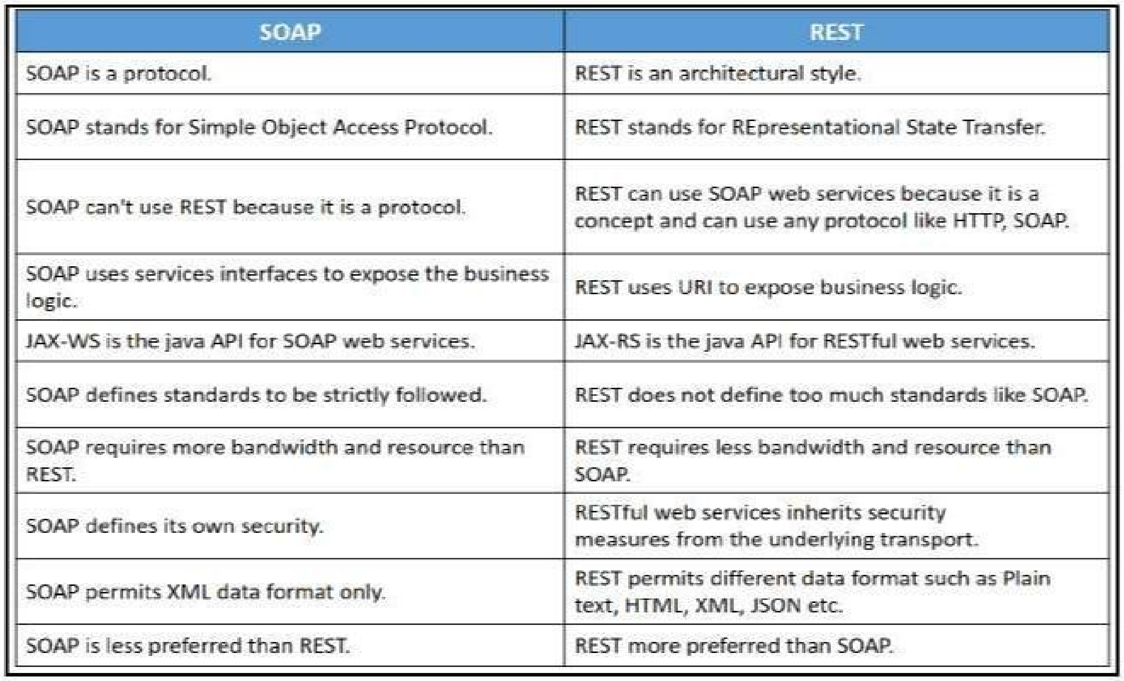


RESTful web services

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:


REST Web Services Server
Supported Formats are XML, JSON , HTML, TEXT etc.
HTTP Request
HTTP Response
REST Web Service Client
RESTful Web Service Architecture

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



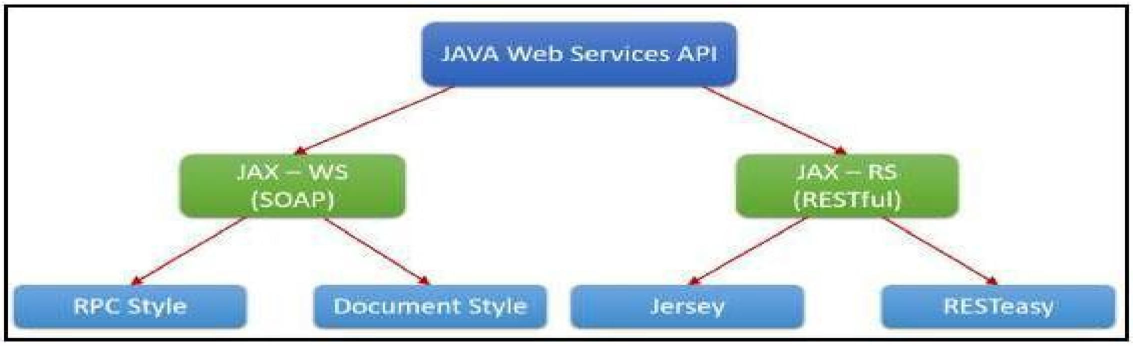
SOAP	REST
SOAP is a protocol.	REST is an architectural style.
SOAP stands for Simple Object Access Protocol.	REST stands for REpresentational State Transfer.
SOAP can't use REST because it is a protocol.	REST can use SOAP web services because it is a concept and can use any protocol like HTTP, SOAP.
SOAP uses services interfaces to expose the business logic.	REST uses URI to expose business logic.
JAX-WS is the java API for SOAP web services.	JAX-RS is the java API for RESTful web services.
SOAP defines standards to be strictly followed.	REST does not define too much standards like SOAP.
SOAP requires more bandwidth and resource than REST.	REST requires less bandwidth and resource than SOAP.
SOAP defines its own security.	RESTful web services inherits security measures from the underlying transport.
SOAP permits XML data format only.	REST permits different data format such as Plain text, HTML, XML, JSON etc.
SOAP is less preferred than REST.	REST more preferred than SOAP.

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 we don"t need to add any jars to work with them.



Students are required to implement both i.e. using SOAP and RESTful APIs.

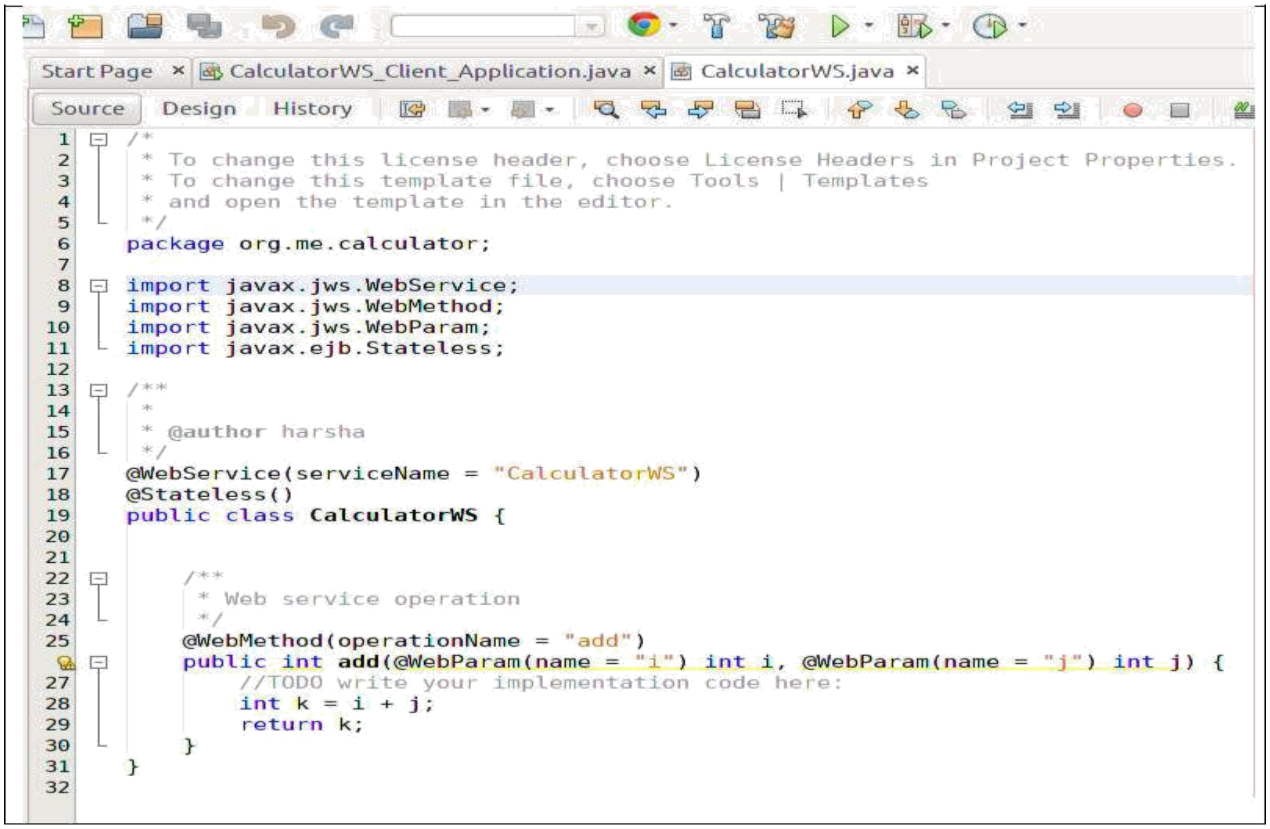
Implementing the solution:

1. **Creating a web service CalculatorWSApplication:**
   - Create New Project for `CalculatorWSApplication`.
   - Create a package `org.calculator`
   - Create class `CalculatorWS`.
   - Right-click on the `CalculatorWS` and create New Web Service.
   - IDE starts the glassfish server, builds the application and deploys the application on server.

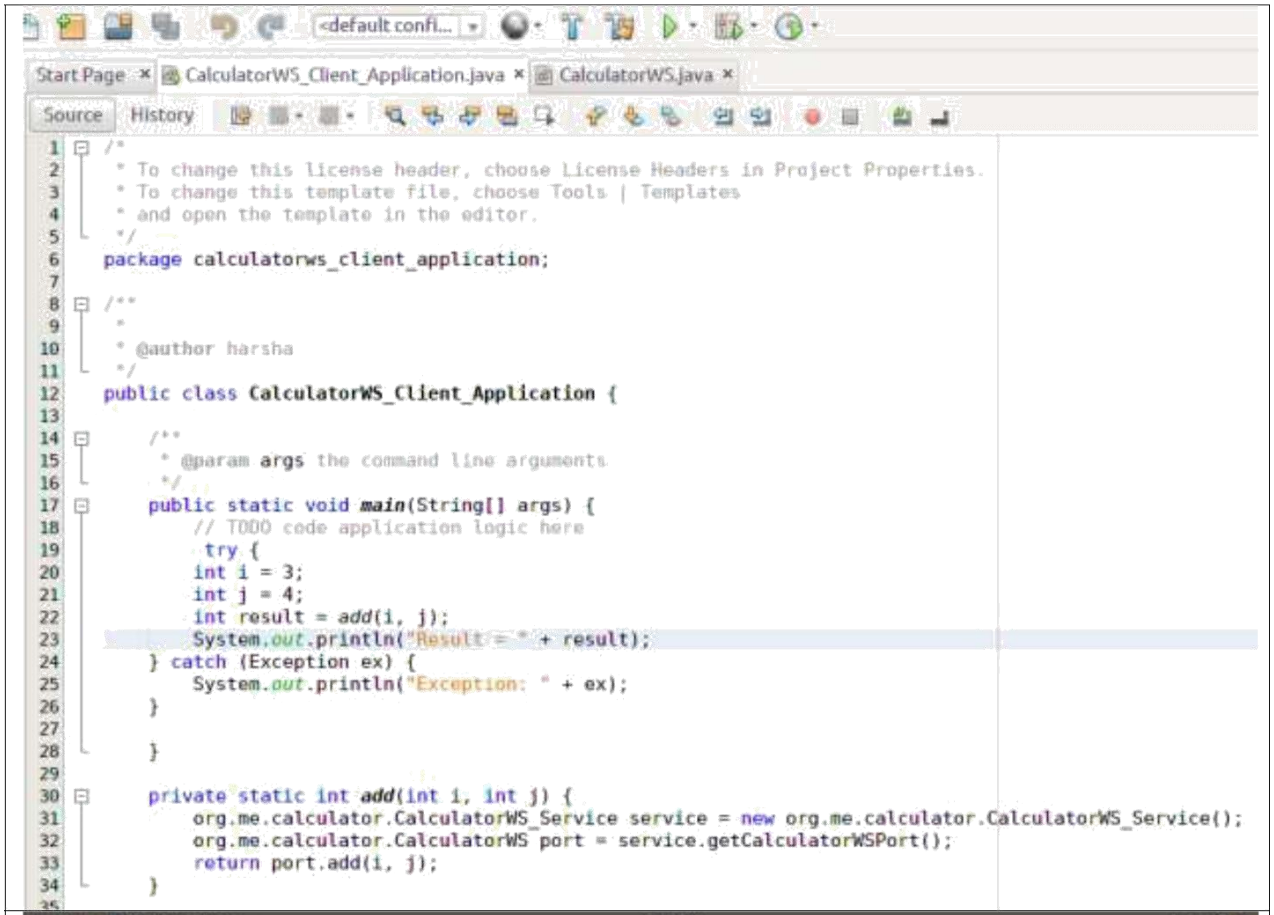
2. **Consuming the Webservice:**
   - Create a project with an `CalculatorClient`
   - Create package `org.calculator.client;`
   - add java class `CalculatorWS.java, addresponse.java, add.java, CalculatorWSService.java` and `ObjectFactory.java`

3. **Creating servlet in web application**
   - Create new jsp page for creating user interface.

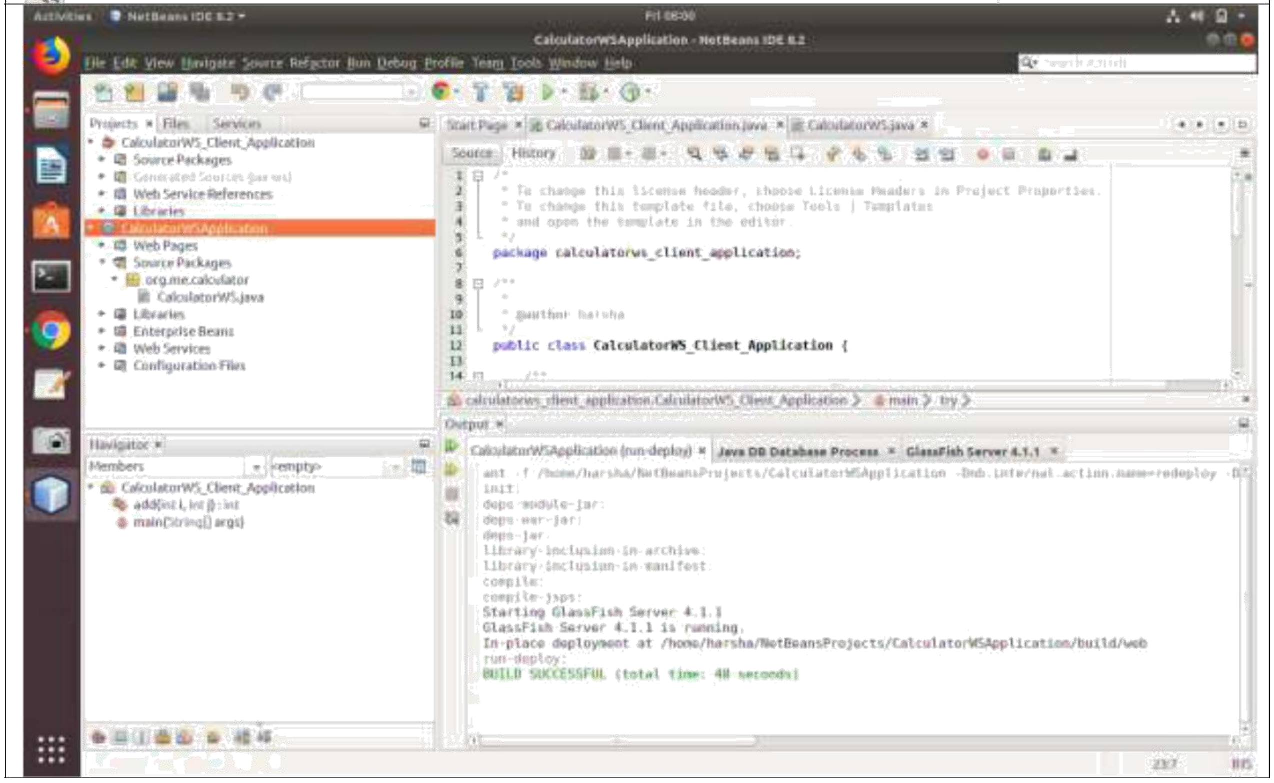
Writing the source code:



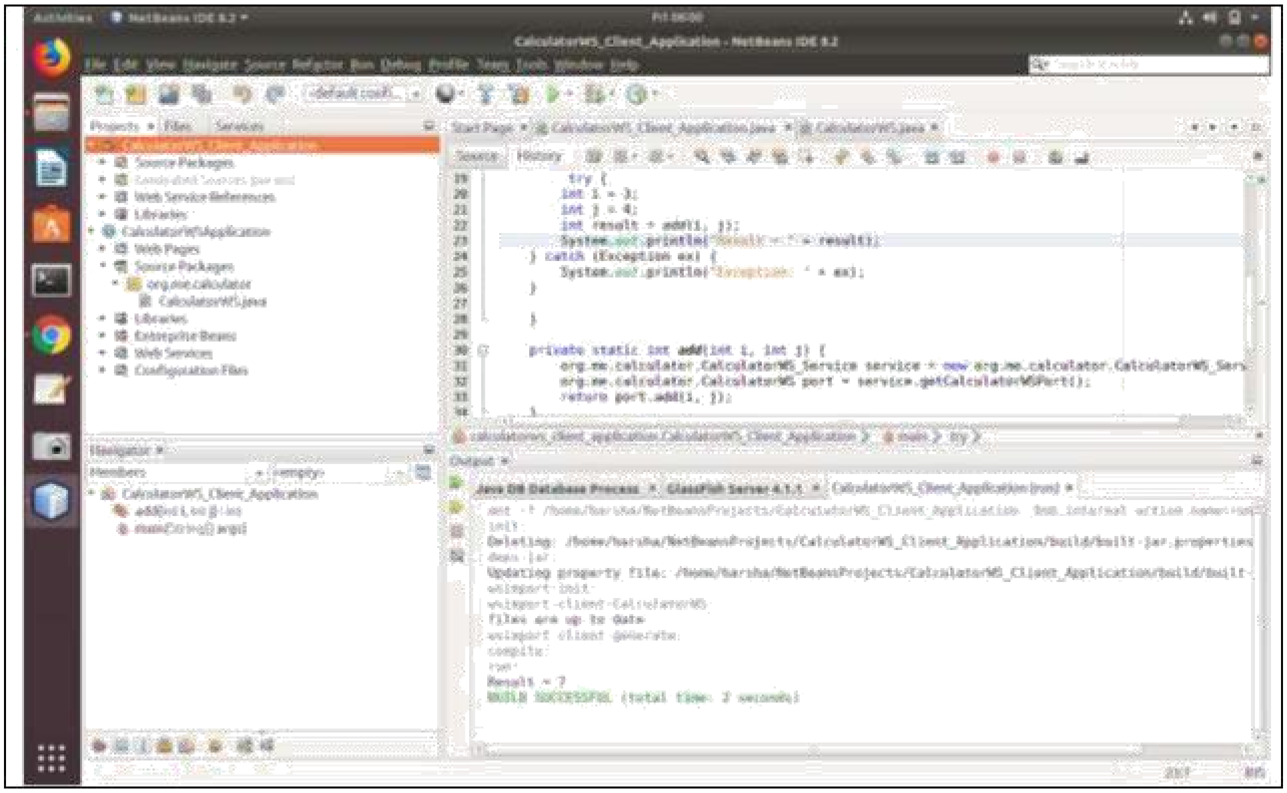
```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package org.me.calculator;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.ejb.Stateless;

/**
 *
 * @author harsha
 */
@WebService(serviceName = "CalculatorWS")
@Stateless()
public class CalculatorWS {


    /**
     * Web service operation
     */
    @WebMethod(operationName = "add")
    public int add(@WebParam(name = "i") int i, @WebParam(name = "j") int j) {
        //TODO write your implementation code here:
        int k = i + j;
        return k;
    }
}
```



```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package calculatorws_client_application;

/**
 *
 * @author harsha
 */
public class CalculatorWS_Client_Application {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
         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);
    }

    }

    private static int add(int i, int j) {
        org.me.calculator.CalculatorWS_Service service = new org.me.calculator.CalculatorWS_Service();
        org.me.calculator.CalculatorWS port = service.getCalculatorWSPort();
        return port.add(i, j);
    }
```



```
ant -f /home/harsha/NetBeansProjects/CalculatorWSApplication -Dnb.internal.action.name=redeploy -D
init:
deps-module-jar:
deps-ear-jar:
deps-jar:
library-inclusion-in-archive:
library-inclusion-in-manifest:
compile:
compile-jsps:
Starting GlassFish Server 4.1.1
GlassFish Server 4.1.1 is running.
In-place deployment at /home/harsha/NetBeansProjects/CalculatorWSApplication/build/web
run-deploy:
BUILD SUCCESSFUL (total time: 48 seconds)
```

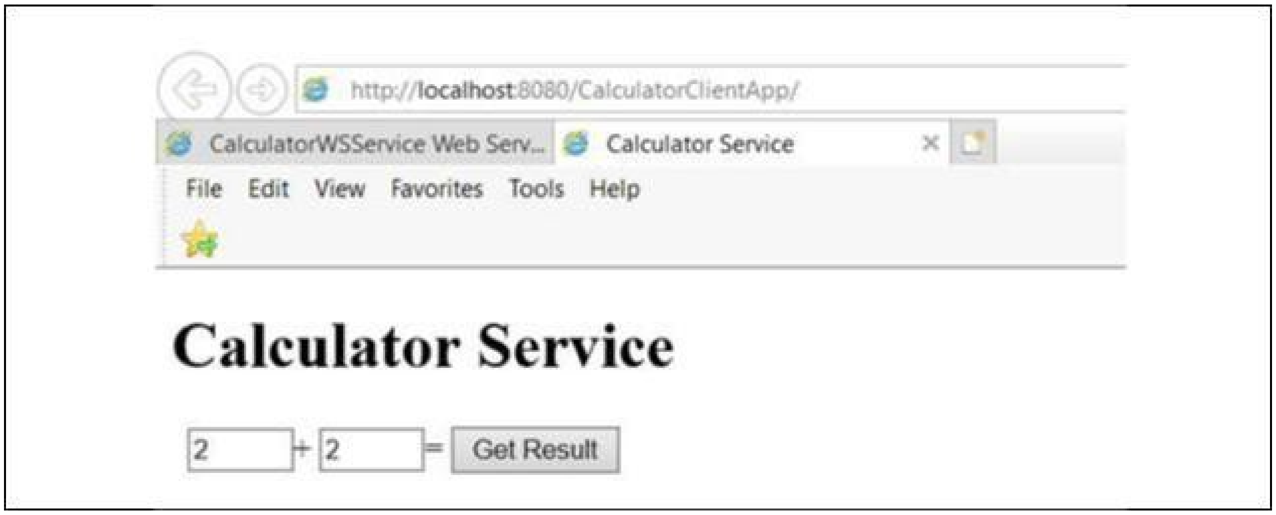


```
        try {
            int i = 3;
            int j = 4;
            int result = add(i, j);
            System.out.println("Result = " + result);
        } catch (Exception ex) {
            System.out.println("Exception: " + ex);
        }

    }

    private static int add(int i, int j) {
        org.me.calculator.CalculatorWS_Service service = new org.me.calculator.CalculatorWS_Serv
        org.me.calculator.CalculatorWS port = service.getCalculatorWSPort();
        return port.add(i, j);
    }
```

```
ant -f /home/harsha/NetBeansProjects/CalculatorWS_Client_Application -Dnb.internal.action.name=run
init:
Deleting: /home/harsha/NetBeansProjects/CalculatorWS_Client_Application/build/built-jar.properties
deps-jar:
Updating property file: /home/harsha/NetBeansProjects/CalculatorWS_Client_Application/build/built-
wsimport-init:
wsimport-client-CalculatorWS:
files are up to date
wsimport-client-generate:
compile:
run:
Result = 7
BUILD SUCCESSFUL (total time: 2 seconds)
```

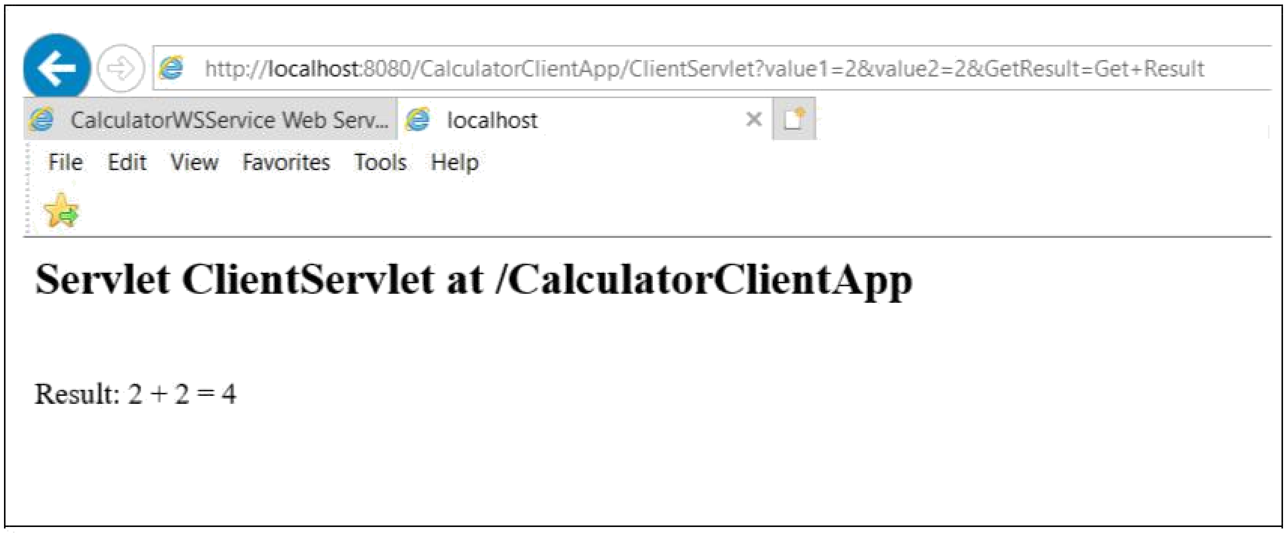
Compiling and Executing the solution:

Right Click on the Project and Choose Run.


http://localhost:8080/CalculatorClientApp/
CalculatorWSService Web Serv...
Calculator Service
File Edit View Favorites Tools Help

Calculator Service

2 + 2 = Get Result


http://localhost:8080/CalculatorClientApp/ClientServlet?value1=2&value2=2&GetResult=Get+Result
CalculatorWSService Web Serv...
localhost
File Edit View Favorites Tools Help

Servlet ClientServlet at /CalculatorClientApp

Result: 2 + 2 = 4

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.