Practical 2

Create a Simple SOAP service.

**Software Tools Required:**

* **Code Editor:** Eclipse IDE
* **Build Automation Tool:** Apache Maven
* **SOAP Testing Tool:** SOAPUI
* **Framework:** Apache CXF
* **Programming Language:** Java

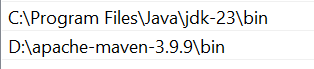
**Downloads Required:**

* Apache Maven: [Download Apache Maven – Maven](https://maven.apache.org/download.cgi)
* JDK: [Java Downloads | Oracle India](https://www.oracle.com/in/java/technologies/downloads/#jdk23-windows) (x64 MSI Installer)
* Eclipse IDE: [Eclipse downloads - Select a mirror | The Eclipse Foundation](https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2024-09/R/eclipse-inst-jre-win64.exe)
* SOAP Testing Tool: [Download REST & SOAP Automated API Testing Tool | Open Source | SoapUI](https://www.soapui.org/downloads/soapui/) (SoapUI Open Source)

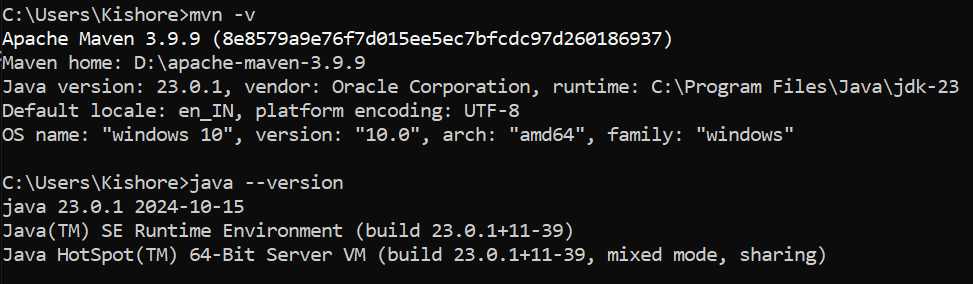
After Downloading JDK and Maven, Set the Path in Environment Variable in User Variables > Path > Edit > New and Paste the Path for JDK and Maven and check the versions for both in command prompt.

**Demonstration:**

Path Set



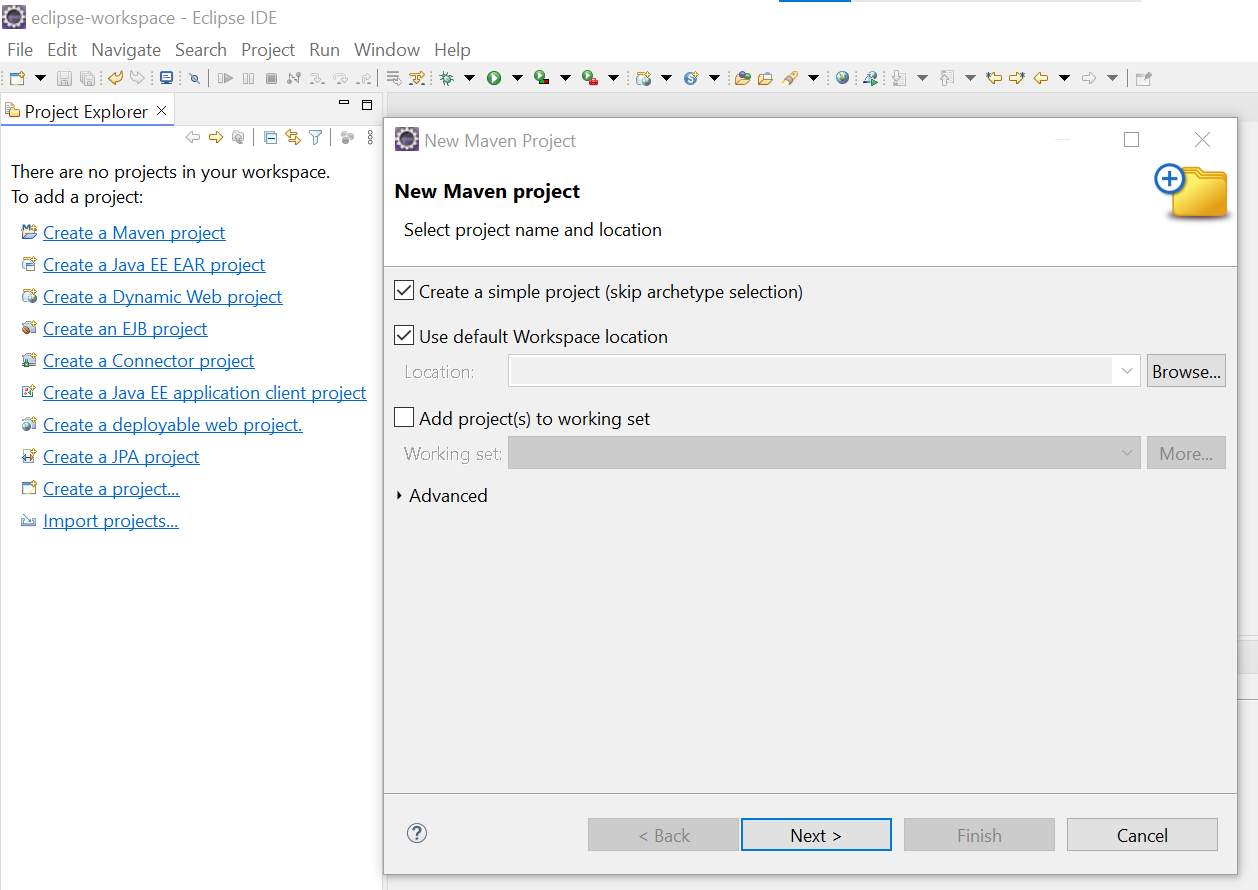
Version checked in command prompt



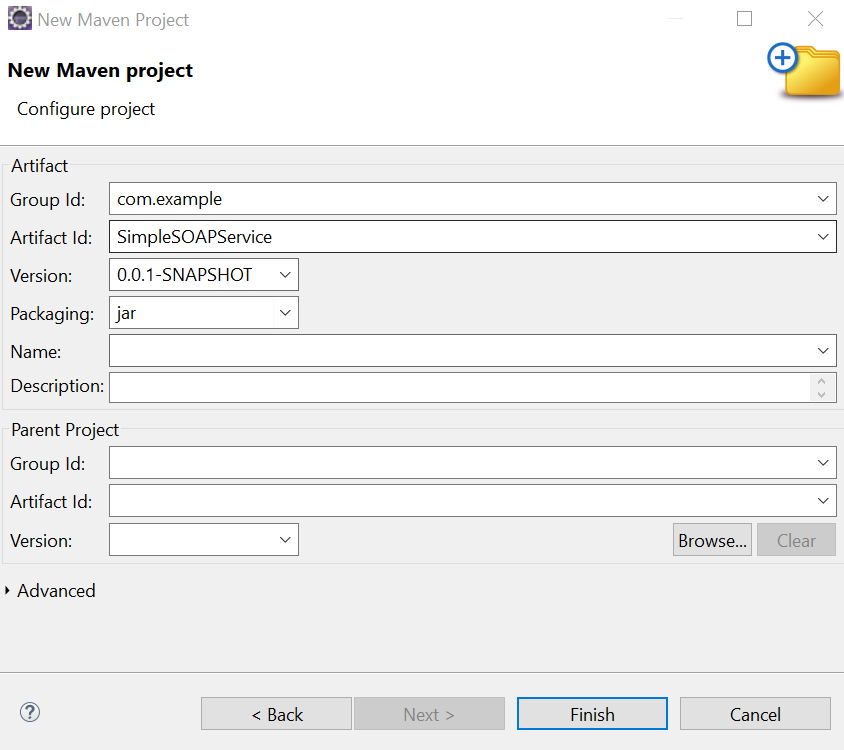
**Java Programming Language**

**Project Name:** SimpleSOAPService

**Step 1:** Open Eclipse IDE, Create a Maven Project, Check – Create a simple project (skip archetype selection) and Click Next.



**Step 2:** In New Maven Project window add the following details – Group id: com.example, Artifact Id: SimpleSOAPService and Version: 0.0.1- SNAPSHOT and click Finish



**Step 3:** After creating the Maven project, update the pom.xml to include the necessary dependencies for Apache CXF and JAX-WS. Here’s the full pom.xml file.

**pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>SimpleSOAPService</artifactId>

<version>0.0.1-SNAPSHOT</version>

**<dependencies>**

**<!-- Apache CXF for JAX-WS -->**

**<dependency>**

**<groupId>org.apache.cxf</groupId>**

**<artifactId>cxf-rt-frontend-jaxws</artifactId>**

**<version>3.4.5</version>**

**</dependency>**

**<!-- Apache CXF HTTP Transport -->**

**<dependency>**

**<groupId>org.apache.cxf</groupId>**

**<artifactId>cxf-rt-transports-http</artifactId>**

**<version>3.4.5</version>**

**</dependency>**

**<!-- Apache CXF HTTP Jetty Transport (for embedded HTTP server) -->**

**<dependency>**

**<groupId>org.apache.cxf</groupId>**

**<artifactId>cxf-rt-transports-http-jetty</artifactId>**

**<version>3.4.5</version>**

**</dependency>**

**<!-- Jakarta XML WS (JAX-WS API) -->**

**<dependency>**

**<groupId>jakarta.xml.ws</groupId>**

**<artifactId>jakarta.xml.ws-api</artifactId>**

**<version>2.3.3</version>**

**</dependency>**

**<!-- Java Annotations -->**

**<dependency>**

**<groupId>javax.annotation</groupId>**

**<artifactId>javax.annotation-api</artifactId>**

**<version>1.3.2</version>**

**</dependency>**

**</dependencies>**

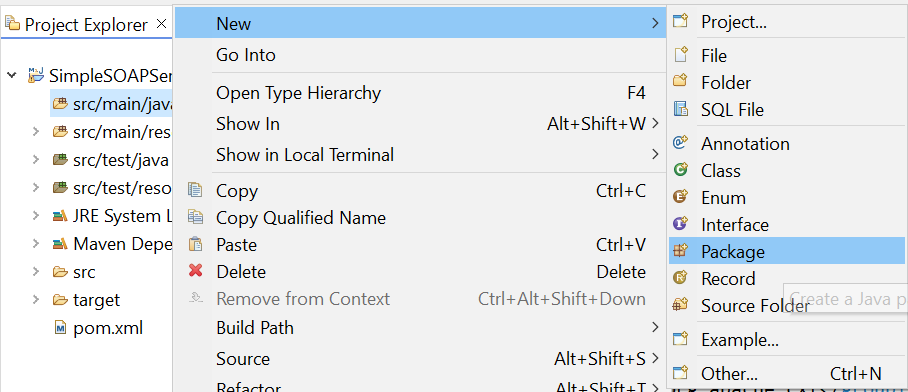
</project>

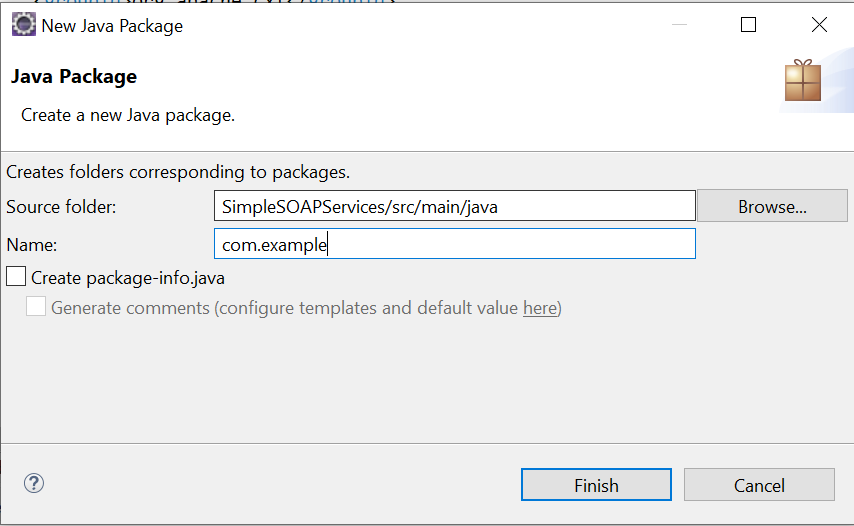
After adding dependencies Save it by Ctrl+S

**Step 4:** Create the Service Interface

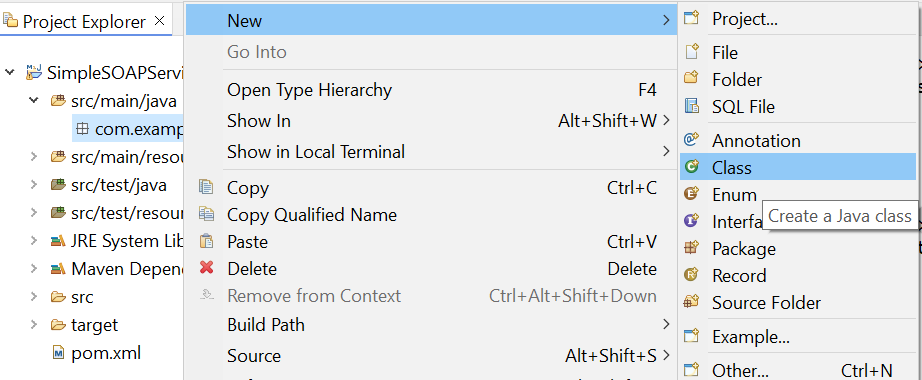
Now, create a simple SOAP service interface to define the operations.

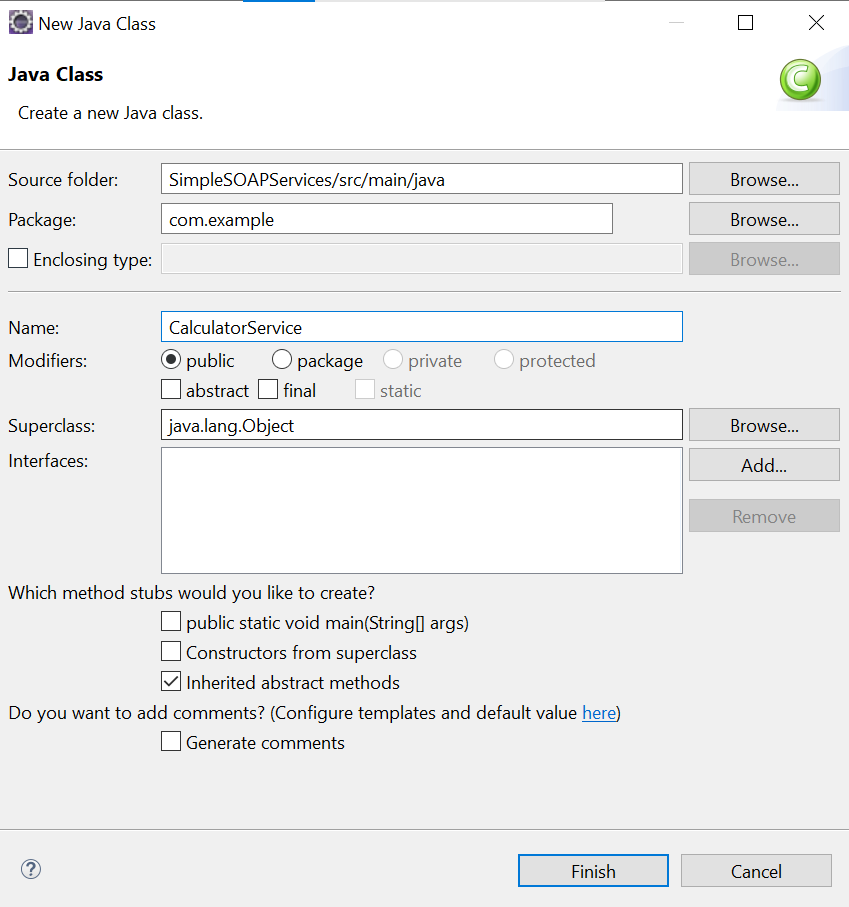
1. **Right-click** on src/main/java → **New** → **Package** and name it com.example





1. **Right-click** on the com.example package → **New** → **Class** and name it CalculatorService.java.





**File Name**: CalculatorService.java

**Code:**

**package** com.example;

**import** javax.jws.WebMethod;

**import** javax.jws.WebService;

@WebService

**public** **interface** CalculatorService {

@WebMethod

**int** add(**int** num1, **int** num2);

@WebMethod

**int** subtract(**int** num1, **int** num2);

}

**Save it**

**Step 5:** Implement the Service

Now, create a class that implements the CalculatorService interface.

1. **Right-click** on com.example → **New** → **Class** and name it CalculatorServiceImpl.java.
2. Add the following code to CalculatorServiceImpl.java:

**File Name:** CalculatorServiceImpl.java

**Code:**

**package** com.example;

**import** javax.jws.WebService;

@WebService(endpointInterface = "com.example.CalculatorService")

**public** **class** CalculatorServiceImpl **implements** CalculatorService {

@Override

**public** **int** add(**int** num1, **int** num2) {

**return** num1 + num2;

}

@Override

**public** **int** subtract(**int** num1, **int** num2) {

**return** num1 - num2;

}

} **Save It**

**Step** **6:** Create the SOAP Server

Next, create a class to publish the SOAP service.

1. **Right-click** on com.example → **New** → **Class** and name it SOAPServer.java.
2. Add the following code to SOAPServer.java:

**File Name:** SOAPServer.java

**Code:** package com.example;

import javax.xml.ws.Endpoint;

public class SOAPServer {

public static void main(String[] args) {

// Create an instance of the service implementation

CalculatorServiceImpl implementor = new CalculatorServiceImpl();

// Publish the service

String address = "http://localhost:8080/CalculatorService";

Endpoint.publish(address, implementor);

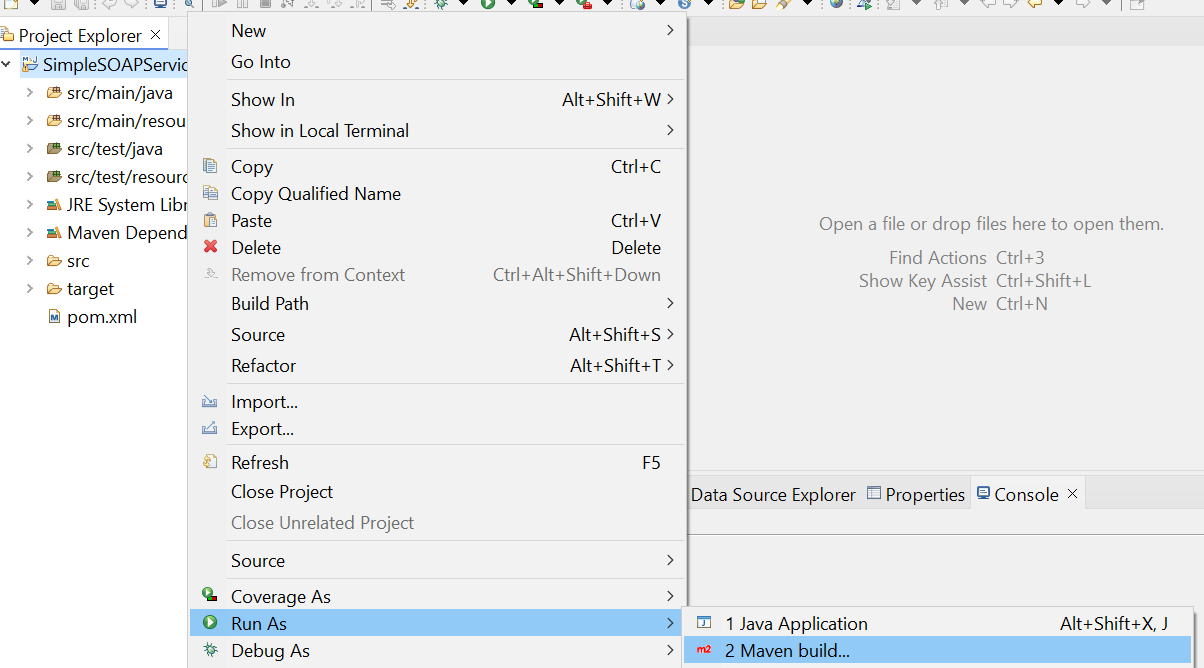
System.out.println("SOAP Service started at " + address);

    }

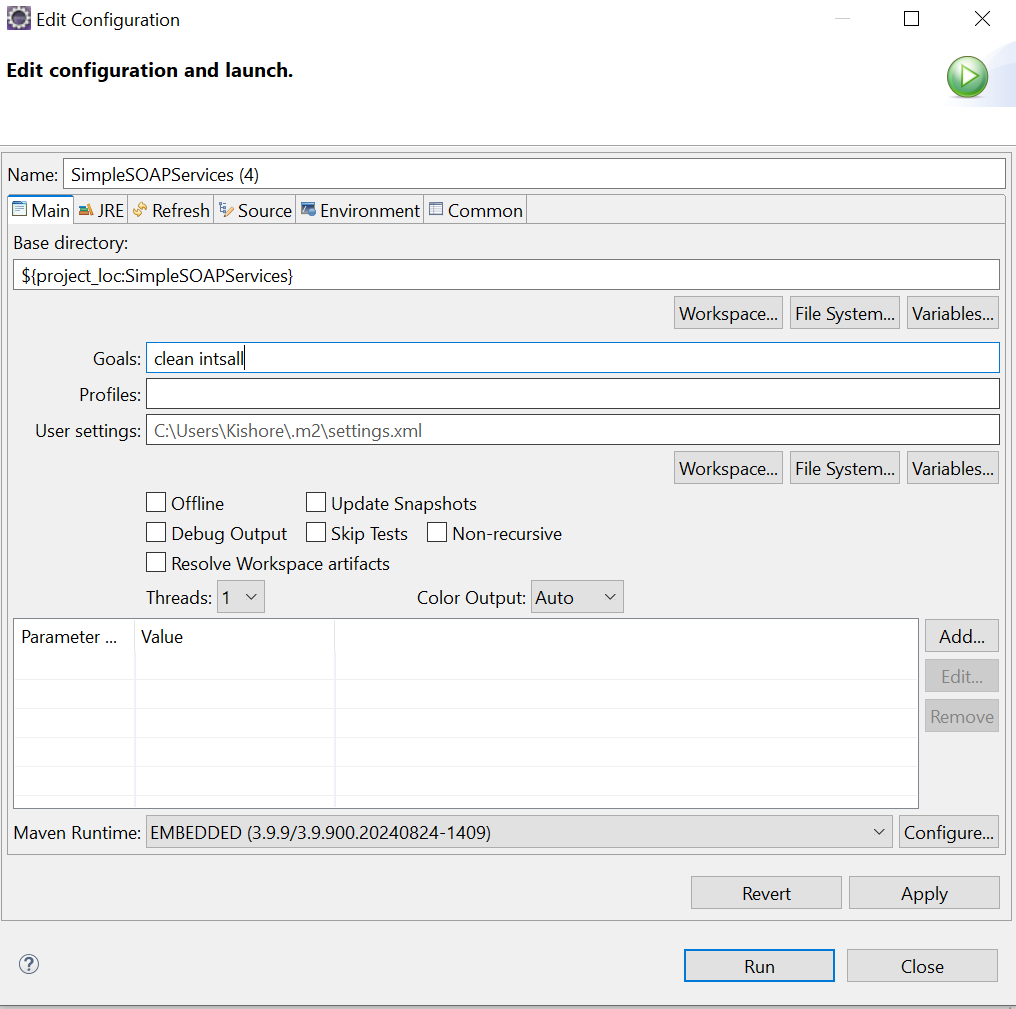
}  **Save It**

**Output:**

**mvn clean install:** The clean install command is commonly used in Maven to build and install a project. The purpose of running clean is to remove any previously compiled code or artifacts, ensuring that the next build starts fresh without using any old files that might cause conflicts or errors.

****

In Goal type clean install and run

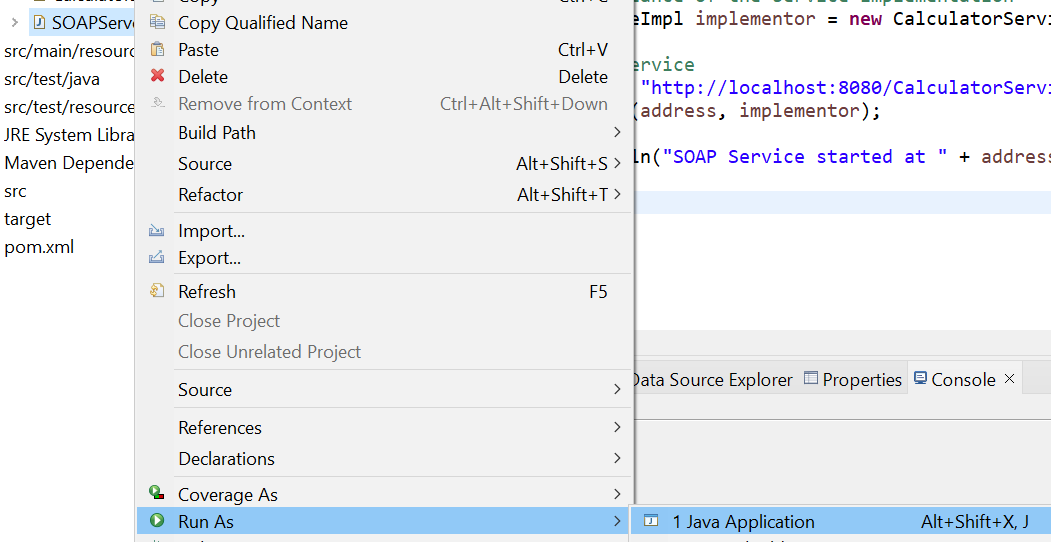
****

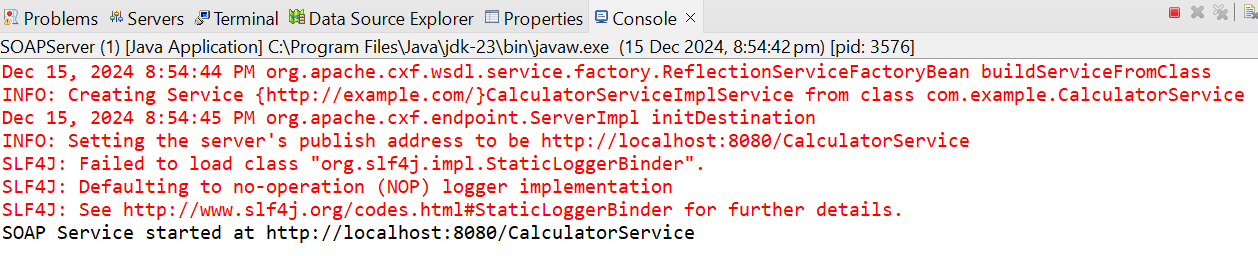
****

**Run the Application**

1. Right-click on SOAPServer.java → Run As → Java Application.
2. You should see the following output:

SOAP Service started at <http://localhost:8080/CalculatorService>

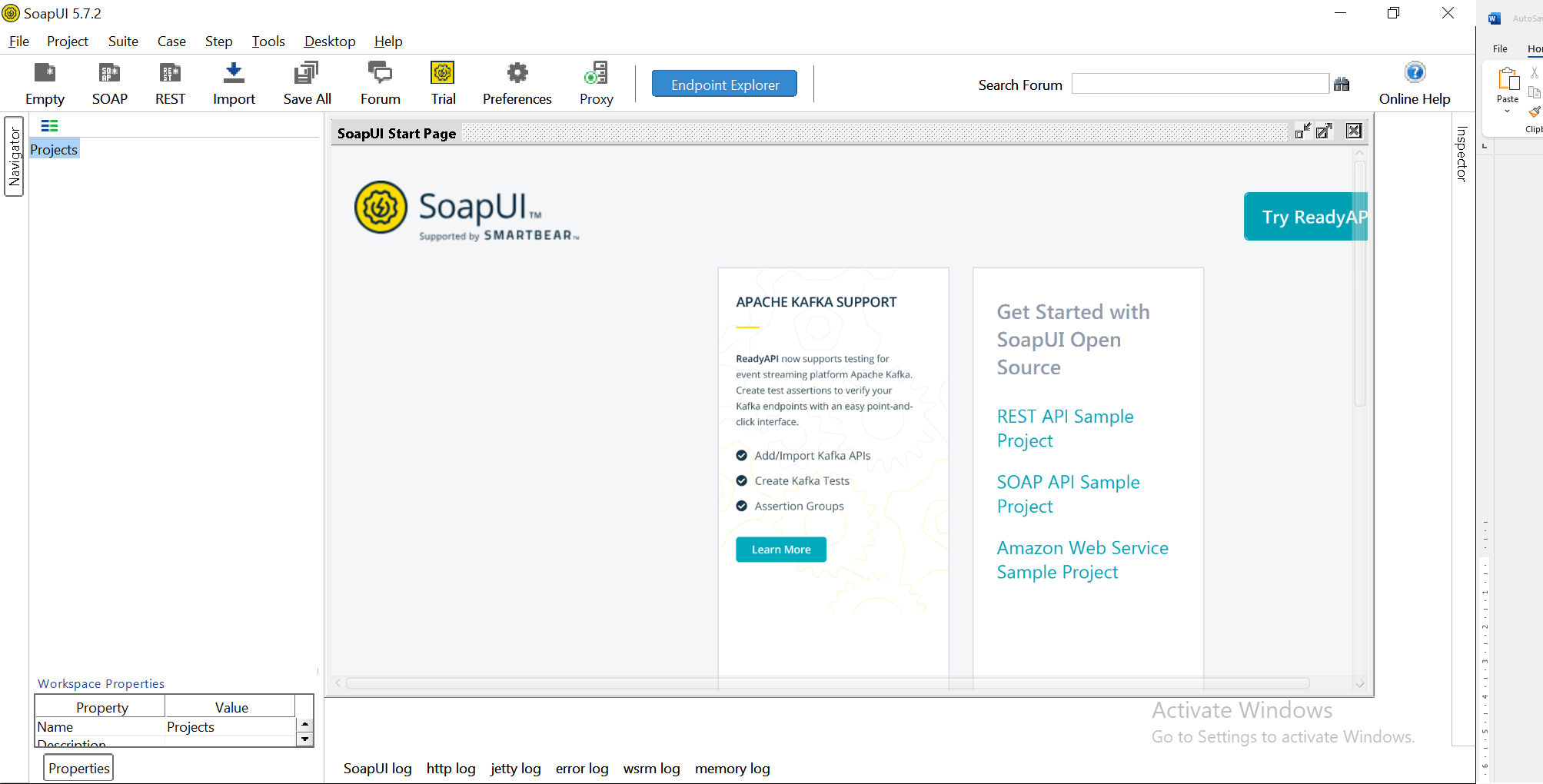
****

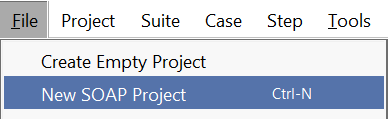
****

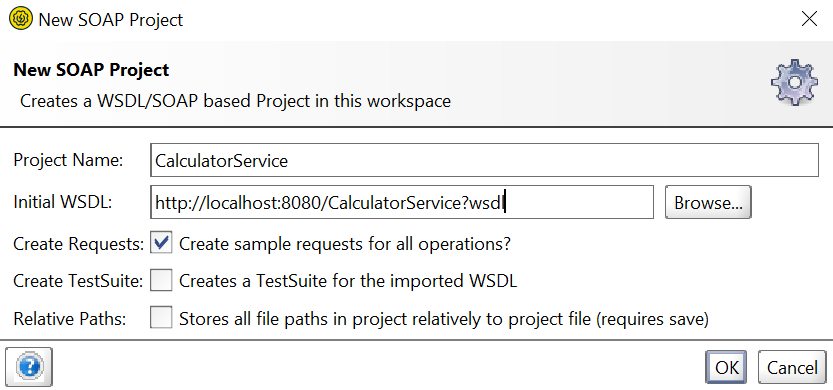
**Step 7.** Test the SOAP Service Using SOAPUI

1. Open SOAPUI and create a new project:
   1. Click File → New SOAP Project.
   2. In the Project Name field, enter CalculatorService.
   3. In the Initial WSDL field, enter: **http://localhost:8080/CalculatorService?wsdl**

**SOAPUI:**

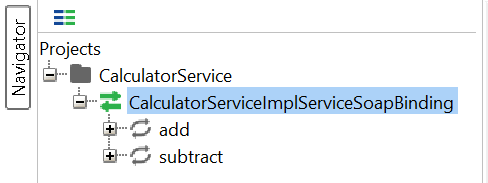
****

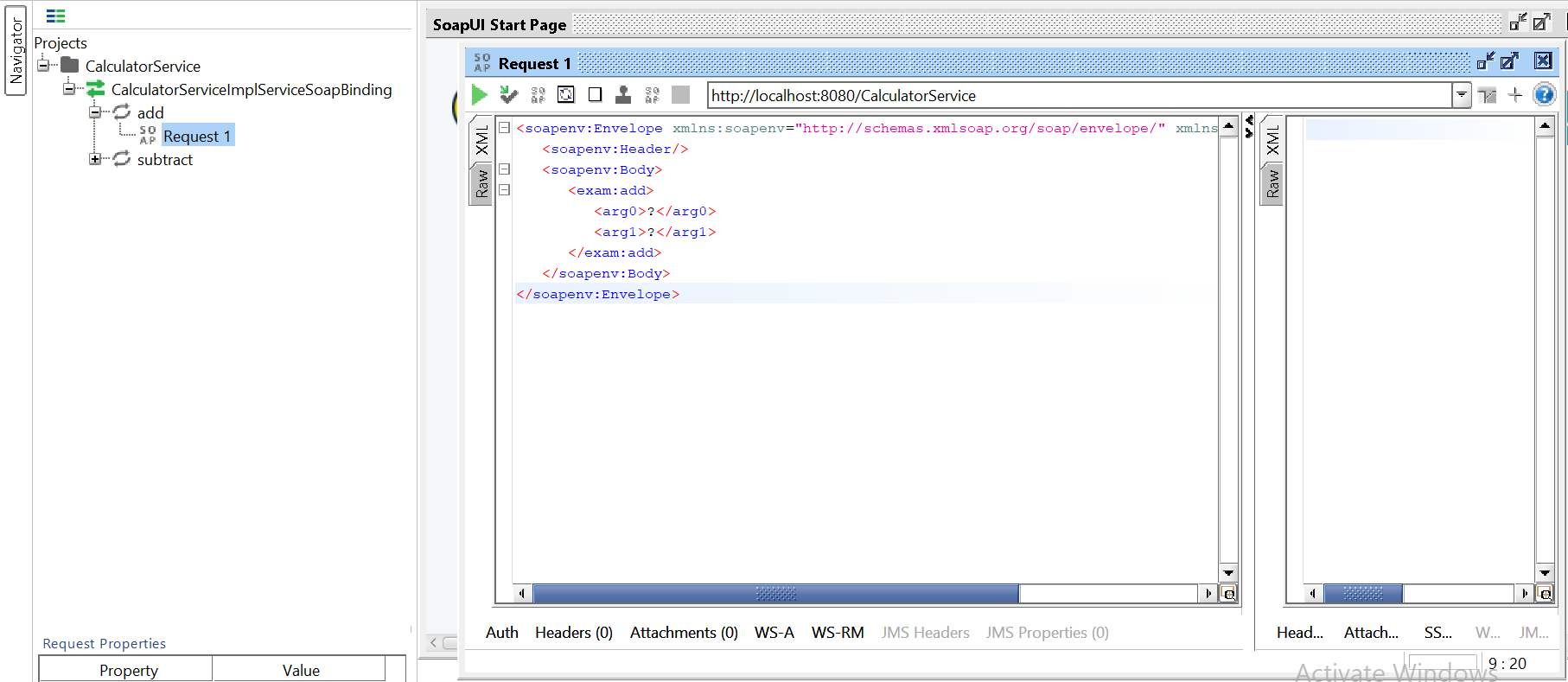
****

****

**Click OK**

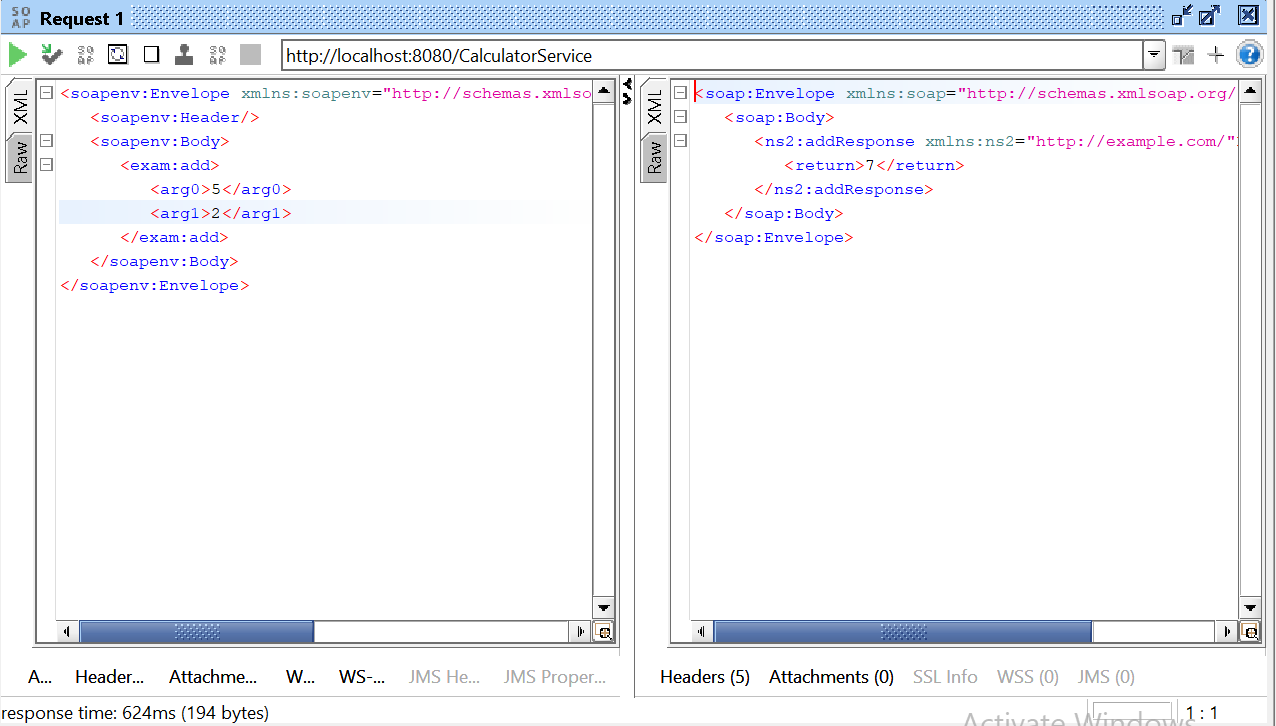
SOAPUI will import the WSDL and display the operations (add and subtract). You can now test the SOAP service by right-clicking on the operations and sending requests.

****

****

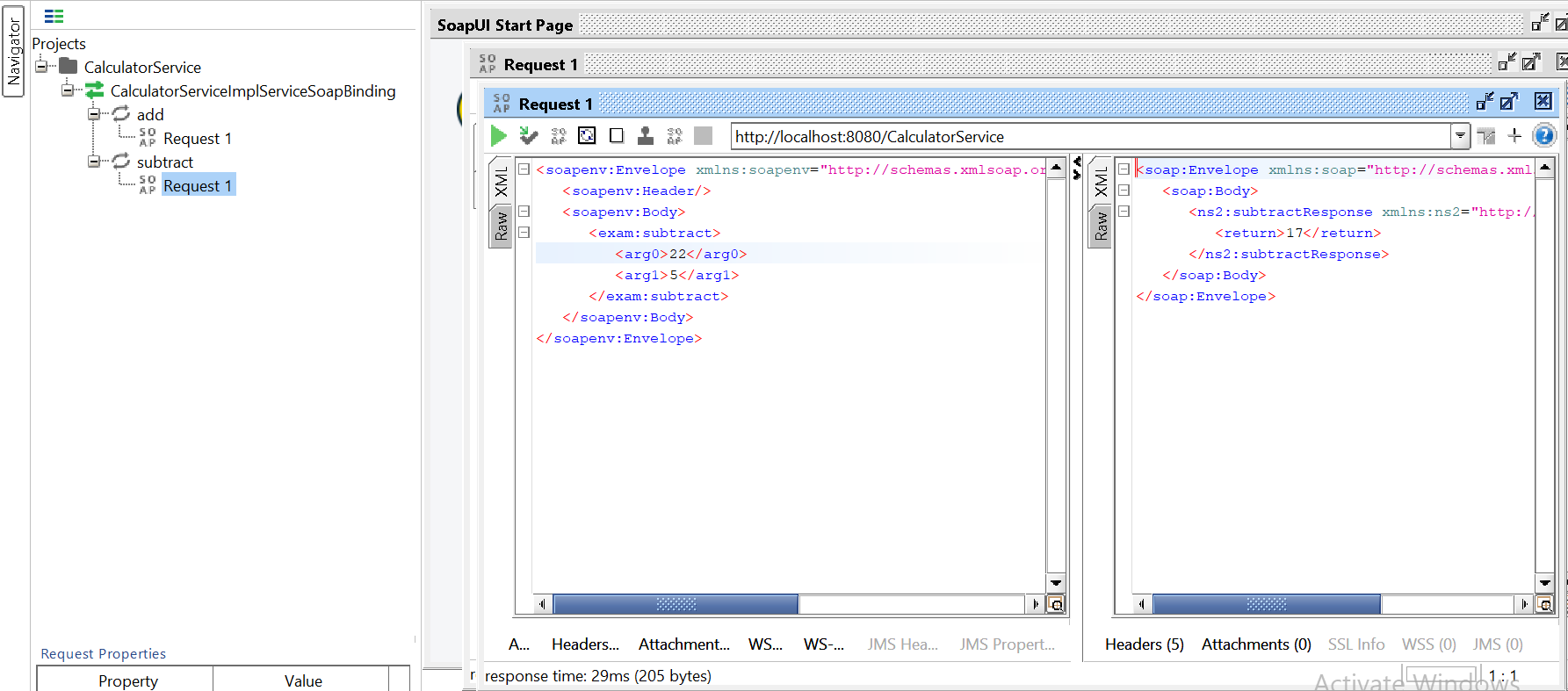
**add > Request 1**

Here, in add we got the request and edited 5 and 2 in argument tags and output displayed in xml.

****

**subtract > Request 1**

Here, in subtract we got the request and edited 22 and 5 in argument tags and output displayed in xml.

****

**Conclusion:** Why Use SoapUI to Create SOAP Services and Get Output in XML for Add and Subtract Operations

SoapUI is a popular tool for testing and interacting with SOAP (Simple Object Access Protocol) web services. It is used to create and test SOAP services, allowing developers and testers to ensure that their web services function correctly and handle various input scenarios.

Logic to Work with SOAP Services in SoapUI:

Creating SOAP Service:

* WSDL (Web Services Description Language): SoapUI uses the WSDL file to generate the request and response structure for SOAP services. The WSDL defines the operations, input/output messages, and service endpoint.
* SOAP Request and Response: In SoapUI, you can create SOAP requests based on the operations defined in the WSDL. For operations like "Add" and "Subtract," the SOAP request will include the parameters (e.g., two numbers) to be processed by the web service.

-------------------------------