

Lab Assignment No. 07

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

Objectives:

- Develop a Simple Web Service:
- Create a Distributed Application:
- Learn about Web Services:
- Implement Communication Between Distributed Components:
- Ensure Scalability and Reliability:

Theory:

➤ Web Services Overview:

A web service is a system designed to support interoperable machine-to-machine interaction over a network. It allows applications to communicate with each other over HTTP using standard protocols such as REST (Representational State Transfer) or SOAP (Simple Object Access Protocol). The service exposes various endpoints, and applications can invoke those services using HTTP methods like GET, POST, PUT, DELETE, etc.

Key Concepts of Web Services:

1. **RESTful Services:** REST is an architectural style that uses HTTP and focuses on simplicity. A RESTful service uses HTTP methods (GET, POST, etc.) and URL patterns to define resources (such as /users, /products, etc.).
2. **JSON/XML:** Web services often use JSON (JavaScript Object Notation) or XML to structure the data being exchanged between systems.
3. **Statelessness:** Web services should be stateless, meaning each request from the client should contain all the information necessary to understand and process the request.

➤ Distributed Systems Overview:

A **distributed system** is a network of independent computers that appears to its users as a single coherent system. These systems share resources and communicate over a network.

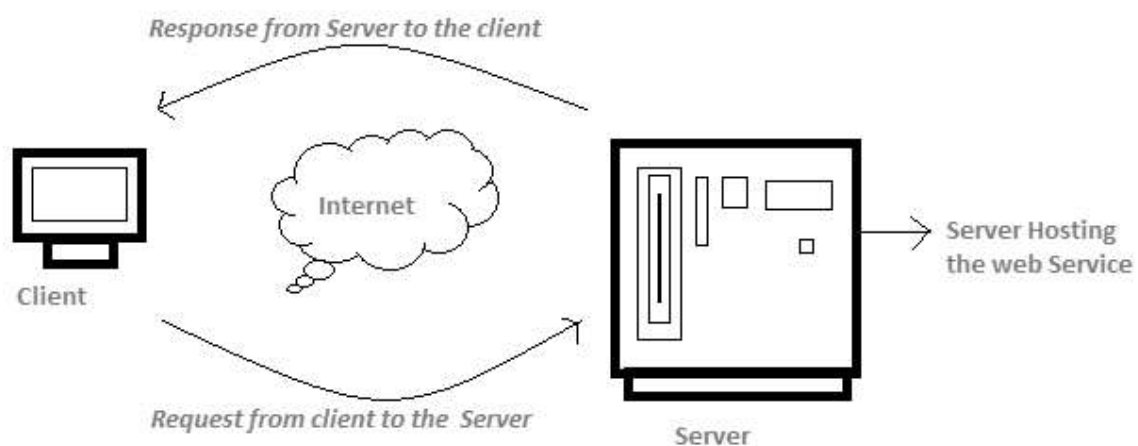
Distributed applications are built with components running on different machines that interact to complete tasks.

In distributed systems:

- **Communication** occurs through remote procedure calls (RPCs) or web service calls.
- **Data Storage** is often replicated across multiple nodes to ensure fault tolerance and high availability.
- **Load Balancing** is used to distribute workloads efficiently among the system nodes.

➤ **Client-Server Model:**

In the client-server architecture, a client (consumer) sends requests to the server (provider), which processes the request and returns a response. The server typically hosts the web service and provides a set of APIs (application programming interfaces). The client is responsible for sending requests and processing the responses.



➤ **Distributed Application Design:**

Distributed applications are built using multiple independent systems (servers, databases, etc.) that work together. Key considerations include:

- **Scalability:** Ensuring the system can handle increased loads by adding more resources.
- **Fault Tolerance:** Ensuring the system remains functional even if some components fail.
- **Concurrency:** Handling multiple requests simultaneously in a distributed manner.

Applications:

- **Web Services:**
 - E-commerce: Payment gateways, inventory systems integration.
 - Cloud Computing: Managing cloud resources (e.g., AWS, Azure).
 - Mobile Apps: Fetching data from backend servers (e.g., weather data).
 - Enterprise Integration: Connecting internal systems with external services.
- **Distributed Applications:**
 - Microservices: Decoupled services communicating via APIs.
 - Real-time Data Processing: Systems for stock trading or IoT.
 - Cloud Apps: Scalable, fault-tolerant services (e.g., Google Docs).
 - Gaming: Multiplayer games using distributed servers.
 - Big Data: Large-scale data processing and analysis (e.g., Apache Hadoop).

Advantages:

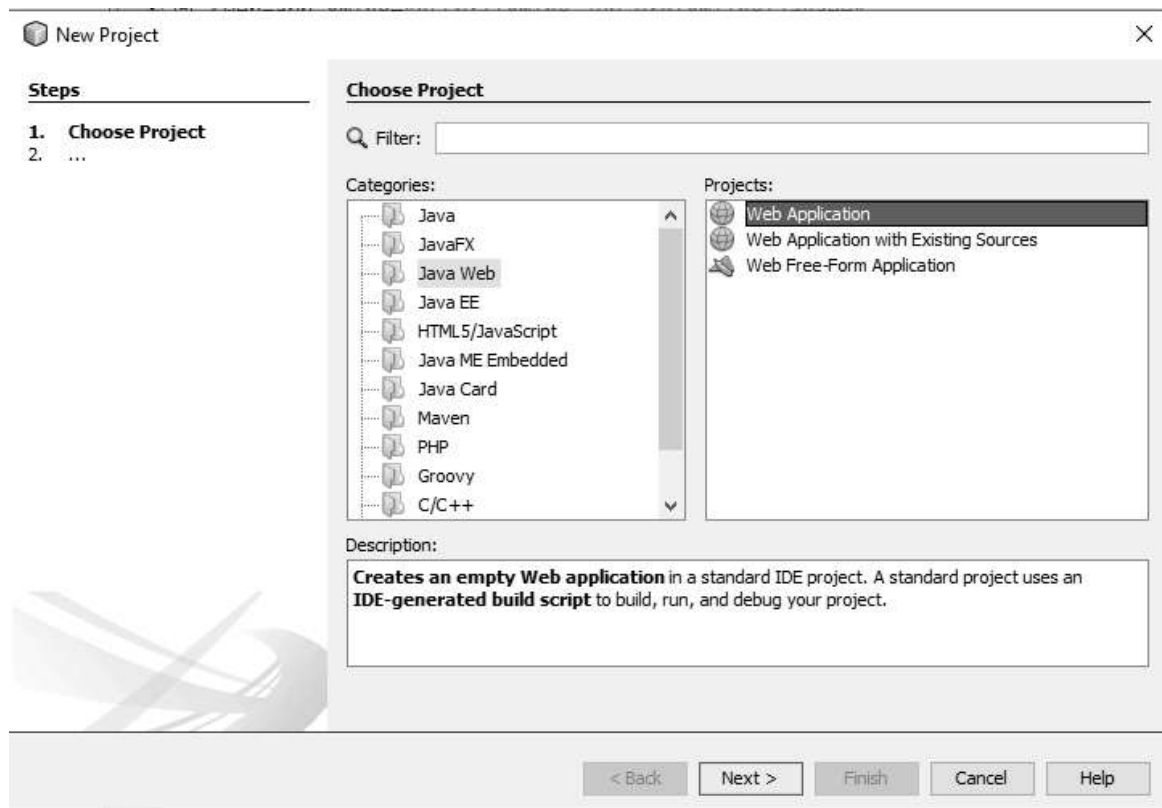
- **Web Services:**
 - Interoperability: Cross-platform communication.
 - Scalability: Easy to scale by adding servers.
 - Loose Coupling: Independent systems with consistent APIs.
 - Flexibility: Supports multiple data formats (JSON, XML).
- **Distributed Applications:**
 - Scalability: Easy to add resources as demand increases.
 - Fault Tolerance: Systems remain functional even if parts fail.
 - Performance: Distributed workloads improve efficiency.
 - Real-time Processing: Suitable for applications like gaming or financial services.

Disadvantages:

- **Web Services:**
 - Latency: Delays in network communication.
 - Security Risks: Exposure of sensitive data.
 - Complex Error Handling: Managing errors across multiple services.
 - Performance Overheads: Parsing data formats can slow down response times.
- **Distributed Applications:**
 - Complexity: Difficult to develop and maintain.
 - Network Dependency: Vulnerable to network issues.
 - Consistency Issues: Ensuring data consistency can be challenging.
 - Resource Overheads: High resource consumption for communication and synchronization.

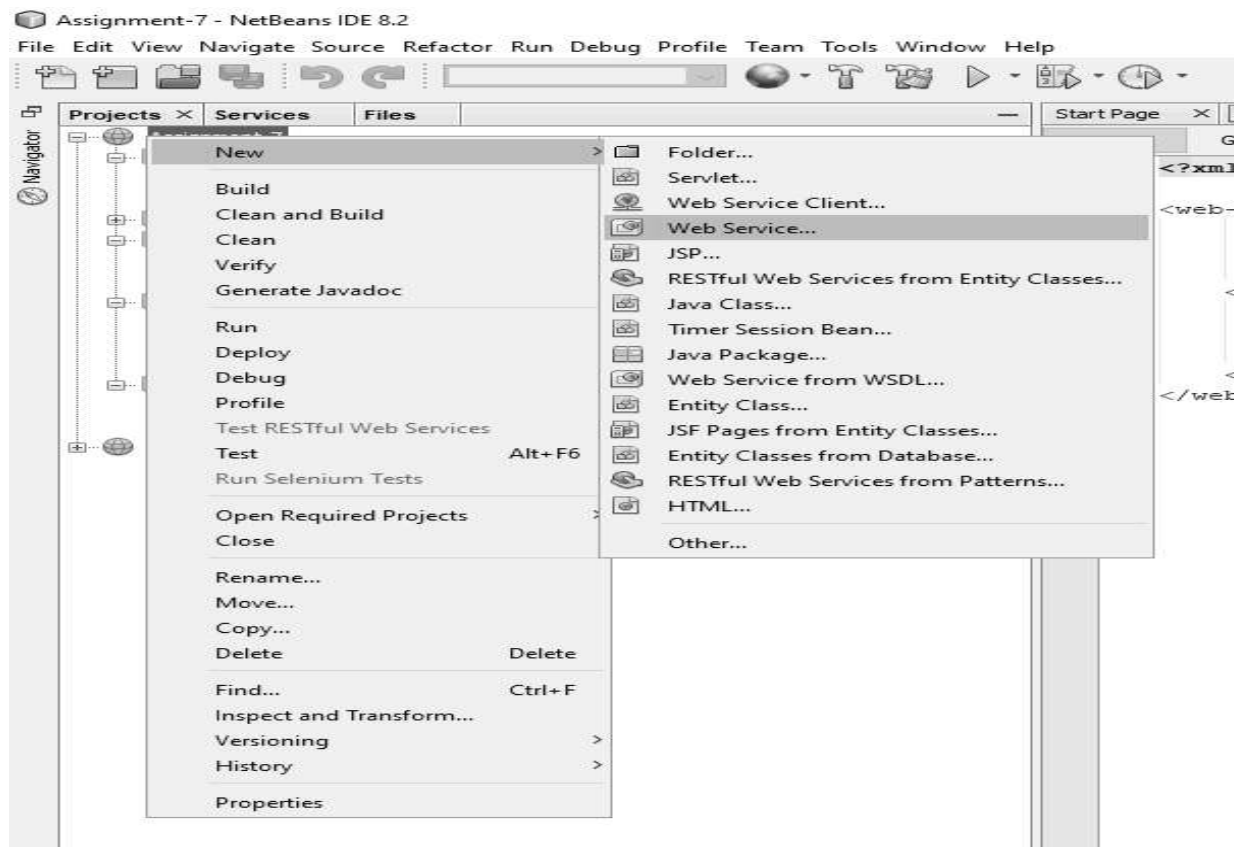
Step 1 : Create a new project

File -> New Project -> Java Web -> Web Application -> Next -> Enter Project Name (Assignment 7) -> Next -> Finish



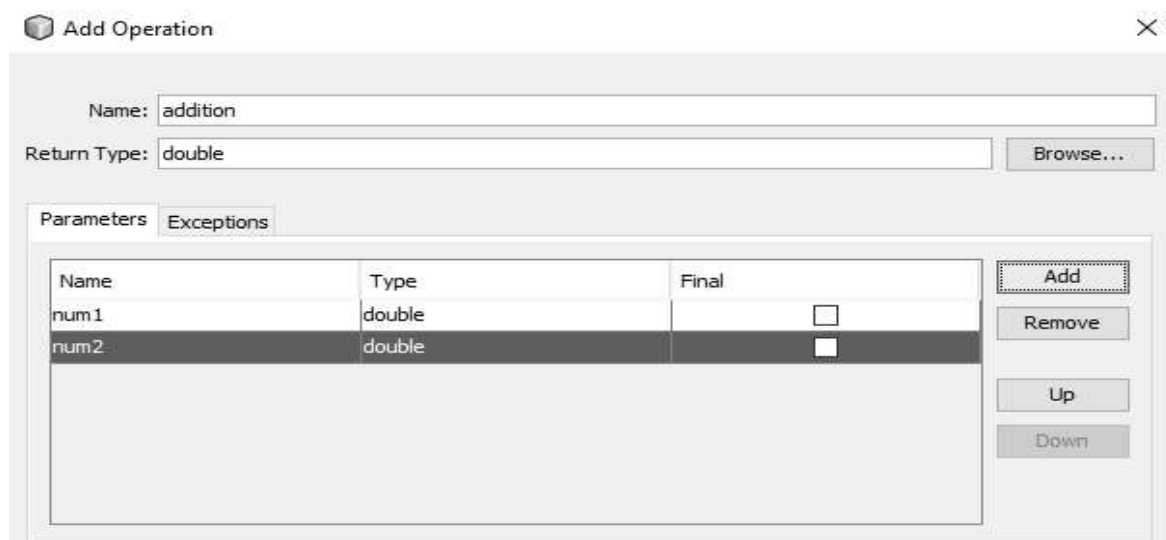
Step 2 : Create a new Web Services

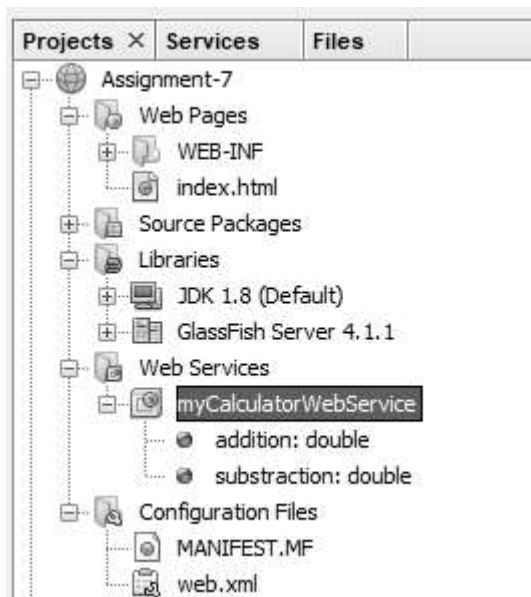
Right click on project name -> New -> Web service -> Enter name (MyCalculatorWebService) -> in package field enter (com.myservice) -> Finish



Step 3 : Add Operations

Right click on MyCalculatorWebService -> Add Operation -> Enter name (addition, subtraction, etc) -> Enter return type (double) -> Add parameters (num1, num2) -> OK





Change the function return 0.0 to num1 (operation) num2 for all operations.

Example: num1+num2

```

15  /*
16  @WebService(serviceName = "myCalculatorWebService")
17  public class myCalculatorWebService {
18
19      /**
20       * Web service operation
21       */
22      @WebMethod(operationName = "addition")
23      public double addition(@WebParam(name = "num1") double num1, @WebParam(name = "num2") double num2) {
24          //TODO write your implementation code here:
25          return num1 + num2;
26      }
27
28      /**
29       * Web service operation
30       */
31      @WebMethod(operationName = "subtraction")
32      public double subtraction(@WebParam(name = "num1") double num1, @WebParam(name = "num2") double num2) {
33          //TODO write your implementation code here:
34          return num1 - num2;
35      }
36

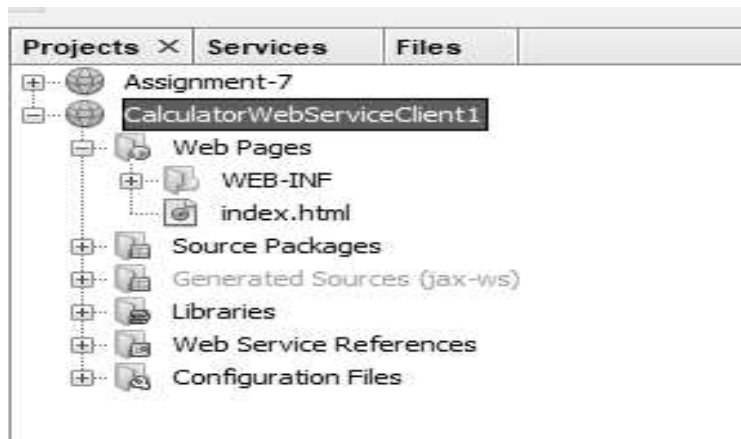
```

Step 4 : Build Project

Right click on project -> Build

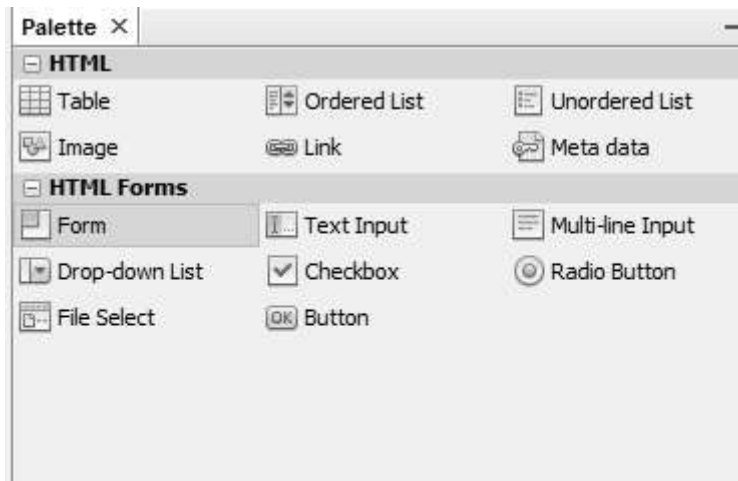
Step 5 : Create new Project

File -> New Project -> Java Web -> Web Application ->Next ->Enter Project Name (CalculatorWebServiceClient1)-> Next -> Finish

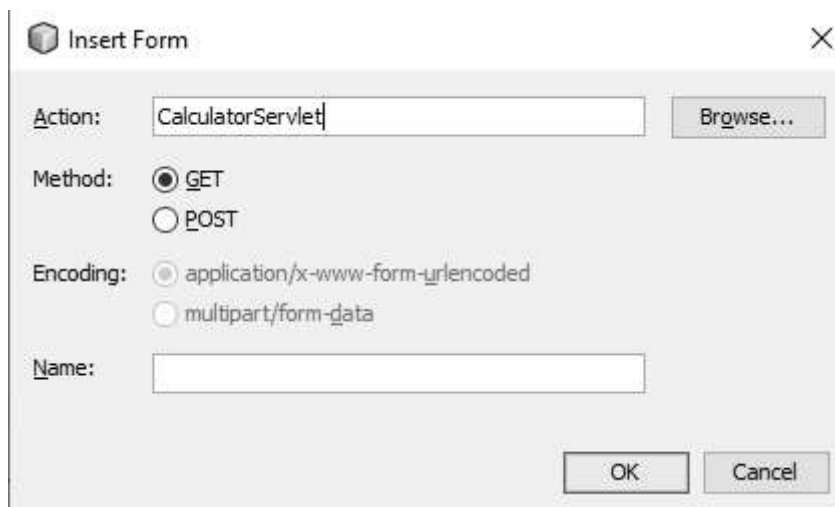


Step 6 : Create index page

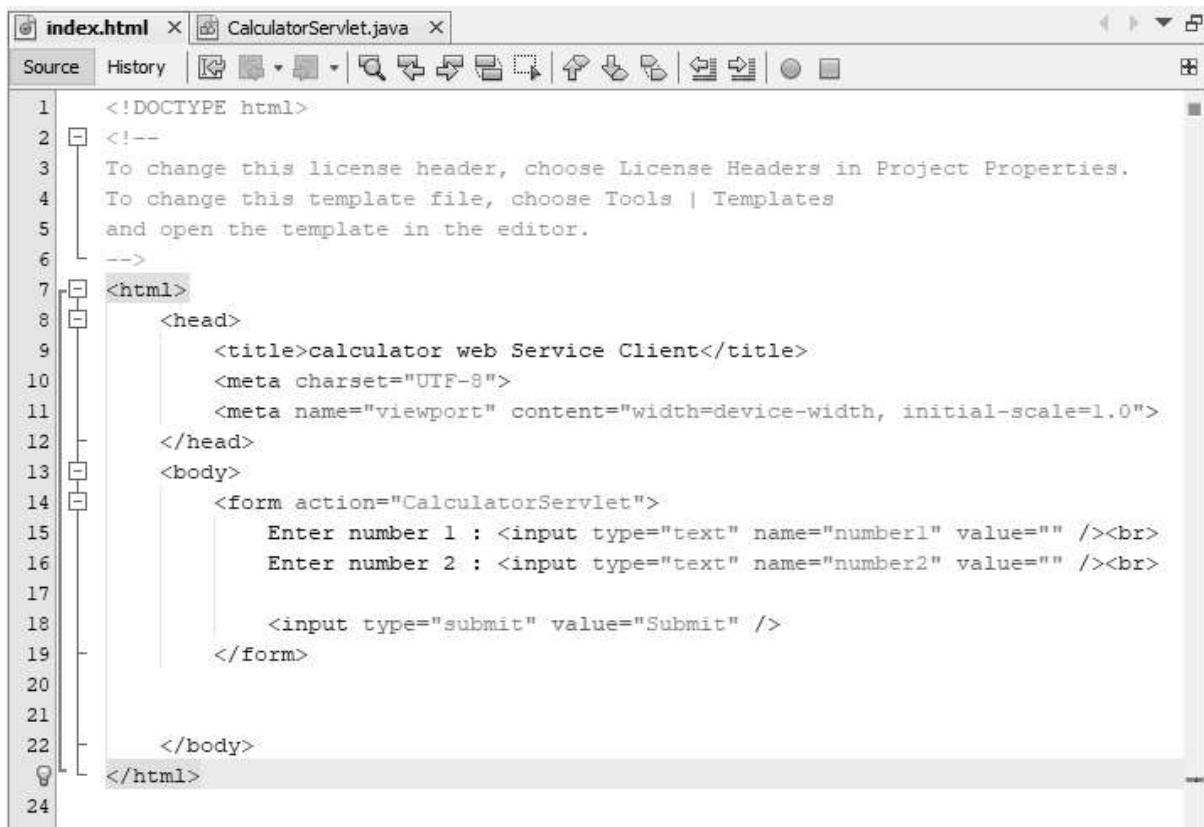
Windows -> IDE Tools -> Palette -> Add required form, text fields and buttons.



- 1) Drag and drop Form in the body section -> Enter Action (Calculatorservlet) -> Method: Get -> OK



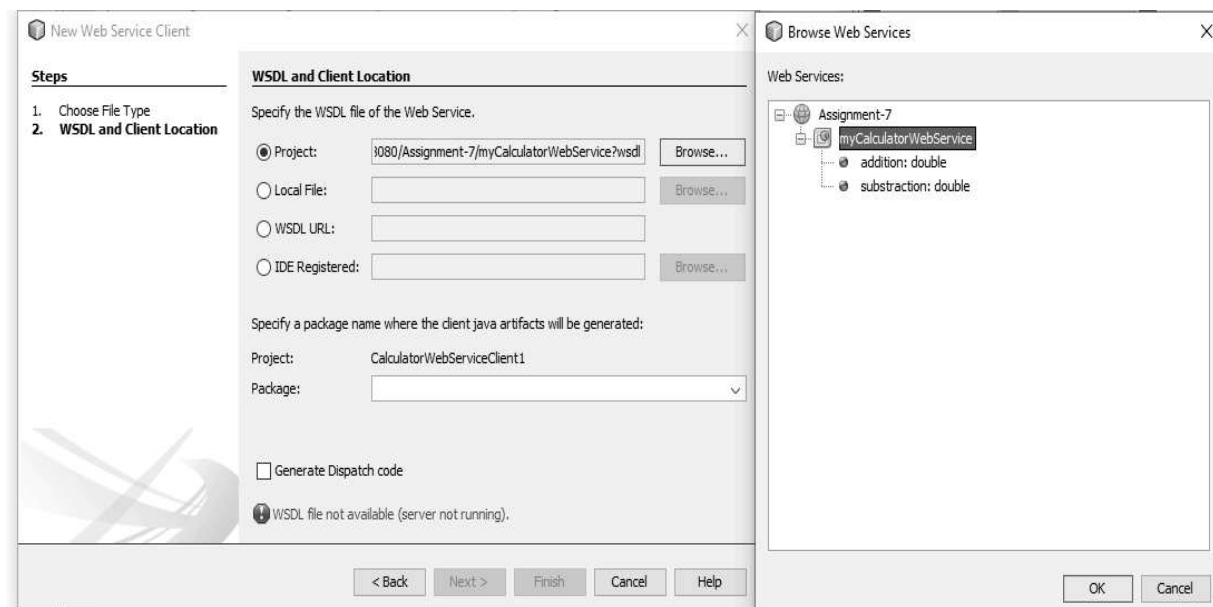
2) Add the text field for numbers input from the palette.



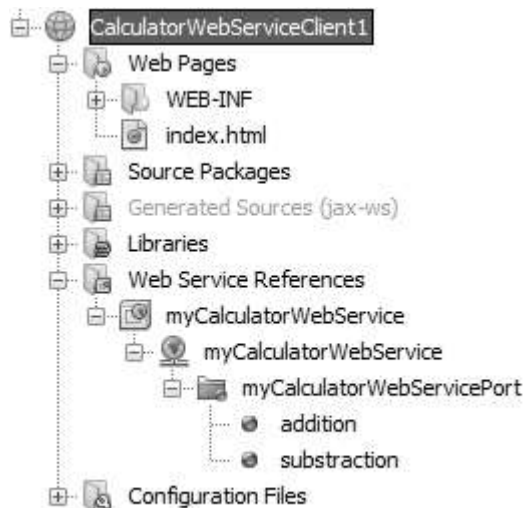
```
1 <!DOCTYPE html>
2 <!--
3 To change this license header, choose License Headers in Project Properties.
4 To change this template file, choose Tools | Templates
5 and open the template in the editor.
6 -->
7 <html>
8 <head>
9 <title>calculator web Service Client</title>
10 <meta charset="UTF-8">
11 <meta name="viewport" content="width=device-width, initial-scale=1.0">
12 </head>
13 <body>
14 <form action="CalculatorServlet">
15 Enter number 1 : <input type="text" name="number1" value="" /><br>
16 Enter number 2 : <input type="text" name="number2" value="" /><br>
17
18 <input type="submit" value="Submit" />
19 </form>
20
21
22 </body>
23 </html>
24
```

Step 7 : Create new Web Servlet Client

Right click on CalculatorWebServiceClient1-> New -> Web Service Client -> Browse the Project Assignment 7 -> OK -> Finish.

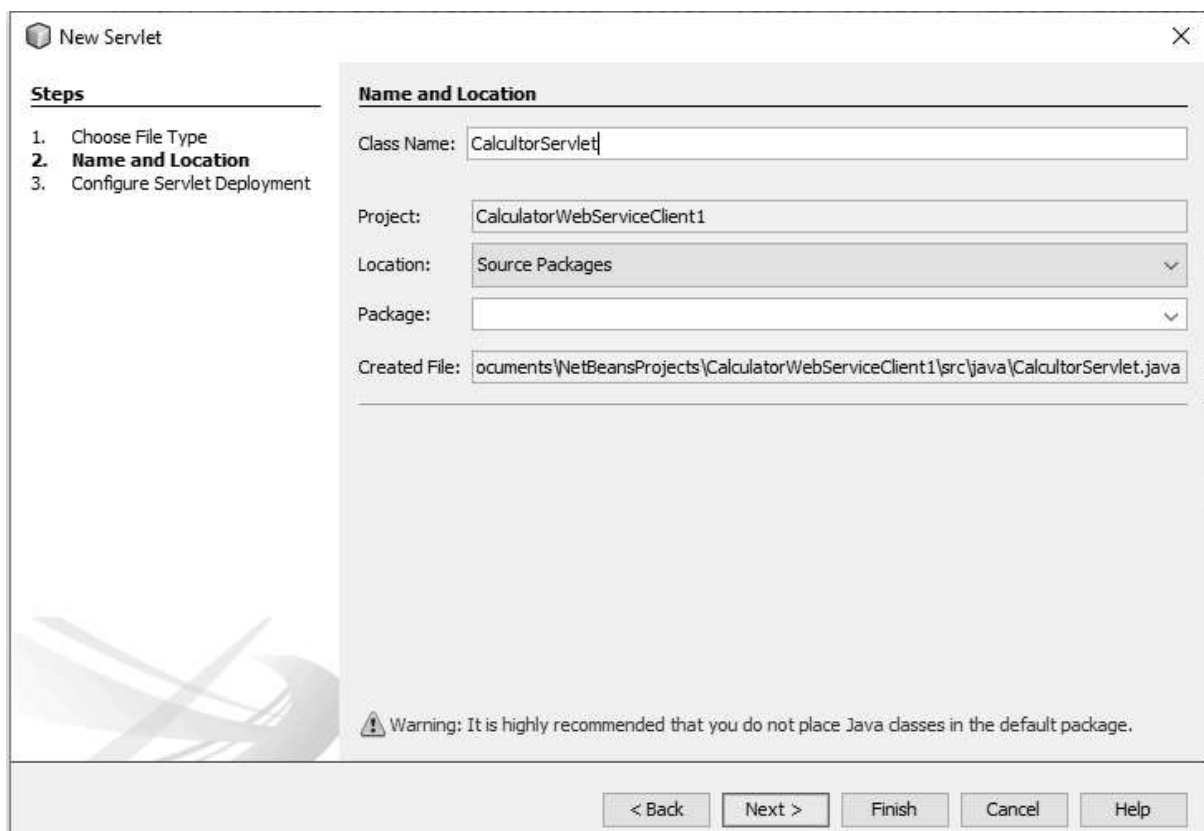


Web Service References are created successfully.



Step 8 : Create new Servlet

Right click on CalculatorWebServiceClient1-> New -> Service...-> Enter class name (CalculatorServlet) -> Next -> Check the box for adding information to web.xml -> Finish.



New Servlet

Steps

1. Choose File Type
2. Name and Location
3. **Configure Servlet Deployment**

Configure Servlet Deployment

Register the Servlet with the application by giving the Servlet an internal name (Servlet Name). Then specify patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.

☒ Add information to deployment descriptor (web.xml)

Class Name:

Servlet Name:

URL Pattern(s):

Initialization Parameters:

Name	Value
------	-------

New Edit... Delete

< Back Next > Finish Cancel Help

Step 9 : Add Operations to the CalculatorServlet

Drag and drop each operations to the CalculatorServlet.java file

HttpServlet methods. Click on the + sign on the left to edit the code.

```

private double addition(double num1, double num2) {
    // Note that the injected javax.xml.ws.Service reference as well as port objects are not thread safe.
    // If the calling of port operations may lead to race condition some synchronization is required.
    com.myservice.MyCalculatorWebService port = service.getMyCalculatorWebServicePort();
    return port.addition(num1, num2);
}

private double subtraction(double num1, double num2) {
    // Note that the injected javax.xml.ws.Service reference as well as port objects are not thread safe.
    // If the calling of port operations may lead to race condition some synchronization is required.
    com.myservice.MyCalculatorWebService port = service.getMyCalculatorWebServicePort();
    return port.subtraction(num1, num2);
}

```

Step 10 : Store the numbers in CalculatorServlet

Numbers defined in index.html file need to be stored in the Servlet to access.

```

/*
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {

        double num1, num2;

        num1 = Double.parseDouble(request.getParameter("number1"));
        num2 = Double.parseDouble(request.getParameter("number2"));

```

Step 11 : Change the output to be displayed

Results in the format for operations like addition, subtraction, etc

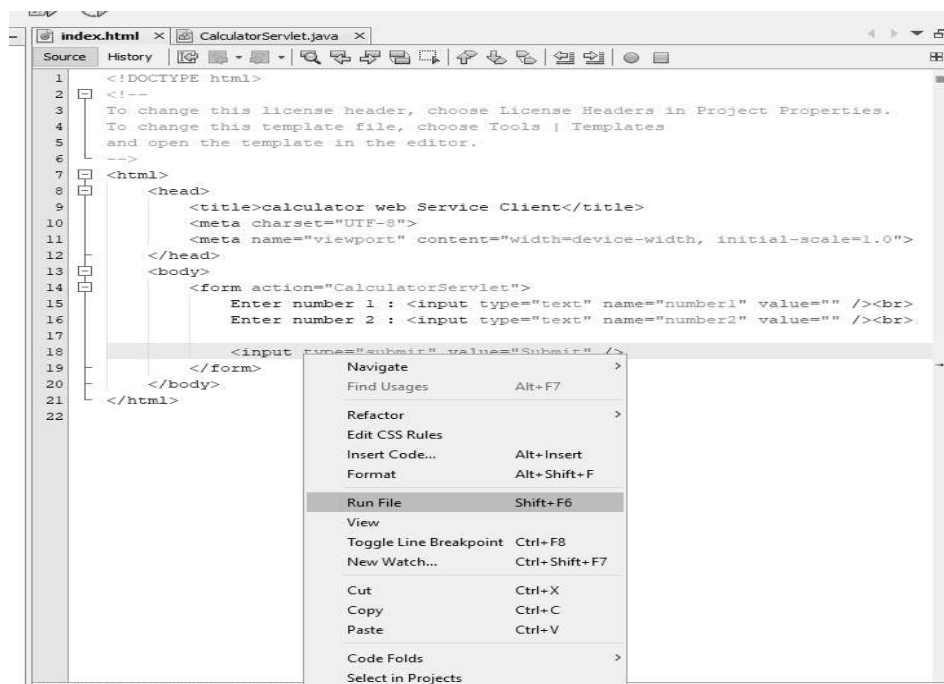
```
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet CalculatorServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Addition is " + addition(num1, num2) + "</h1>");
        out.println("<h1>Substraction is " + subtraction(num1, num2) + "</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Step 12 : Clean and Build

Right click on CalculatorWebServiceClient1-> Clean and Build

Step 13 : Run File

CalculatorWebServiceClient1 ->Web Pages -> index.html -> Right click somewhere on the file -> Run File



calculator web Service Client

localhost:8080/Calculator...

Enter number 1 :

Enter number 2 :

Servlet CalculatorServlet

localhost:8080/CalculatorWebServ...

Addition is 15.0

Substraction is 5.0