

1 JSP

In this chapter we introduce the student to the concept of JSP, a web component that is used for presentation purposes.

1.1 What is JSP?

JSP is an acronym that stands for **Java Server Page**. JSP is a server-side component, meaning that it resides in the container. It allows Java code to be included in HTML. This makes HTML dynamic.

For example, with normal HTML, we can't read the current time from the computer. But with the introduction of Java into HTML (JSP), we are able to work with dynamically generated data, not just static data.

JSP is JEE standard/specification/API. The JSE specification is known as JSR 245 (Java Specification Request 245). It can be located and downloaded from the following website:

[Java EE - Technologies \(oracle.com\)](http://java.ee-technologies.oracle.com)

1.2 Relationship between JSPs and Servlets

A JSP is actually a servlet. When a JSP is first called by the container/server, it is converted into an equivalent **Java source code**. The source code is **compiled** into **byte code** (class file). An instance of the class is **created** and **initialized**, and a **servlet is created**.

1.3 Implicit objects of JSP

JSP has objects that are implicitly (indirectly) imported to it. The objects are readily available, by default to a JSP. Below are the objects:

- request
- response
- session
- exception

- application
- config
- page
- pageContext
- out

1.4 How to write a scriptlet and an expression in JSP?

A scriptlet is a piece of Java code written inside a JSP. We use the special braces, to denote Java code inside JSP. The open brace percentage sign, `<%`, shows the start of the scriptlet, and the percentage sign close brace, `%>`, signals the end of the scriptlet.

For example, if we want to retrieve some data from a session object, we can accomplish the task as follows:

```
<%  
    String name = (String)session.getAttribute("name");  
    Integer age = (Integer)session.getAttribute("age");  
%>
```

The same with an expression, if we want to write a Java expression in JSP, we use the following special braces with an equal sign inside:

```
<%= %>
```

Say we want to determine and display the sum of two numbers, **num1** and **num2**. We can accomplish the task by doing the following:

```
<%= (num1 + num2)%>
```

1.5 How to declare variables, write comments and import packages in JSP?

To declare a variable in JSP, we use the following generic syntax:

```
<%! variableName=value; %>
```

Say we want to declare the a string variable to hold the **name** of a person as **Thato**, and an **age** variable to hold the value **18**. The declaration of the two variables will be done as follows:

```
<%!  
  
    String name = "Thato";  
  
    int age=18;  
  
%>
```

To write comments in JSP we use the following generic syntax:

```
<%-- my comment --%>
```

For example we can comment the declaration of the person variables as follows:

```
<%!  
  
    <%-- name of the person --%>  
  
    String name = "Thato";  
  
    <%-- age of the person --%>  
  
    int age=18;  
  
%>
```

The generic syntax for importing packages into a JSP file is as follows:

```
<% @page importKeyword="fully-qualified-class-name"%>
```

If you are importing more than one class, you will have the following:

```
<% @page importKeyword="fully-qualified-class-name1, fully-qualified-class-name2,.."%>
```

For example, say we want to import both the **DecimalFormat** found in the **java.text** package and the **Random** class found in the **java.util** package. Then we will have the following directive:

```
<% @page import = "java.text.DecimalFormat, java.util.Random"%>
```

1.6 Example

Activity

Create a web application that will determine and display the **sum** of two numbers, **num1** and **num2**. The application must allow the user to enter the two numbers. Upon receiving the two numbers, the application must determine their sum and display it.

To do

Design a solution to the problem. The solution must entail a discussion of the program flow, identification of MVC components which emanate from the discussion, and an implementation of the design in software.

Program flow

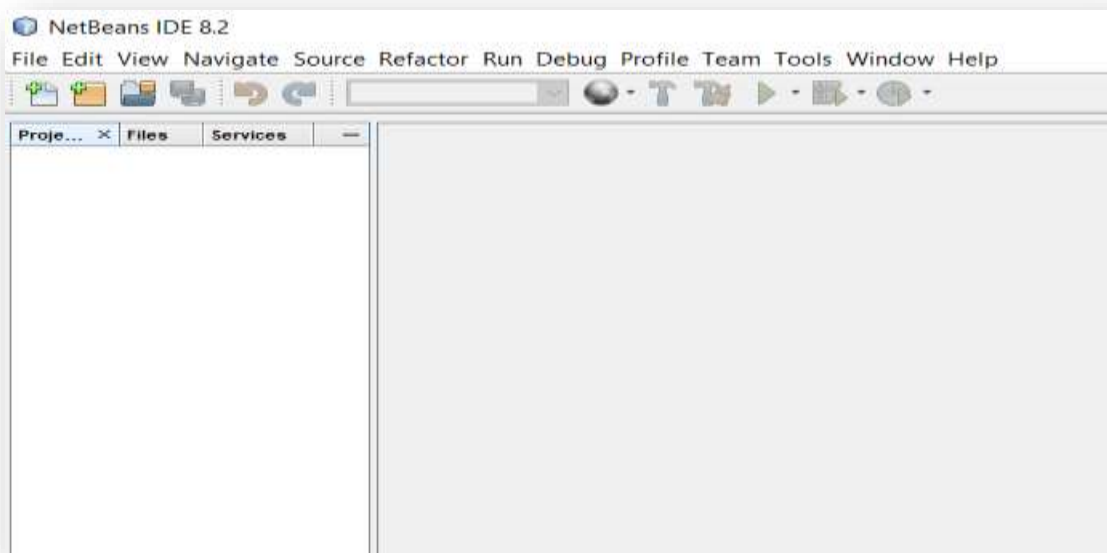
Exercise for you.

MVC components

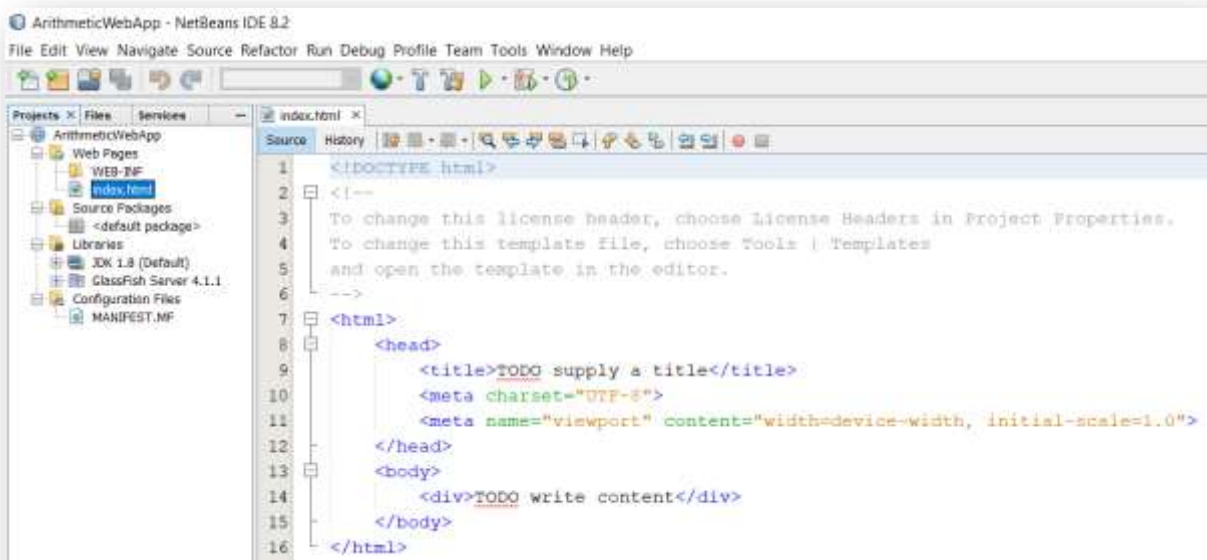
Exercise for you.

Implementation

Launch NetBeans



Create a web application project called **ArithmeticWebApp**

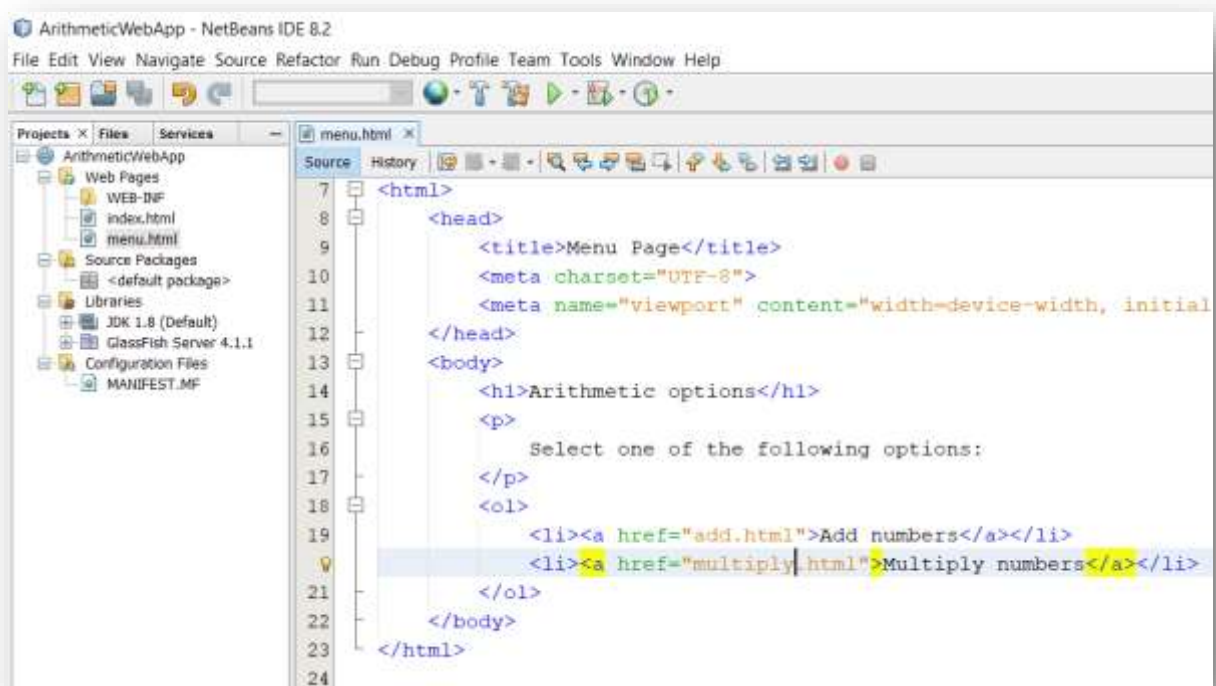


Modify the index.html file



```
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>Welcome Page</title>
10    <meta charset="UTF-8">
11    <meta name="viewport" content="width=device-width, initial-scale=1.0">
12  </head>
13  <body>
14    <h1>Welcome</h1>
15    <p>
16      Welcome to our arithmetic web app. Click <a href="menu.html">here</a> to start.
17    </p>
18  </body>
19 </html>
```

Create the menu.html page

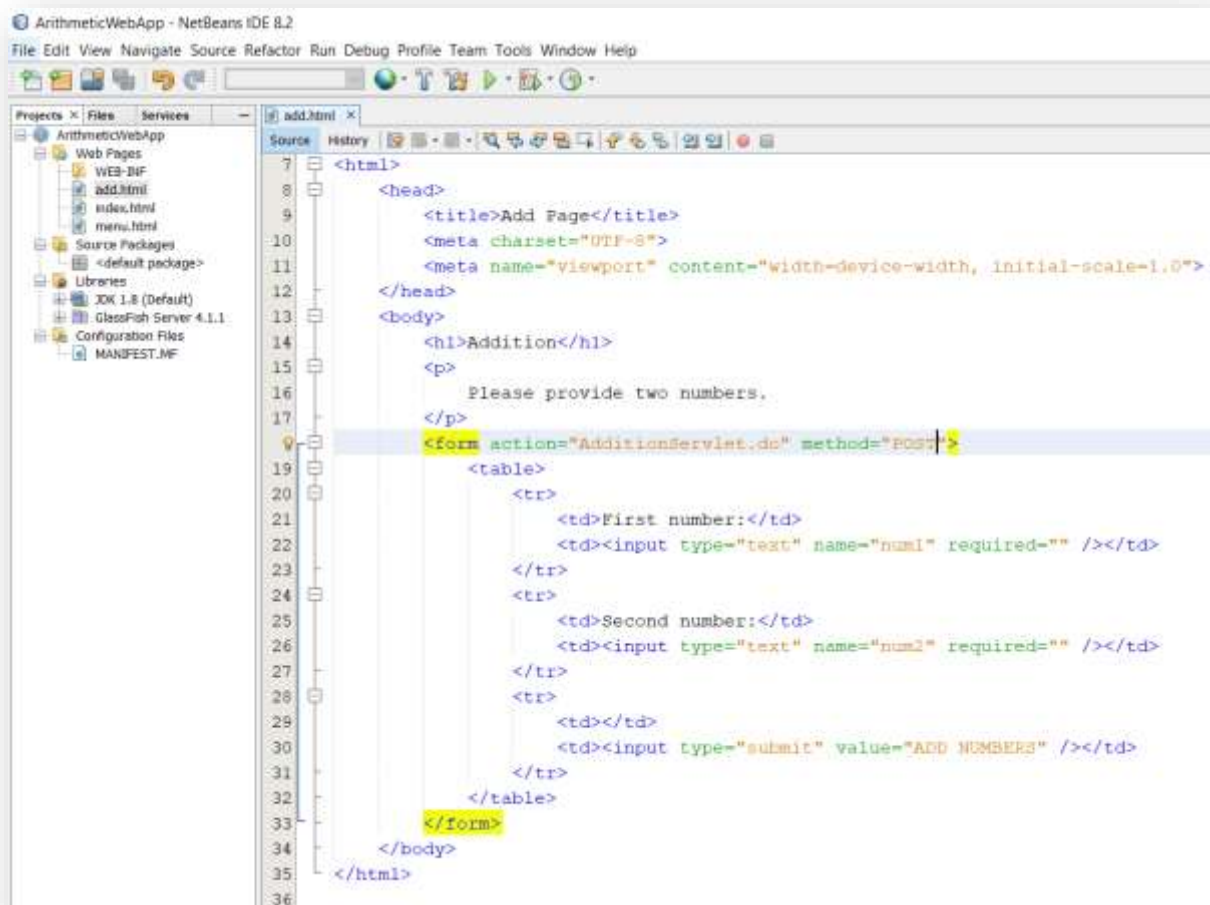


```
ArithmeticWebApp - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects X Files Services
ArithmeticWebApp
  Web Pages
    index.html
    menu.html
  Source Packages
    <default package>
  Libraries
    JDK 1.8 (Default)
    GlassFish Server 4.1.1
  Configuration Files
    MANIFEST.MF

7 <html>
8   <head>
9     <title>Menu Page</title>
10    <meta charset="UTF-8">
11    <meta name="viewport" content="width=device-width, initial
12  </head>
13  <body>
14    <h1>Arithmetic options</h1>
15    <p>
16      Select one of the following options:
17    </p>
18    <ol>
19      <li><a href="add.html">Add numbers</a></li>
20      <li><a href="multiply.html">Multiply numbers</a></li>
21    </ol>
22  </body>
23 </html>
24
```

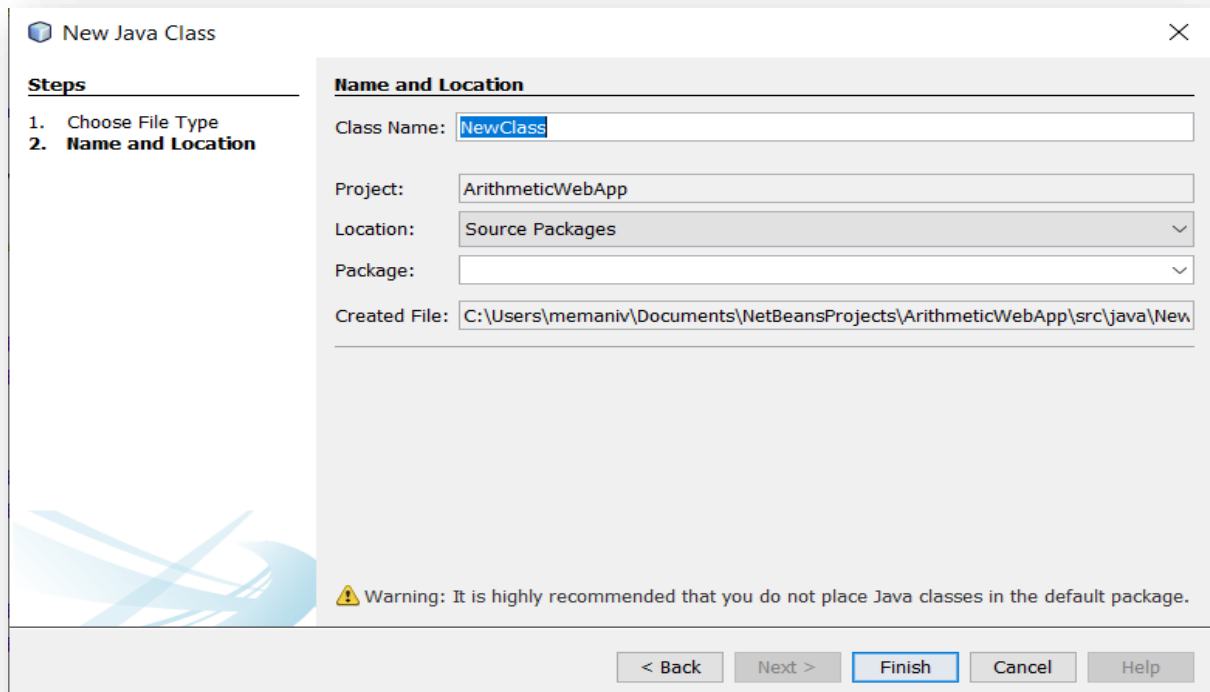
Create the add.html file



The screenshot shows the NetBeans IDE interface for a project named 'ArithmeticWebApp'. The 'Projects' pane on the left shows the project structure, including 'Web Pages' and 'add.html'. The 'Source' editor on the right displays the HTML code for 'add.html'. The code includes a title 'Add Page', a meta charset of 'UTF-8', and a viewport meta tag. The body contains a heading 'Addition', a paragraph 'Please provide two numbers.', and a form with two text input fields for 'num1' and 'num2', and a submit button labeled 'ADD NUMBERS'. The form action is 'AdditionServlet.do' and the method is 'POST'.

```
7 <html>
8 <head>
9   <title>Add Page</title>
10  <meta charset="UTF-8">
11  <meta name="viewport" content="width=device-width, initial-scale=1.0">
12 </head>
13 <body>
14   <h1>Addition</h1>
15   <p>
16     Please provide two numbers.
17   </p>
18   <form action="AdditionServlet.do" method="POST">
19     <table>
20       <tr>
21         <td>First number:</td>
22         <td><input type="text" name="num1" required="" /></td>
23       </tr>
24       <tr>
25         <td>Second number:</td>
26         <td><input type="text" name="num2" required="" /></td>
27       </tr>
28       <tr>
29         <td></td>
30         <td><input type="submit" value="ADD NUMBERS" /></td>
31       </tr>
32     </table>
33   </form>
34 </body>
35 </html>
36
```

Right click on the project and select New | Java Class

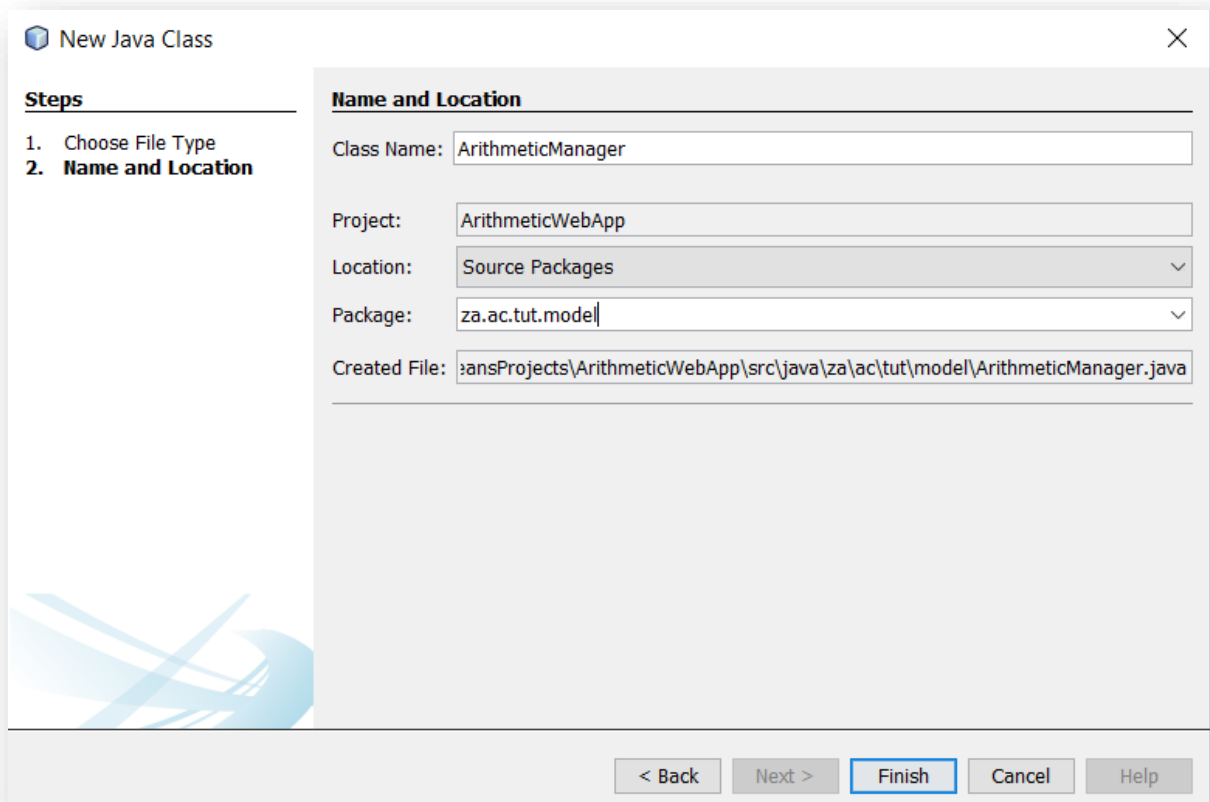


The 'New Java Class' dialog box is shown. It has a title bar with a close button. On the left, under 'Steps', step 2 'Name and Location' is selected. The main area is titled 'Name and Location'. It contains the following fields:

- Class Name:
- Project:
- Location:
- Package:
- Created File:

At the bottom, there is a warning icon and text: "Warning: It is highly recommended that you do not place Java classes in the default package." Below the warning are five buttons: "< Back", "Next >", "Finish" (highlighted), "Cancel", and "Help".

Name the class ArithmeticManager and package in za.ac.tut.model

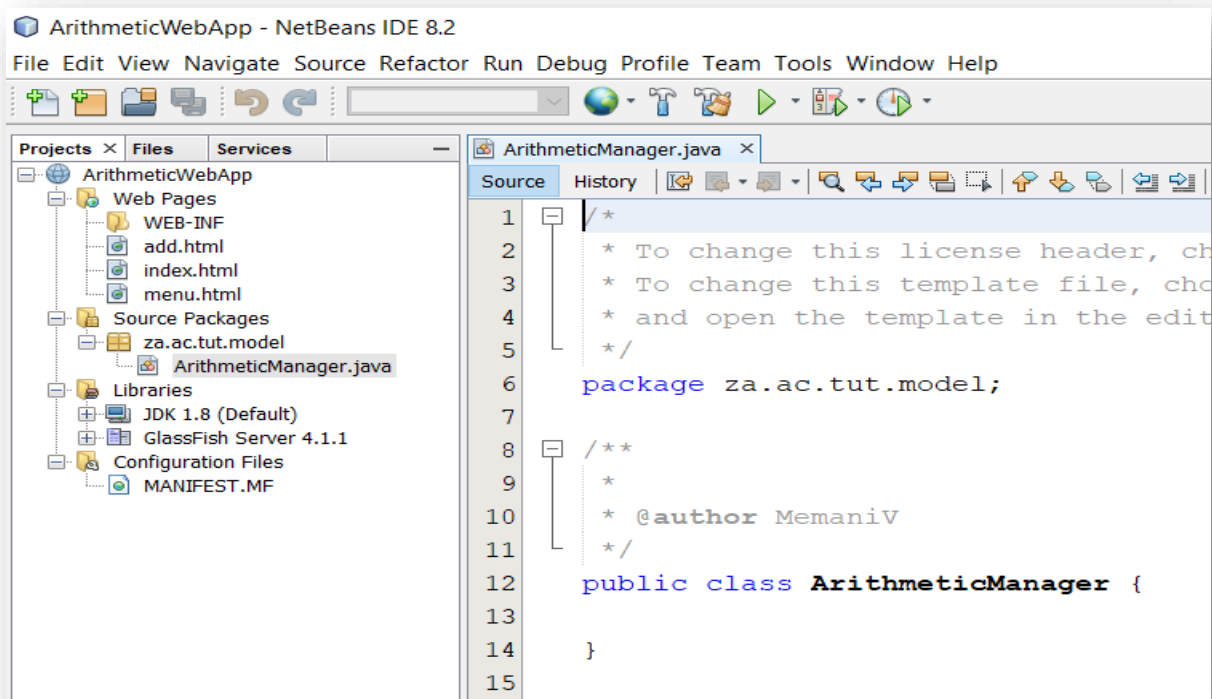


The 'New Java Class' dialog box is shown again, but with different values. The 'Steps' list on the left remains the same. The 'Name and Location' section now contains:

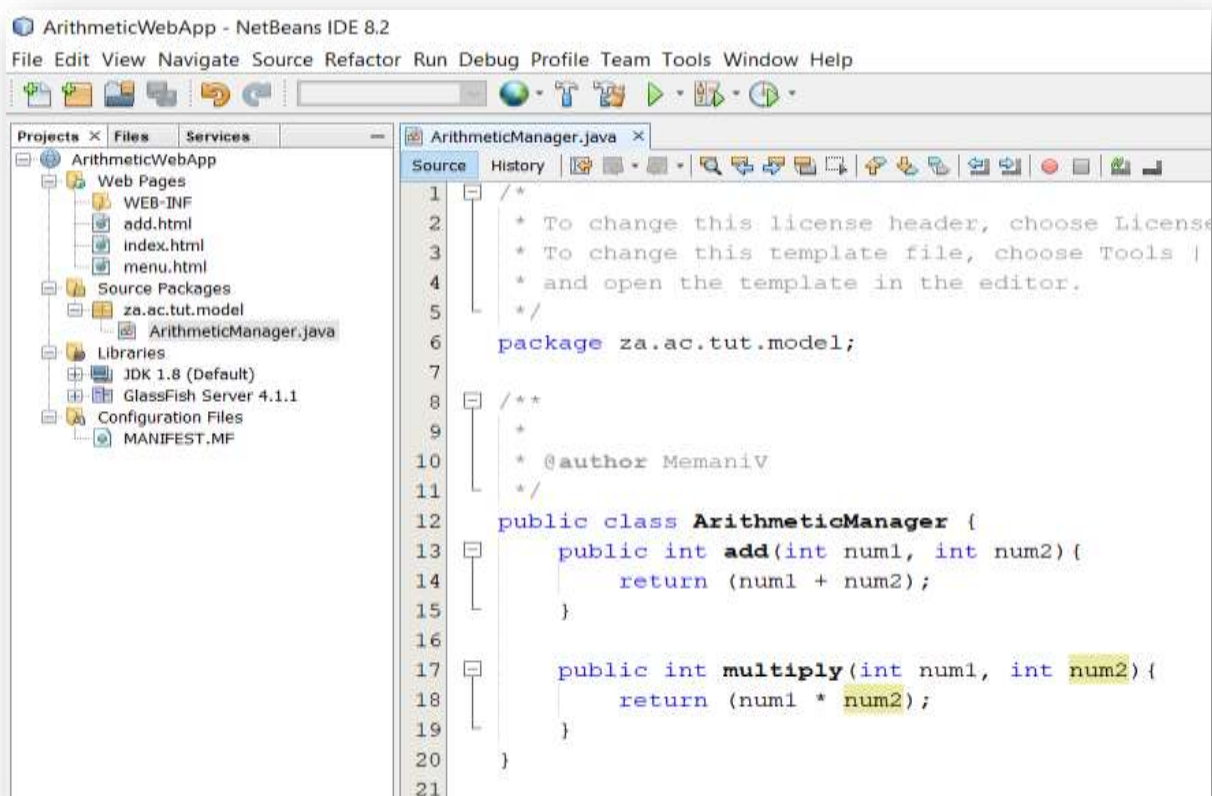
- Class Name:
- Project:
- Location:
- Package:
- Created File:

The 'Finish' button is still highlighted. The warning message is no longer visible.

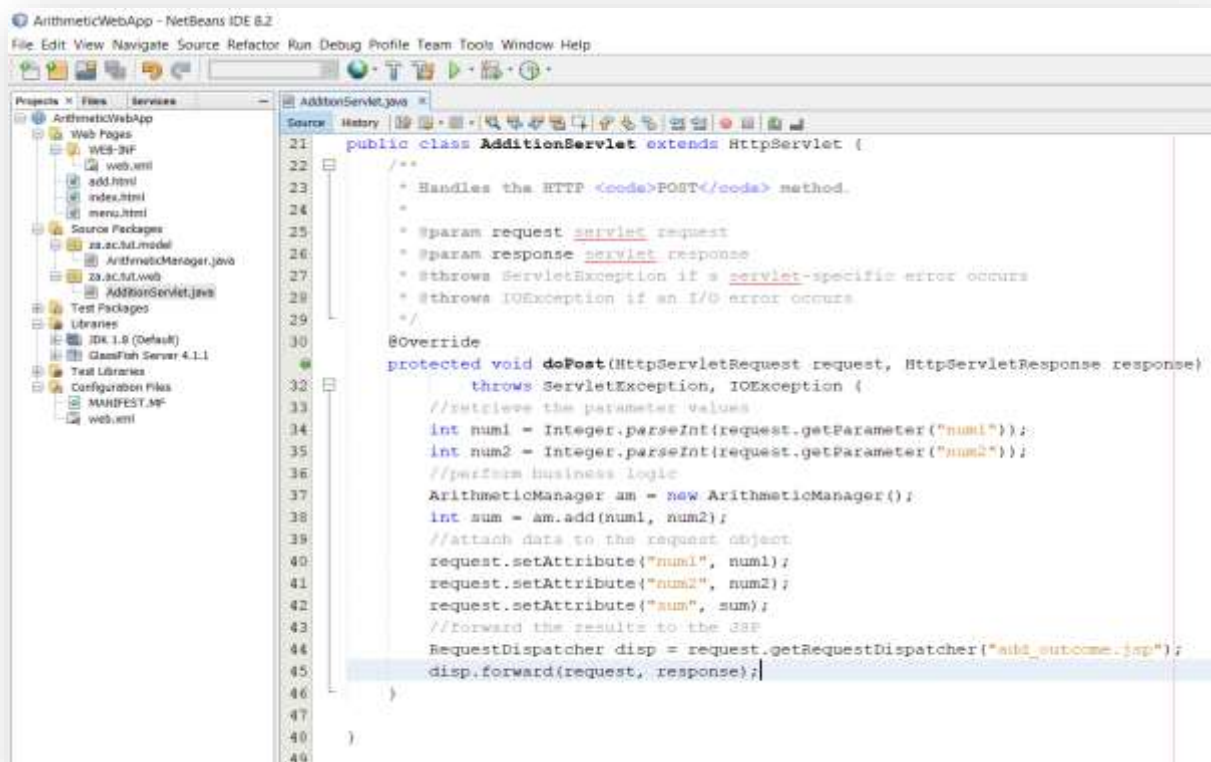
Click Finish



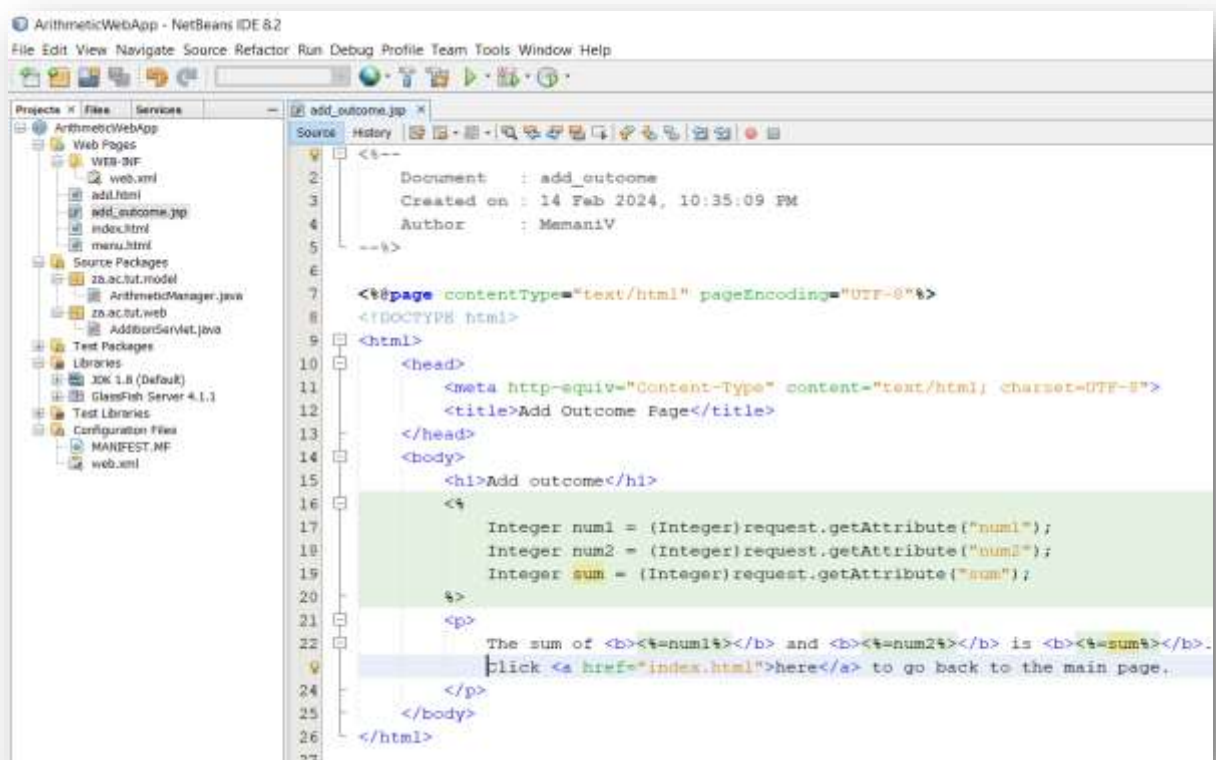
Modify the class



Create a servlet called AdditionServlet



Create the add_outcome.jsp file

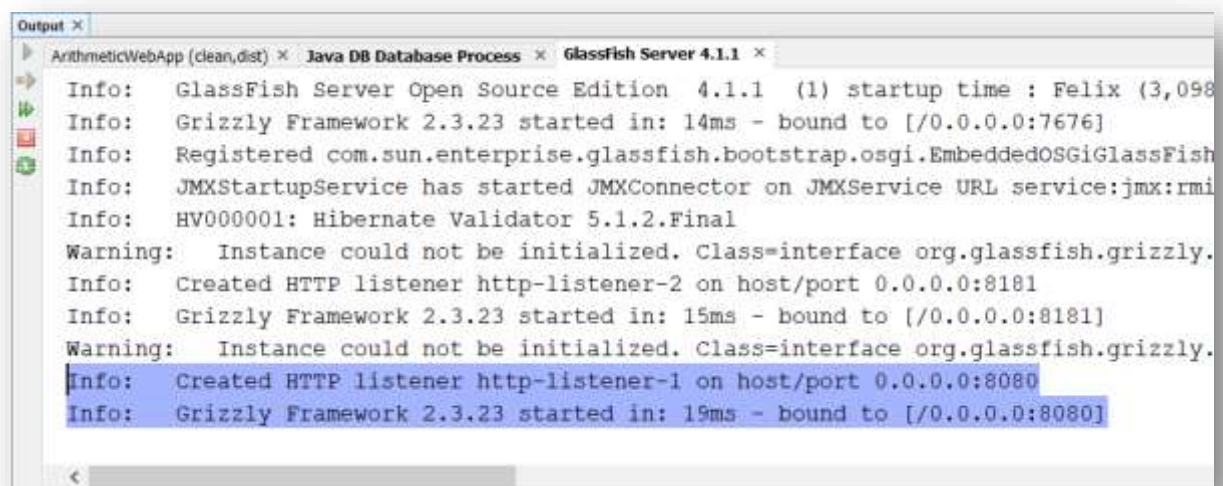


Clean and build the project



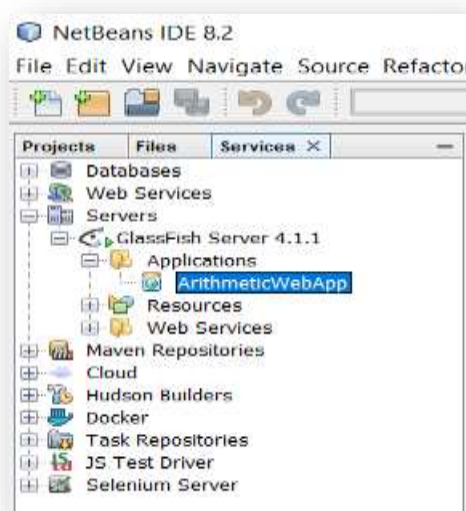
```
Output: ArithmeticWebApp [clean, dist] x
library-inclusion-in-archive:
library-inclusion-in-manifest:
Created dir: C:\Users\memaniv\Documents\NetBeansProjects\ArithmeticWebApp\build\empty
Created dir: C:\Users\memaniv\Documents\NetBeansProjects\ArithmeticWebApp\build\generated-sources\ap-source-output
Compiling 2 source files to C:\Users\memaniv\Documents\NetBeansProjects\ArithmeticWebApp\build\web\WEB-INF\classes
compile:
compile-jsp:
Created dir: C:\Users\memaniv\Documents\NetBeansProjects\ArithmeticWebApp\dist
Building jar: C:\Users\memaniv\Documents\NetBeansProjects\ArithmeticWebApp\dist\ArithmeticWebApp.war
do-dist:
dist:
BUILD SUCCESSFUL (total time: 1 second)
```

Start the server

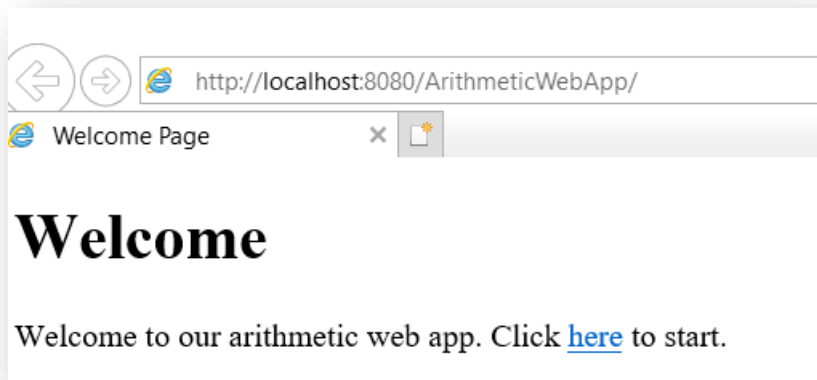


```
Output x
ArithmeticWebApp (clean, dist) x Java DB Database Process x GlassFish Server 4.1.1 x
Info: GlassFish Server Open Source Edition 4.1.1 (1) startup time : Felix (3,098
Info: Grizzly Framework 2.3.23 started in: 14ms - bound to [/0.0.0.0:7676]
Info: Registered com.sun.enterprise.glassfish.bootstrap.osgi.EmbeddedOSGiGlassFish
Info: JMXStartupService has started JMXConnector on JMXService URL service:jmx:rmi
Info: HV000001: Hibernate Validator 5.1.2.Final
Warning: Instance could not be initialized. Class=interface org.glassfish.grizzly.
Info: Created HTTP listener http-listener-2 on host/port 0.0.0.0:8181
Info: Grizzly Framework 2.3.23 started in: 15ms - bound to [/0.0.0.0:8181]
Warning: Instance could not be initialized. Class=interface org.glassfish.grizzly.
Info: Created HTTP listener http-listener-1 on host/port 0.0.0.0:8080
Info: Grizzly Framework 2.3.23 started in: 19ms - bound to [/0.0.0.0:8080]
```

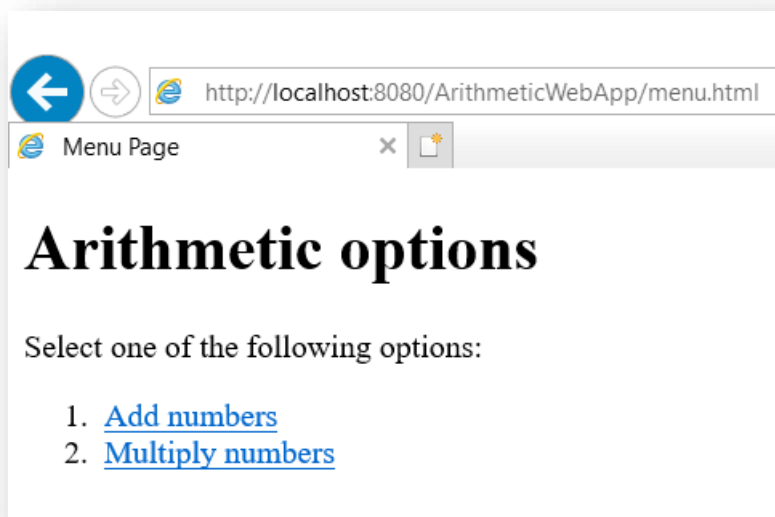
Deploy web application



Run web application



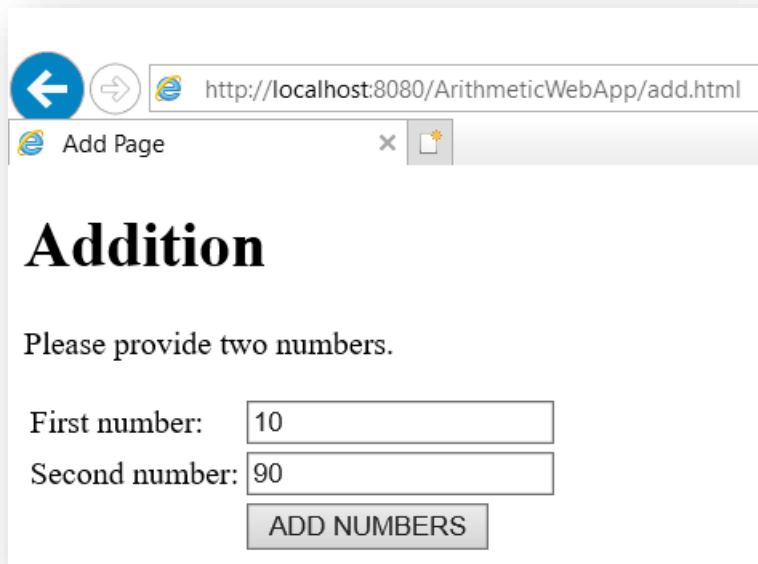
Click on the link






Click on the first link



Enter the numbers



⬅️ ➡️  http://localhost:8080/ArithmeticWebApp/add.html

 Add Page × 

Addition


Please provide two numbers.



First number:

Second number:

Click on the button

here to go back to the main page.' The link 'here' is blue and underlined." data-bbox="137 450 744 638"/>

⬅️ ➡️  http://localhost:8080/ArithmeticWebApp/AdditionServlet.do


 Add Outcome Page × 



Add outcome

The sum of 10 and 90 is 100. Click [here](#) to go back to the main page.

Click on the link

here to start.' The link 'here' is purple and underlined." data-bbox="137 717 647 870"/>

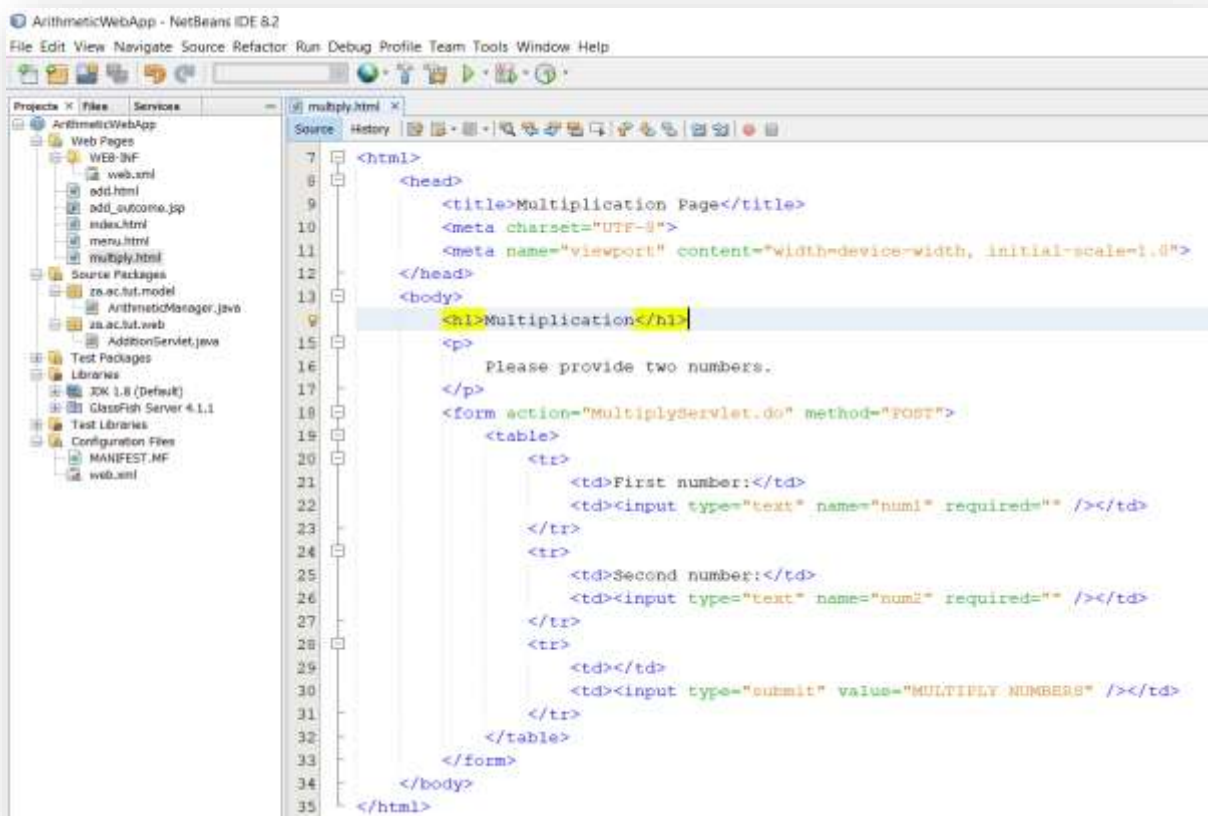
⬅️ ➡️  http://localhost:8080/ArithmeticWebApp/index.html

 Welcome Page × 

Welcome

Welcome to our arithmetic web app. Click [here](#) to start.

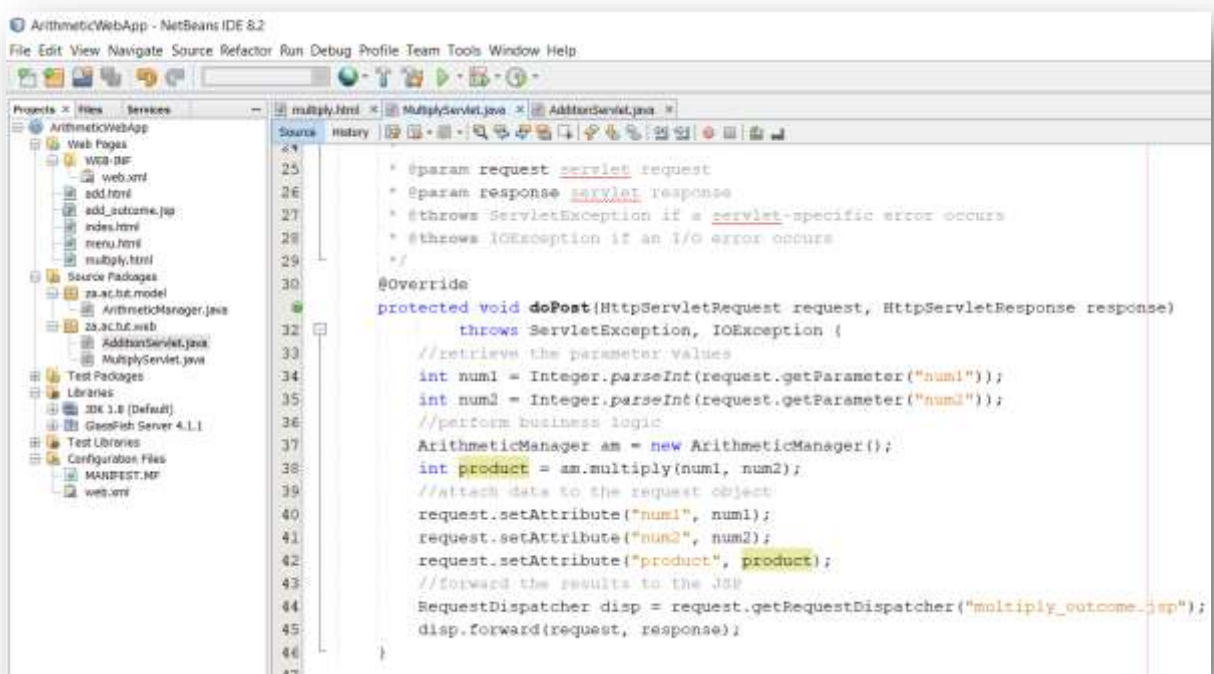
Create an html page called multiply.html



The screenshot shows the NetBeans IDE interface. On the left, the 'Projects' pane displays the project structure for 'ArithmeticWebApp'. The 'Web Pages' folder is expanded, showing files like 'web.xml', 'add.html', 'add_outcome.jsp', 'index.html', 'menu.html', and 'multiply.html'. The 'multiply.html' file is selected. The main editor window shows the HTML code for 'multiply.html'. The code includes a head section with a title 'Multiplication Page', a meta charset of 'UTF-8', and a viewport meta tag. The body section contains a heading 'Multiplication', a paragraph 'Please provide two numbers.', and a form with two text input fields for 'num1' and 'num2', and a submit button labeled 'MULTIPLY NUMBERS'. The form action is 'MultiplyServlet.do' and the method is 'POST'.

```
<html>
<head>
  <title>Multiplication Page</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
  <h1>Multiplication</h1>
  <p>
    Please provide two numbers.
  </p>
  <form action="MultiplyServlet.do" method="POST">
    <table>
      <tr>
        <td>First number:</td>
        <td><input type="text" name="num1" required="" /></td>
      </tr>
      <tr>
        <td>Second number:</td>
        <td><input type="text" name="num2" required="" /></td>
      </tr>
      <tr>
        <td></td>
        <td><input type="submit" value="MULTIPLY NUMBERS" /></td>
      </tr>
    </table>
  </form>
</body>
</html>
```

Create a servlet called MultiplyServlet



The screenshot shows the NetBeans IDE interface. On the left, the 'Projects' pane displays the project structure for 'ArithmeticWebApp'. The 'Web Pages' folder is expanded, showing files like 'web.xml', 'add.html', 'add_outcome.jsp', 'index.html', 'menu.html', and 'multiply.html'. The 'multiply.html' file is selected. The main editor window shows the Java code for 'MultiplyServlet.java'. The code includes a package declaration 'package com.example.arithmeticwebapp.servlet;', imports for 'HttpServletRequest', 'HttpServletResponse', 'ServletException', and 'IOException', and a class declaration 'public class MultiplyServlet extends HttpServlet'. The 'doPost' method is overridden, retrieving parameters 'num1' and 'num2' from the request, performing business logic using 'ArithmeticManager', and forwarding the results to the 'multiply_outcome.jsp' page.

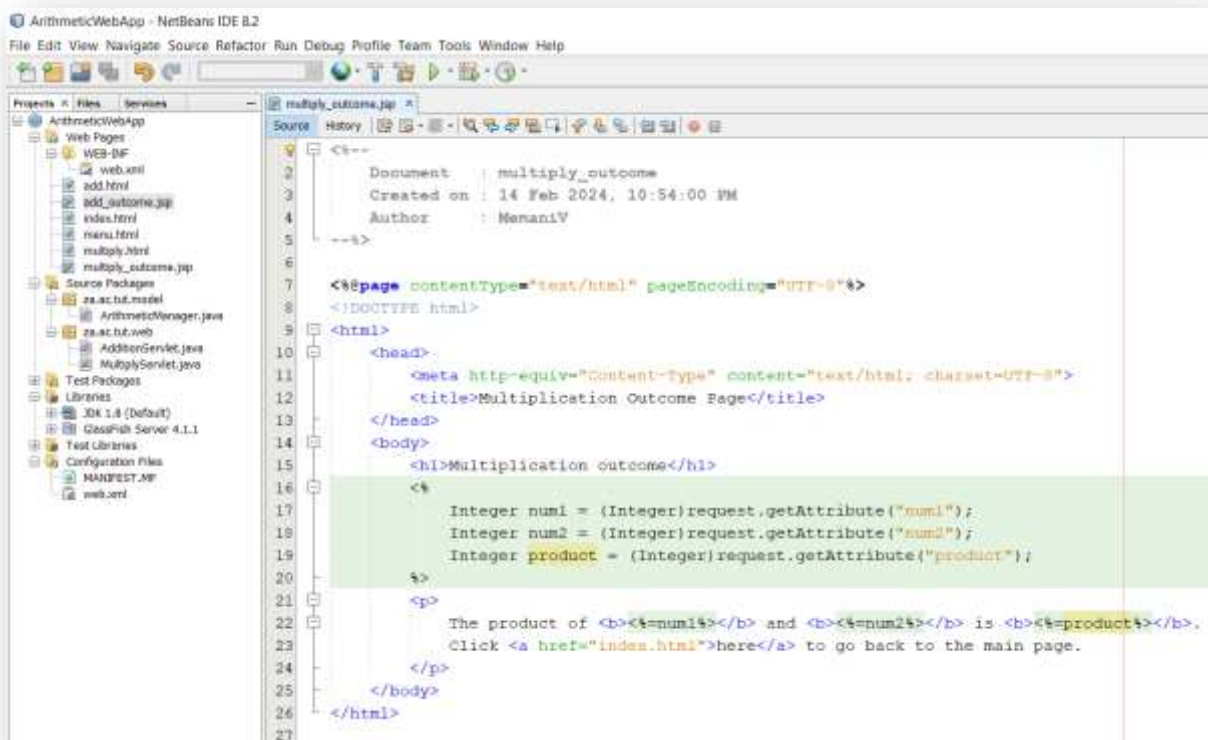
```
package com.example.arithmeticwebapp.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

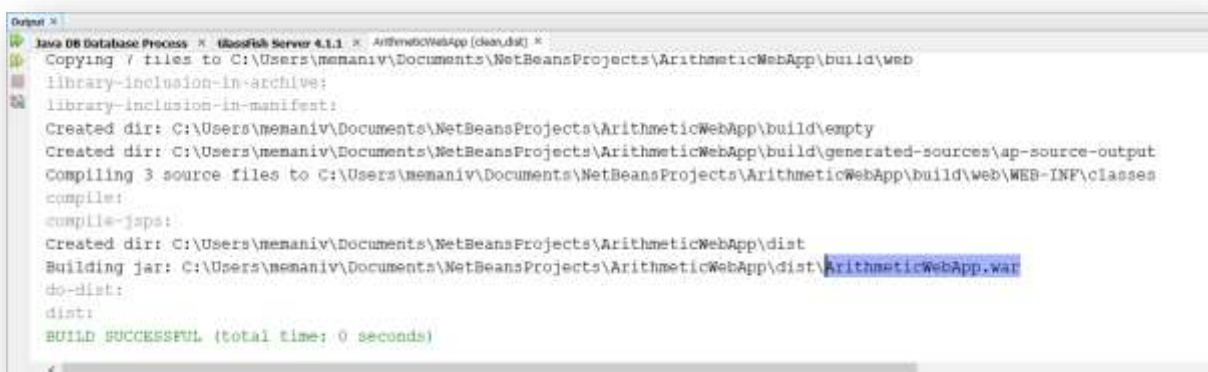
@WebServlet(urlPatterns = {"/multiply"})
public class MultiplyServlet extends HttpServlet {

    /**
     * @param request HttpServletRequest request
     * @param response HttpServletResponse response
     * @throws ServletException if a ServletException-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        //retrieve the parameter values
        int num1 = Integer.parseInt(request.getParameter("num1"));
        int num2 = Integer.parseInt(request.getParameter("num2"));
        //perform business logic
        ArithmeticManager am = new ArithmeticManager();
        int product = am.multiply(num1, num2);
        //attach data to the request object
        request.setAttribute("num1", num1);
        request.setAttribute("num2", num2);
        request.setAttribute("product", product);
        //forward the results to the JSP
        RequestDispatcher disp = request.getRequestDispatcher("multiply_outcome.jsp");
        disp.forward(request, response);
    }
}
```

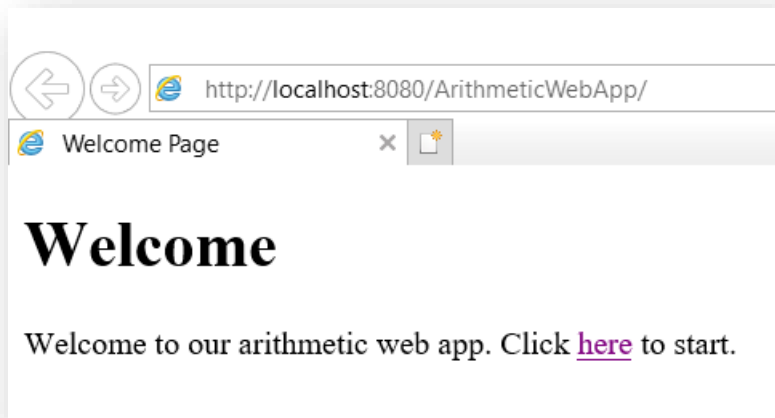
Create a JSP file called multiply_outcome



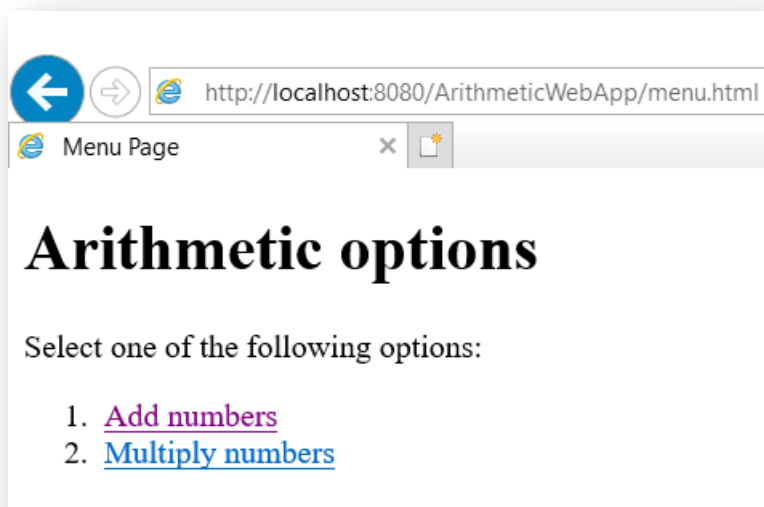
Clean and build the web application



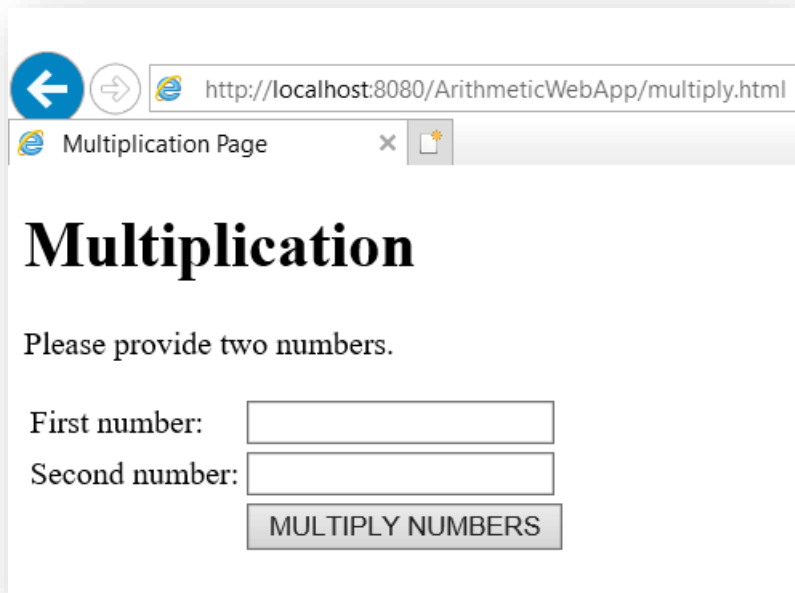
Start server, deploy and run application



Click on the link

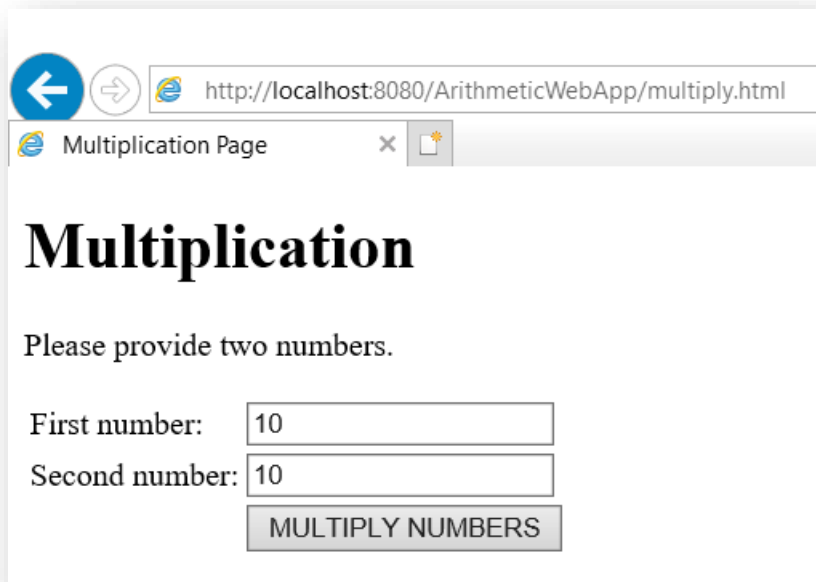


Click on the second link



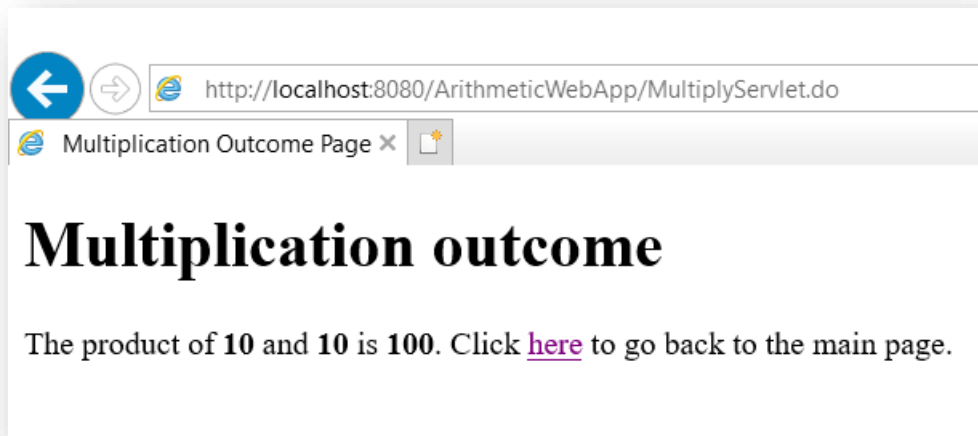
A screenshot of a web browser window. The address bar shows the URL `http://localhost:8080/ArithmeticWebApp/multiply.html`. The browser tab is titled "Multiplication Page". The page content features a large heading "Multiplication" in a bold serif font. Below the heading, the text "Please provide two numbers." is displayed. There are two input fields: the first is labeled "First number:" and the second is labeled "Second number:". Both fields are currently empty. Below the input fields is a button labeled "MULTIPLY NUMBERS".

Enter numbers

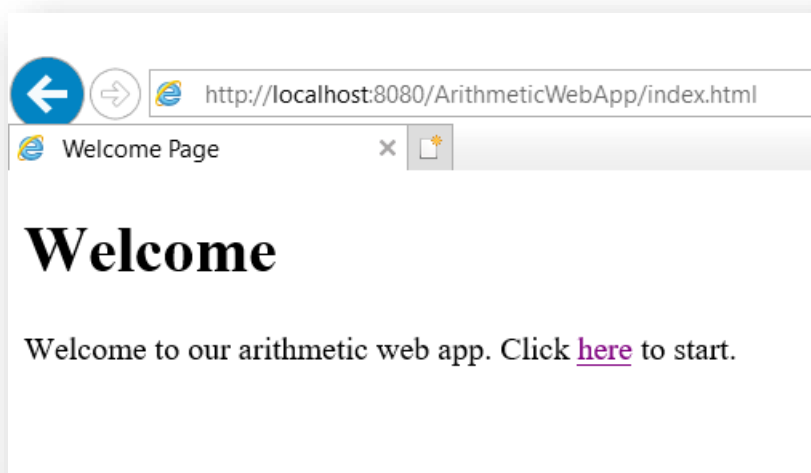


A second screenshot of the same web browser window, showing the same "Multiplication" page. In this state, the input fields have been populated with the number "10". The "First number:" field contains "10" and the "Second number:" field also contains "10". The "MULTIPLY NUMBERS" button remains visible below the input fields.

Click on the button



Click on the link.



1.7 Conclusion

In this chapter we managed to introduce the concept of JSP and demonstrated its application in a web development environment. We also referenced **Servlets**. A detailed explanation of servlets will follow in the next chapter.

Thank you for having taken time to go through the chapter. Please do the given DIYs. Enjoy the rest of the day and God bless you.

1.8 DIY (Do It Yourself)

In this DIY we want you to develop web applications for the given activities.

Activity 1

Expand the example done in this chapter by including subtraction and division.

Activity 2

Create a web application called **AgeCategoryWebApp** that will classify people according to their ages. The classification must be done according to the table below:

Age	Category
0 - 14	Child
15 - 24	Youth
25 - 64	Adult
65 and over	Senior

Activity 3

In the past four years the world was under attack from COVID-19. The pandemic drastically changed the way people led their lives. Suddenly people were expected to wear face masks in public, sanitise their hands and practice social distancing. Entry to public institutions such as schools also required people to have their temperatures checked. Temperatures above 38 degrees Celsius resulted in people being refused entry into public buildings and advised to seek medical help.

Though the pandemic has immensely subsided, and as a result regulations have been done away with, some private institutions still require people to observe the COVID-19 prevention protocols. People are expected to wear their face masks, sanitise hands, and take temperature measurements.

Create a web application for a private institution, XYZ, that will allow access into the premises of XYZ if a person is wearing a face mask, is willing to sanitise hands, and is having a temperature value less than 38 degrees Celsius. If a person is not meeting these requirements, access must be denied. The reason for denying access must be clearly spelt out.