# Chapter 7
# Working with Servlets

7.1 Introduction
7.2 organization of a web application
7.3 creating a web application (using netbeans)
7.4 creating a servlet
7.5 compiling and building the web application


## 7.1 Introduction


Working with Servlets is an important step in the process of application development and delivery through the Internet. A Servlet as explained earlier is a Java program that must be run by a Java Servlet engine on a Java enabled web server. The servlet's output is delivered to a web browser.

Sun GlassFish Enterprise Server that comes bundled with the NetBeans IDE has built-in support for running Java Servlets.


### What is a web application?

A web application, sometimes called a web app, is an application that is accessed using a web browser over a network such as the internet or an Intranet. While many web applications are written directly in PHP or perl, Java remains a commonly used programming language for writing Web applications.


## 7.2 organization of a web application


Before describing how to organize your source code directories, it is useful to examine the runtime organization of a web application. Prior to the Servlet API Specification, version 2.2, there was little consistency between server platforms. However, servers that conform to the 2.2 (or later) specification are required to accept a **Web Application Archive** in a standard format, which is discussed further below.

A web application is defined as a hierarchy of directories and files in a standard layout. Such a hierarchy can be accessed in its "unpacked" form, where each directory and file exists in the filesystem separately, or in a "packed" form known as a Web ARchive, or WAR file. The former format is more useful during development, while the latter is used when you distribute your application to be installed.

The system administrator deploys your application into a particular server, he or she assigns a context path to your application (a later section of this manual describes deployment on Tomcat). Thus, if the system administrator assigns your application to the context path/catalog, then a request URI referring to /catalog/index.html will retrieve the index.html file from your document root. The top-level directory of your web application hierarchy is also the *document root* of your application. Here, you will place the HTML files and JSP pages that comprise your application's user interface.
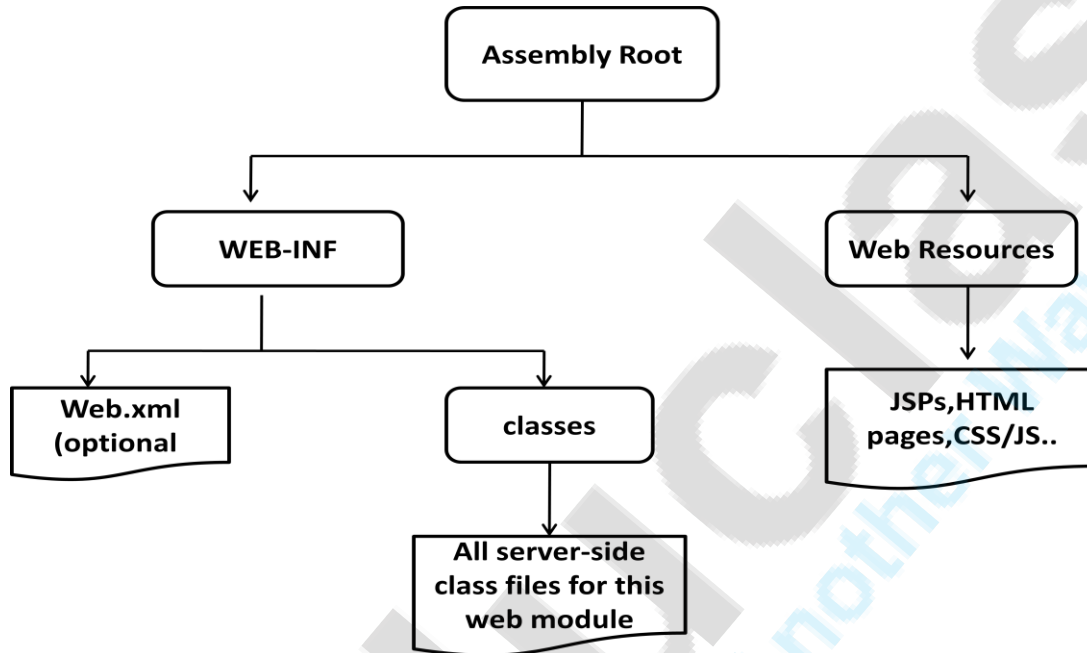


Diagram 7.1: The Web Application directory structure

7.2.1 Web Resources

In the Java EE architecture, web components and static Web content files such as images are called Web resources. A web module is the smallest deployable and usable unit of Web resources. A Java EE Web module corresponds to a Web application as defined in the Java Servlet specification.

The top-level directory of the Web application hierarchy is also the document root of the application. Here, all the HTML files and JSP pages can be placed that comprise the applications user interface.

7.2.2 WEB-INF

- **/WEB-INF/web.xml** - The *Web Application Deployment Descriptor* for your application. This is an XML file describing the servlets and other components that make up your application, along

with any initialization parameters and container-managed security constraints that you want the server to enforce for you. This file is discussed in more detail in the following subsection.

- **/WEB-INF/classes/** - This directory contains any Java class files (and associated resources) required for your application, including both servlet and non-servlet classes, that are not combined into JAR files. If your classes are organized into Java packages, you must reflect this in the directory hierarchy under `/WEB-INF/classes/`. For example, a Java class named `com.mycompany.mypackage.MyServlet` would need to be stored in a file named `/WEB-INF/classes/com/mycompany/mypackage/MyServlet.class.`

- **/WEB-INF/lib/** - This directory contains JAR files that contain Java class files (and associated resources) required for your application, such as third party class libraries or JDBC drivers.

### 7.2.3 Deployment Descriptor

As mentioned above, the `/WEB-INF/web.xml` file contains the Web Application Deployment Descriptor for your application. As the filename extension implies, this file is an XML document, and defines everything about your application that a server needs to know (except the *context path*, which is assigned by the system administrator when the application is deployed).

### 7.2.4 Context Path

When the application is being deployed on a particular Web server, a context path to the application has to be assigned. This means if the application is assigned to the context path as /bookshop, then a request URI referring to /bookshop/index.html will retries the index.html file from that document root.

### 7.3 creating a web application(using netbeans)

Let's create a simple HTML form that accepts user's name and submits the same to the servlet. The servlet in response welcomes the user based on the user name it receives.

The NetBeans IDE requires the Java Development Kit (JDK) to run properly. If you do not have any of the resources listed above, the JDK should be the first item that you download and install.

The NetBeans IDE Java Bundle includes Java Web and EE technologies, which are required for the application you build.

Let's create a new Web Application Project called **MyFirstServlet.**

Run the NetBeans IDE and select File -> New Project to create a new Web Application project in it as shown in Diagram 7.2
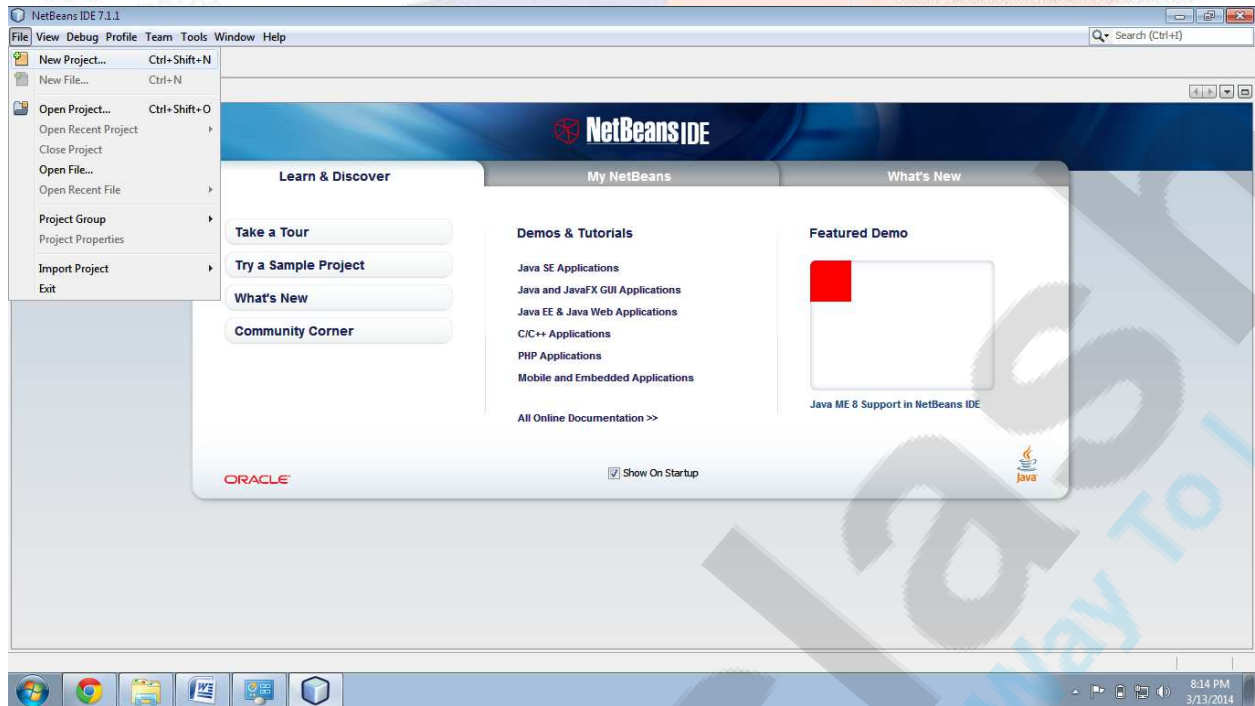
Diagram 7.2: File-> New Project

New Project dialog box appears as shown in diagram select Java Web option available under categories list and Web Application option available under Projects list as shown in below diagram:
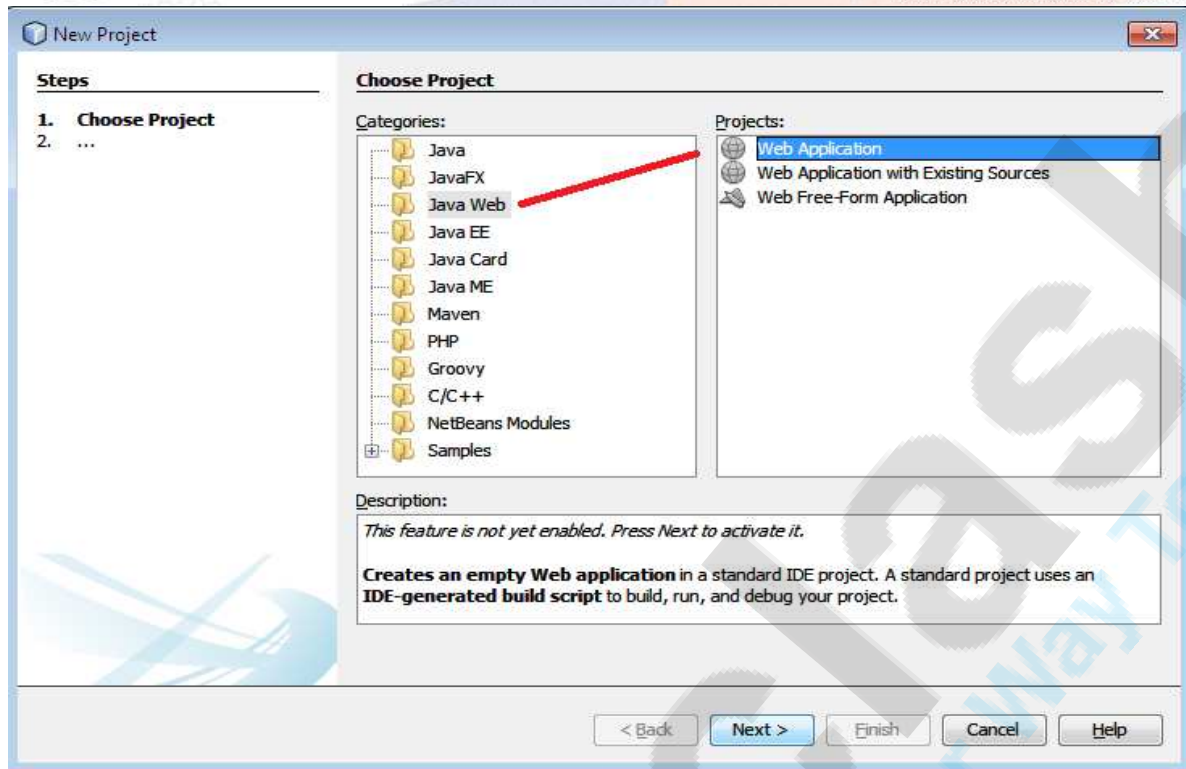
Diagram 7.3 : New Project dialog box

Click **Next.** New Web Application dialog box appears as shown in below diagram.

Enter name of the Web application as MyFirstServlet as shown in diagram.

Diagram 7.4: Entering the name of the Web application

Click **Next.** Server and Setting section of the New Web Application dialog box appears as shown in below diagram i.e. GlassFish V3 Domain as the web server, the Java EE 6 Web as the Java EE version and the Context path as shown in below diagram.
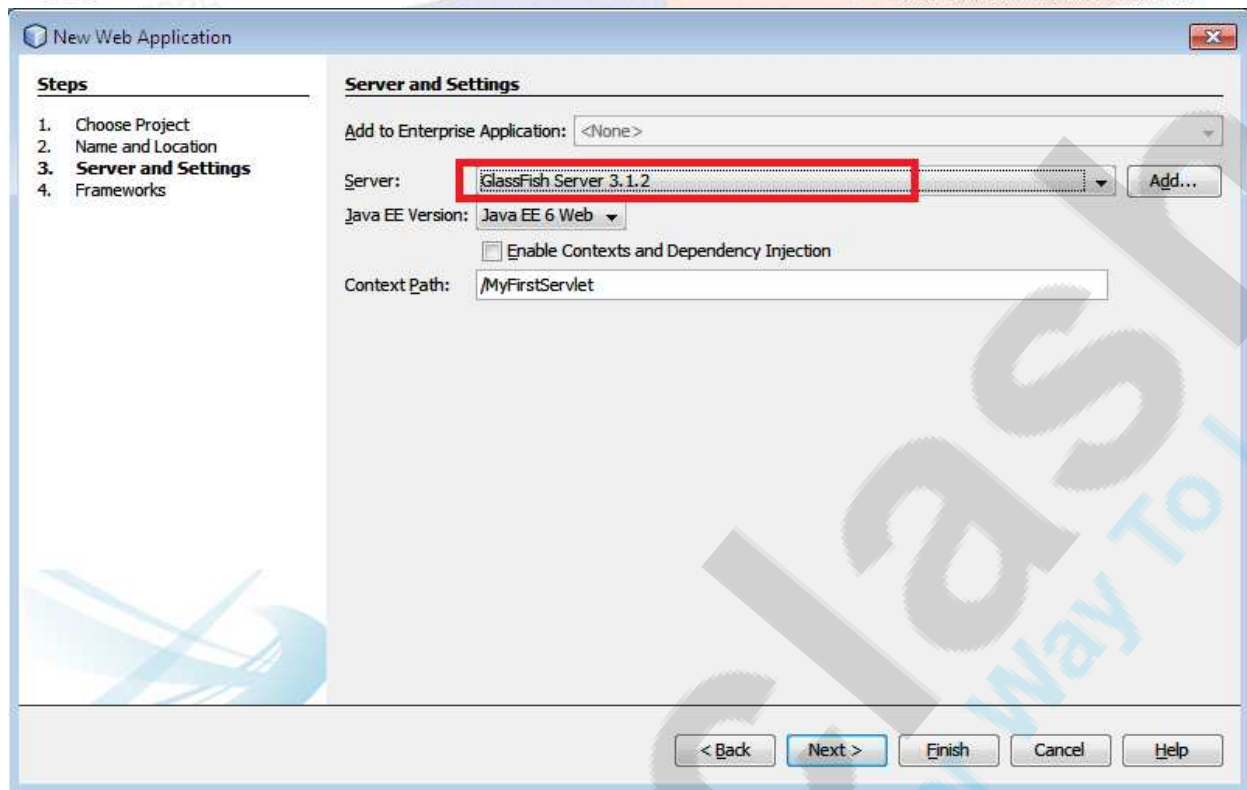
Diagram 7.5: selecting the server, Java EE version and the context path

Click **Next**. Do not choose a framework in the frameworks section of the New Web Application dialog box.

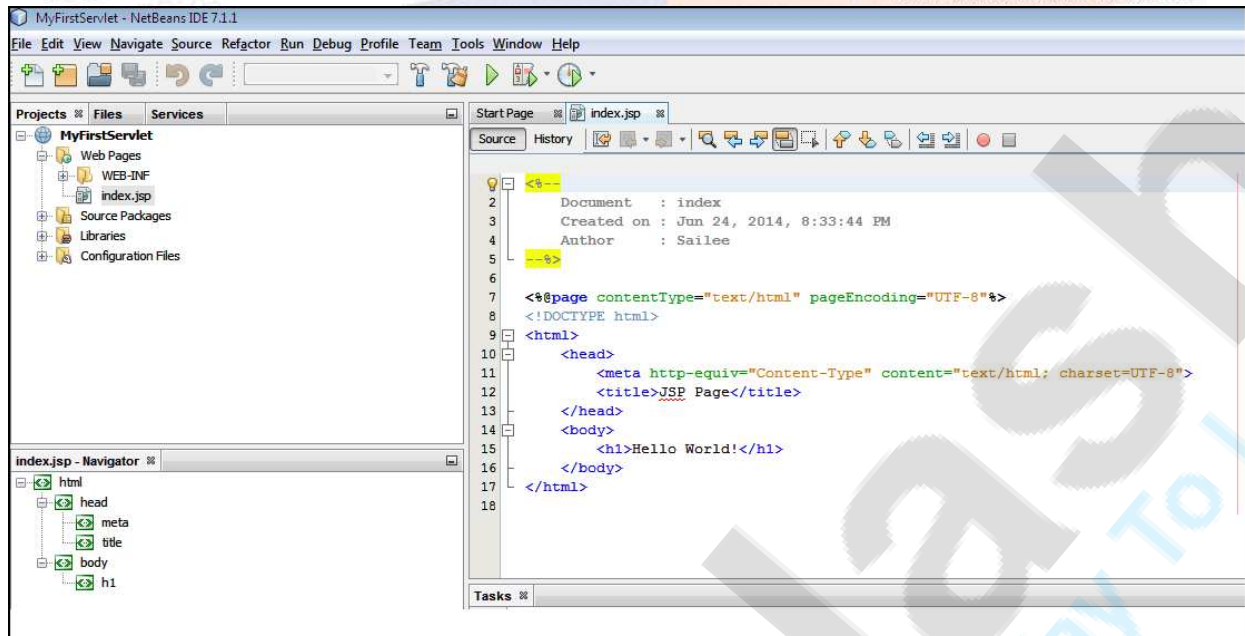Click **Finish**.The MyFirstServlet application is created in the NetBeans IDE as shown in below diagram.

Diagram 7.6 : MyFirstServlet in NetBeans IDE

7.3 Creating A Servlet

WelcomeServlet.java

Create WelcomeServlet.java using NetBeans IDE.

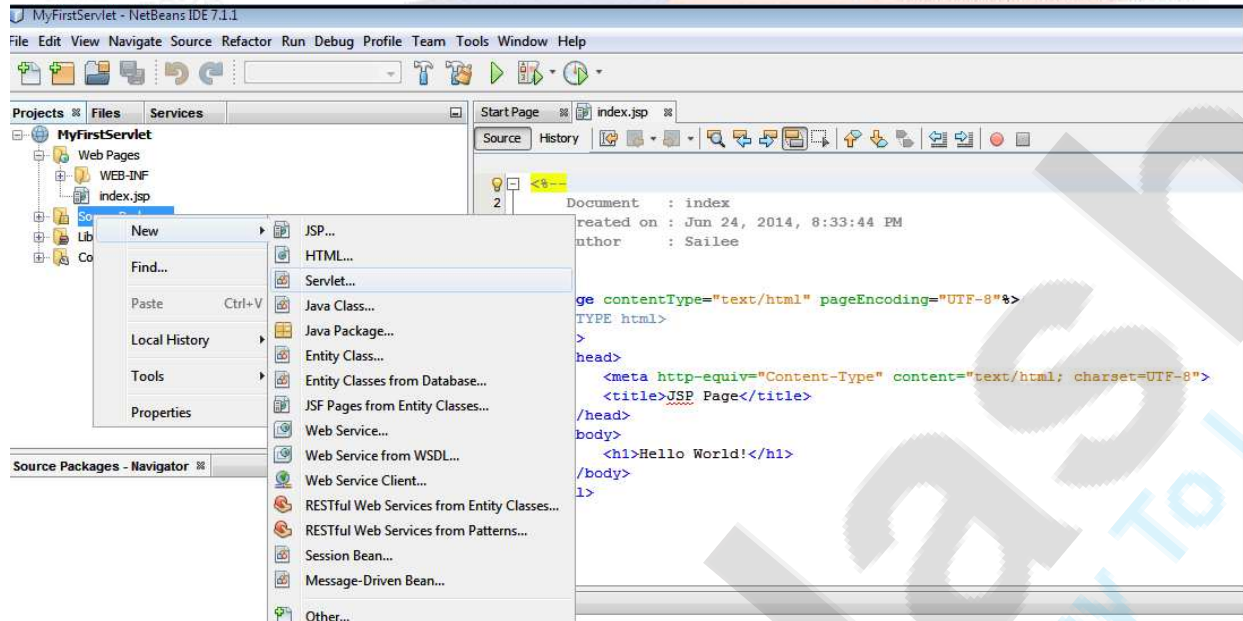Right click Source Package directory, select New->Servlet as shown in below diagram.

Diagram 7.7 : Crating the servlet

New servlet dialog box appears as shown in below diagram. Enter WelcomeServlet in the Class Name textbox and enter servlet in the Package textbox as shown in diagram.

Diagram 7.8 : Entering the class name and the package.

Click **Next**. Configure Servlet Deployment section of the New Servlet dialog box appears as shown in diagram. Keep the defaults but select the option Add information to deployment descriptor(web.xml)as shown in diagram.
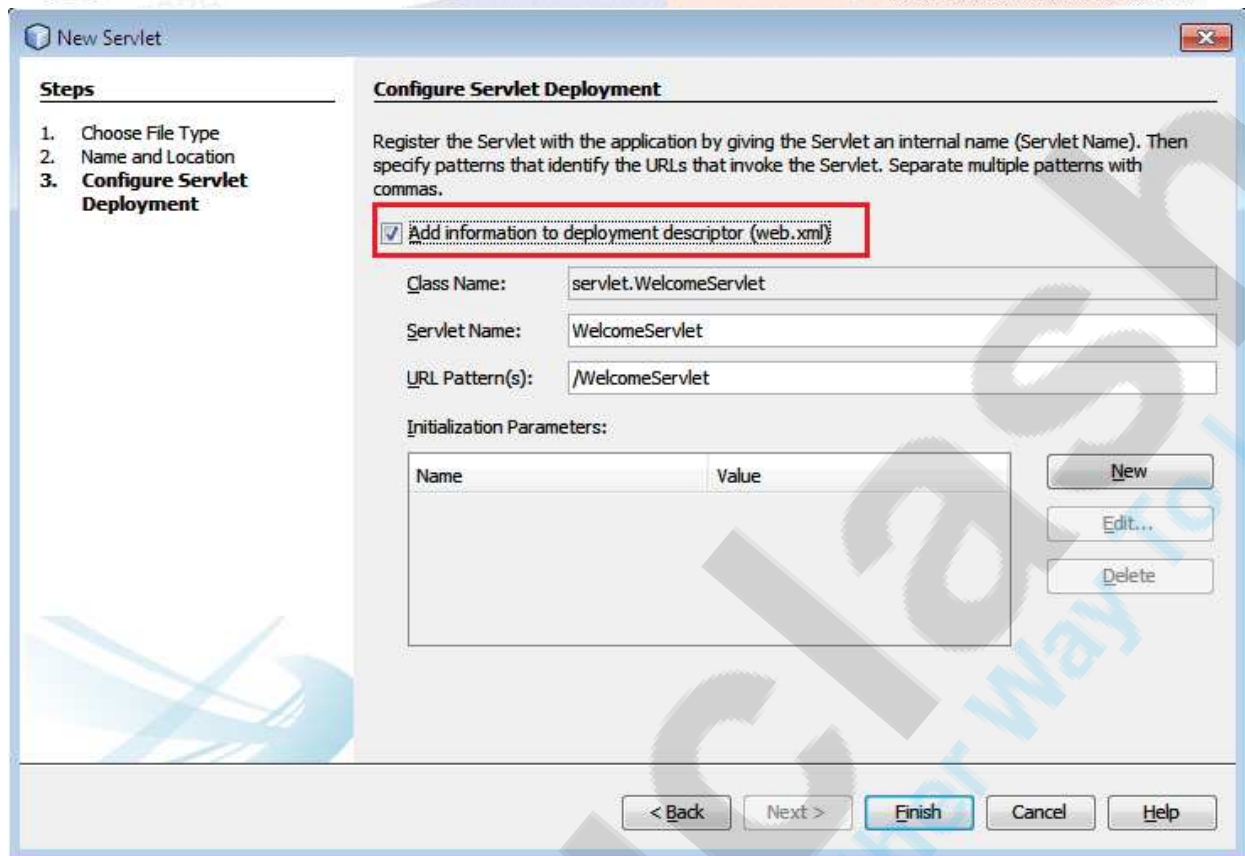
Diagram 7.8: Configuring the Servlet deployment.

Click **Finish**.

The Servlet named WelcomeServlet.java is created under the servlet package.

WelcomeServlet.java is available with the default skeleton created by NetBeans IDE which needs to be modified to hold the actual code.

Before that, for demonstrating the deployment and execution of a web application,

Here an HTML form has been added. This form accepts User's Name using a text box and passes it to a Servlet called WelcomeServlet for further processing.

**Index.jsp**

```jsp
<%--

    Document    : index

    Created on : Jun 24, 2014, 8:33:44 PM

    Author      : Subodh Deolekar

--%>


<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

    <head>

        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

        <title>Welcome To Servlet</title>

    </head>

    <body>

        <h1>Hello World!</h1>

        <form method="POST" action="WelcomeServlet">

            <label for="name" title="Enter the name">Name:</label>

            <input type="text" id="txtName" name="txtName"/><br><br>

            <input type="submit" value="Submit"/>

        </form>

    </body>

</html>
```

WelcomeServlet.java

```java
package servlet;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletConfig;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;


/**
 *
 * @author Subodh Deolekar
 */
public class WelcomeServlet extends HttpServlet {


    /**
     * Processes requests for both HTTP
     * <code>GET</code> and
     * <code>POST</code> methods.
     *
     * @param request servlet request
     * @param response servlet response
```

```java
 * @throws ServletException if a servlet-specific error occurs

 * @throws IOException if an I/O error occurs

 */

public void init(ServletConfig config)throws ServletException{

    super.init(config);

}

protected void processRequest(HttpServletRequest request, HttpServletResponse response)

        throws ServletException, IOException {

    response.setContentType("text/html;charset=UTF-8");

    PrintWriter out = response.getWriter();

    try {

        /*

         * TODO output your page here. You may use following sample code.

         */

        out.println("<html>");

        out.println("<head>");

        out.println("<title>WelcomeServlet</title>");

        out.println("</head>");

        out.println("<body>");

        out.println("<h1>Welcome " + request.getParameter("txtName") + "</h1>");

        out.println("<a href='index.jsp'>Click here to return to home page</a>");

        out.println("</body>");

        out.println("</html>");
```

```
    } finally {

        out.close();

    }

}


// <editor-fold defaultstate="collapsed" desc="HttpServlet
methods. Click on the + sign on the left to edit the code.">

/**

 * Handles the HTTP

 * <code>GET</code> method.

 *

 * @param request servlet request

 * @param response servlet response

 * @throws ServletException if a servlet-specific error occurs

 * @throws IOException if an I/O error occurs

 */

@Override

protected void doGet(HttpServletRequest request,
HttpServletResponse response)

        throws ServletException, IOException {

    processRequest(request, response);

}


/**

 * Handles the HTTP

 * <code>POST</code> method.
```

```
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }


    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
    public void destroy(){
    }
}
```

7.4 Compiling and Building The Web Application

In NetBeans, while creating the servlet since the option Add Information To Deployment Descriptor(web.xml) was chosen, the web.xml file is automatically created with the servlet details.

Deployment Descriptor[web.xml]

Web.xml is used by the server during deployment of a Web module. It describes the web components used by that web module, environment variables and security requirements.

This information is stored as WEB-INF/web.xml.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

    <servlet>

        <servlet-name>WelcomeServlet</servlet-name>

        <servlet-class>servlet.WelcomeServlet</servlet-class>

    </servlet>

    <servlet-mapping>

        <servlet-name>WelcomeServlet</servlet-name>

        <url-pattern>/WelcomeServlet</url-pattern>

    </servlet-mapping>

    <session-config>

        <session-timeout>

            30

        </session-timeout>

    </session-config>

</web-app>
```

**Web.xml** is a deployment descriptor XML file that consists of several elements:

Depending on the components in the Web application, there is no need to include all of the elements to configure and deploy the web application. <servlet> element is the most important element for describing the Servlets in an application.

The following sections describe the deployment descriptor elements defined in the web.xml file under the root element `<web-app>`:

- The `<servlet>` element:
  The `servlet` element contains the declarative data of a servlet.
  - ✓ `<servlet-name>` Defines the canonical name of the servlet, used to reference the servlet definition elsewhere in the deployment descriptor.

  - ✓ `<servlet-class>` The fully-qualified class name of the servlet.

- The `<servlet-mapping>` element
  The `servlet-mapping` element defines a mapping between a servlet and a URL pattern.

  - ✓ `<servlet-name>`
    The name of the servlet to which you are mapping a URL pattern. This name corresponds to the name you assigned a servlet in a `<servlet>`declaration tag.

  - ✓ `<url-pattern>`
    Describes a pattern used to resolve URLs. The portion of the URL after the http://host:port+ WebAppNameis compared to the `<url-pattern>`by WebLogic Server. If the patterns match, the servlet mapped in this element will be called.

- The `<session-config>` element
  The `session-config` element defines the session attributes for this Web application.

  - ✓ `<session-timeout>`
    The number of minutes after which sessions in this Web application expire.

- The `<welcome-file-list>` Element
  The optional `welcome-file-list` element contains an ordered list of `welcome-file` elements.

When the URL request is a directory name, WebLogic Server serves the first file specified in this element. If that file is not found, the server then tries the next file in the list.

✓ `<welcome-file>`

File name to use as a default welcome file, such as index.html.

## 7.5 Compiling and Building Using NetBeans

Since the servlet i.e WelcomeServlet.java the jsp i.e. index.jsp and web.xml are in place, it's time to compile and build the web application.

Right click MYFirstServlet project and select Build as shown in diagram
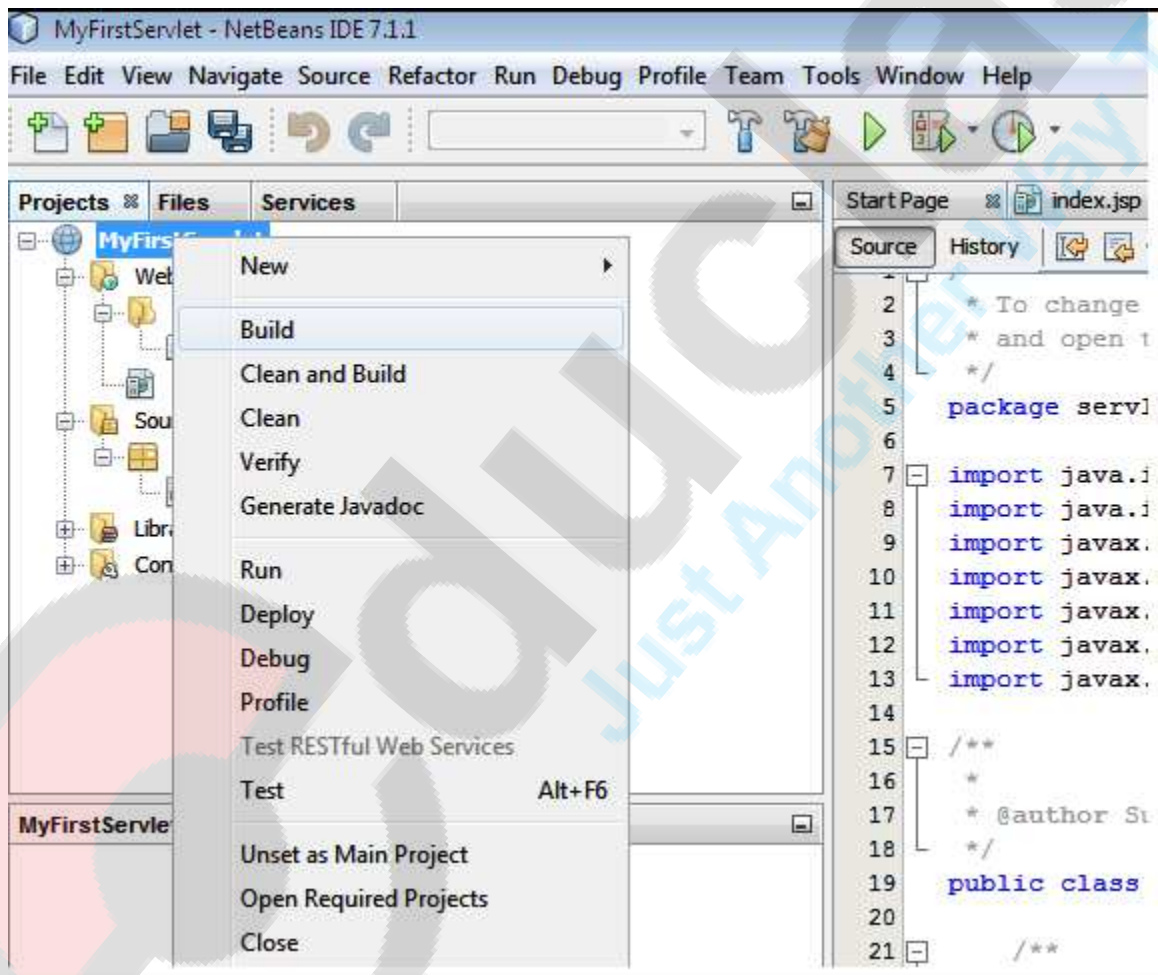


Diagram 7.9: Building the project

Once the Build menu item is clicked, the details about the compilation and building of the MyFirstServlet web application appear in the Output- MyFirstServlet(dist) window.

**Running the web application:**

Once the compilation and building of the web application is done, run the application.

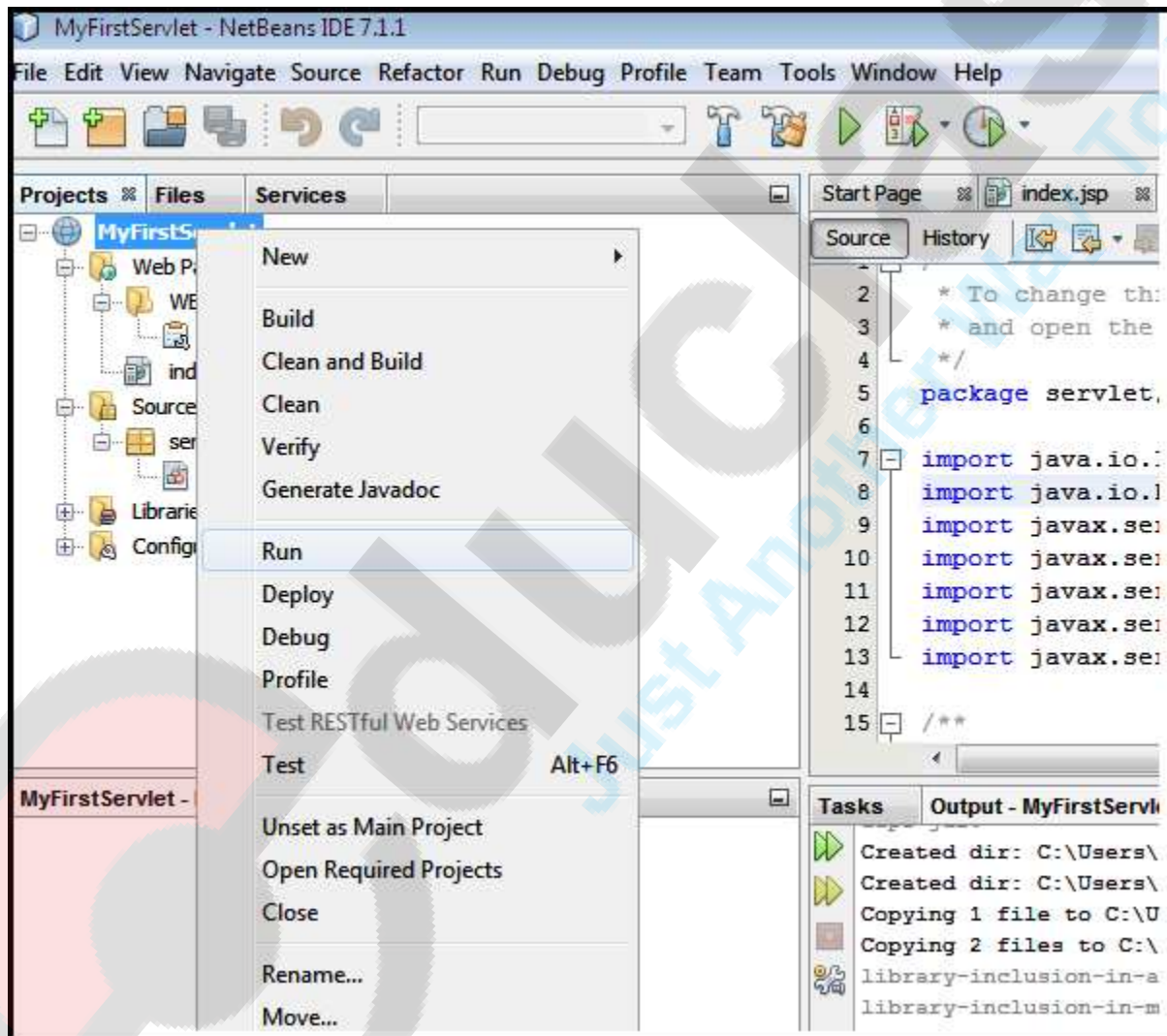Right click the MyFirstServlet project and select Run as shown in diagram:



Diagram 7.10: Running the project

Once Run menu item is clicked, the GlassFish v3 Domain is started by the NetBeans IDE as shown in diagram:
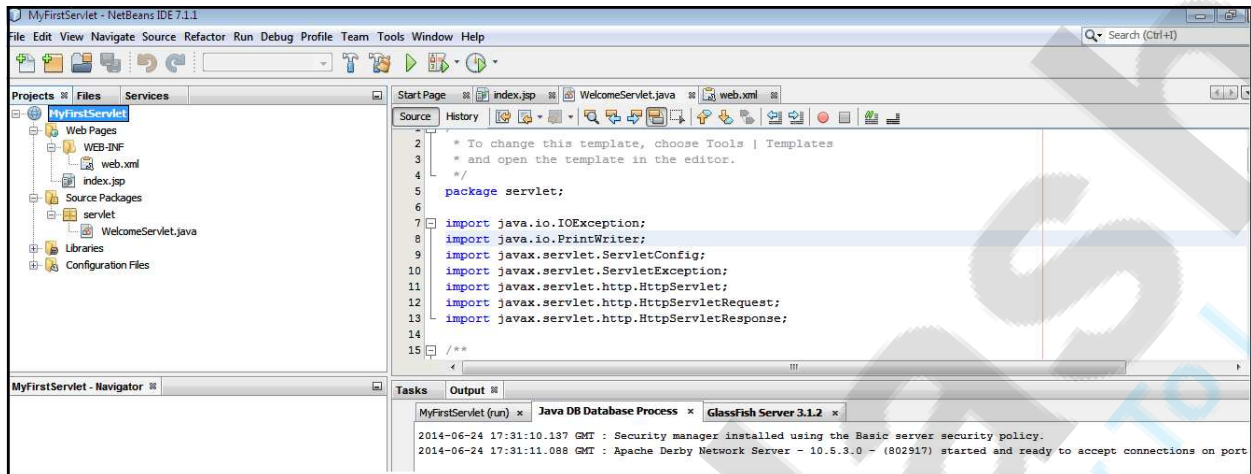


Diagram 7.11:The GlassFish v3 Domain window
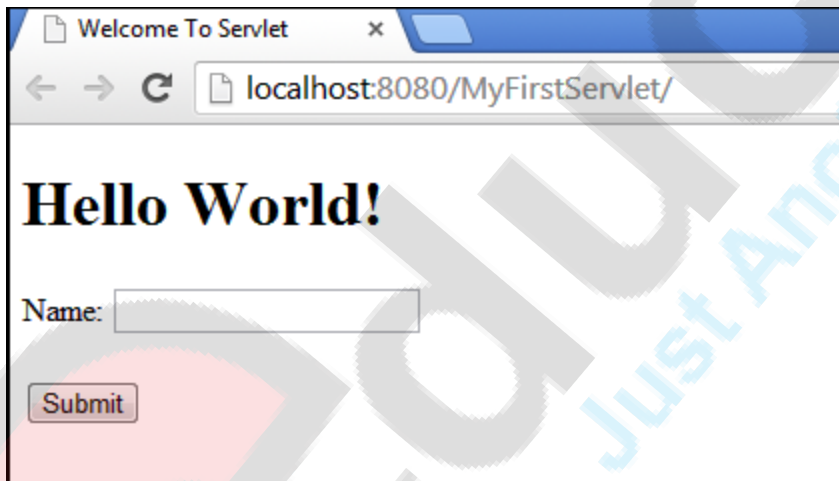
Index.jsp is served in the web browser as shown in diagram



Diagram 7.12: the application run in the web browser

Enter the name in **Name** text box as shown in diagram

Diagram 7.13: Entering the name

Click **Submit.**

WelcomeServlet is called, which displays the welcome message along with the name entered as shown in diagram:
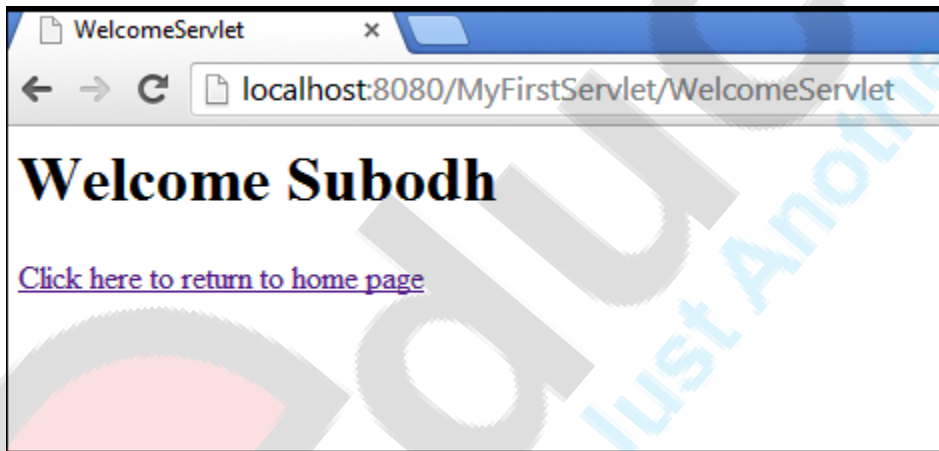


Diagram7.14: Welcome message displayed

Click "Click here to return to home page" to return to the main page i.e. the index.jsp page.