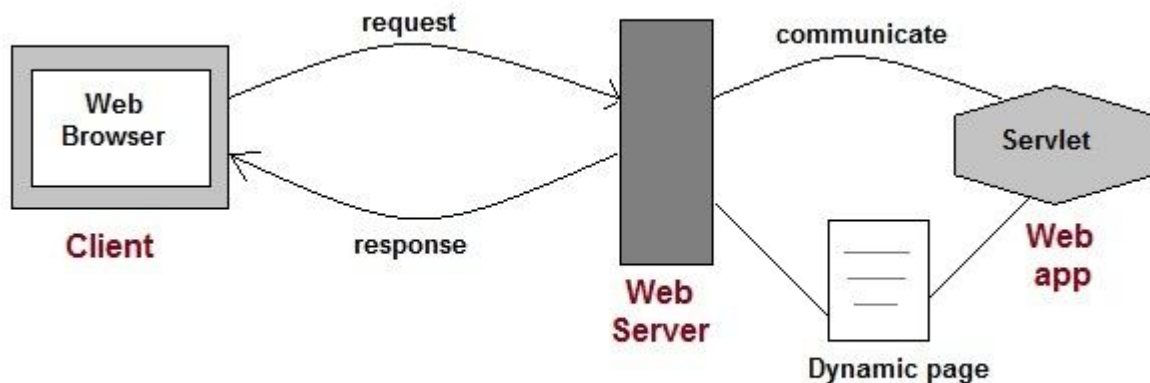


Introduction to Servlet

Servlet Technology is used to create web applications. **Servlet** technology uses Java language to create web applications.

Web applications are helper applications that resides at web server and build dynamic web pages. A dynamic page could be anything like a page that randomly chooses picture to display or even a page that displays the current time.

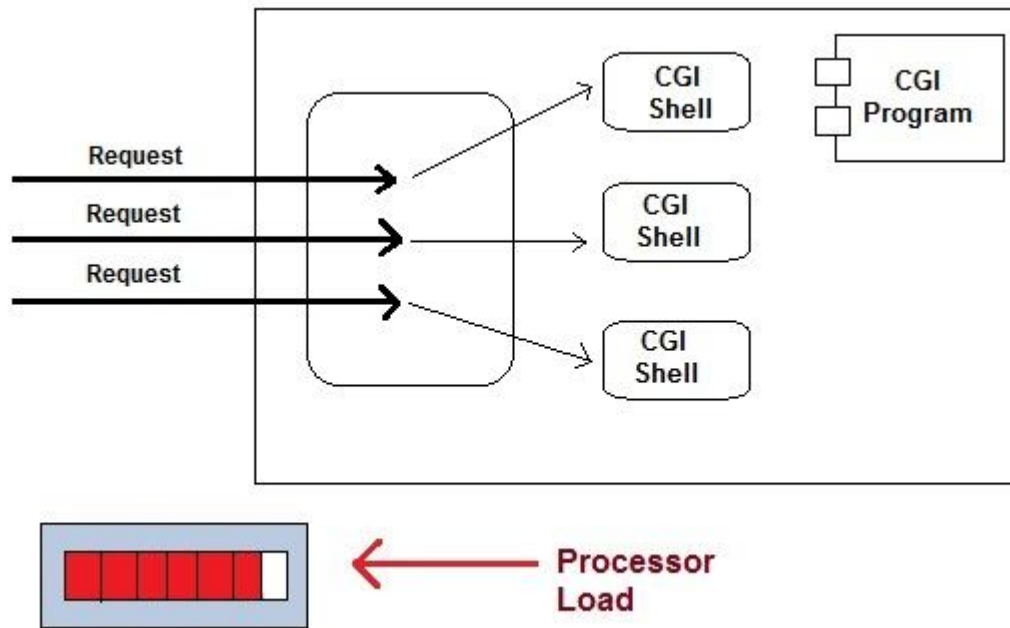


As Servlet Technology uses Java, web applications made using Servlet are **Secured, Scalable** and **Robust**.

CGI (Common Gateway Interface)

Before Servlets, CGI(Common Gateway Interface) programming was used to create web applications. Here's how a CGI program works :

- User clicks a link that has URL to a dynamic page instead of a static page.
- The URL decides which CGI program to execute.
- Web Servers run the CGI program in separate OS shell. The shell includes OS environment and the process to execute code of the CGI program.
- The CGI response is sent back to the Web Server, which wraps the response in an HTTP response and send it back to the web browser.



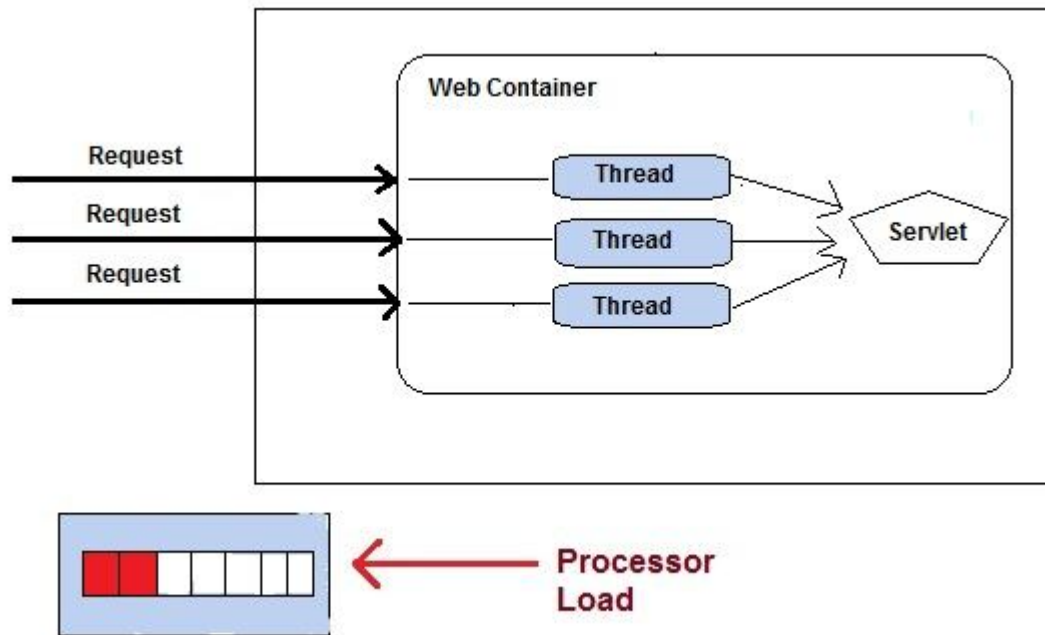
Drawbacks of CGI programs

- High response time because CGI programs execute in their own OS shell.
- CGI is not scalable.
- CGI programs are not always secure or object-oriented.
- It is Platform dependent.

Because of these disadvantages, developers started looking for better CGI solutions. And then Sun Microsystems developed **Servlet** as a solution over traditional CGI technology.

Advantages of using Servlets

- Less response time because each request runs in a separate thread.
- Servlets are scalable.
- Servlets are robust and object oriented.
- Servlets are platform independent.



Servlet API

Servlet API consists of two important packages that encapsulates all the important classes and interface, namely :

- `javax.servlet`
- `javax.servlet.http`

Some Important Classes and Interfaces of `javax.servlet`

INTERFACES	CLASSES
Servlet	ServletInputStream
ServletContext	ServletOutputStream
ServletConfig	ServletRequestWrapper
ServletRequest	ServletResponseWrapper

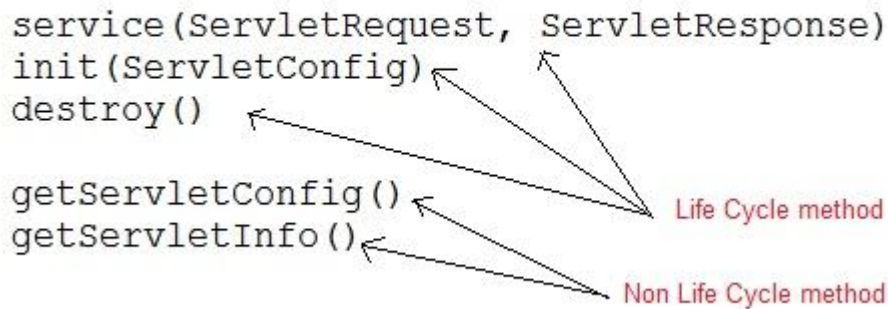
ServletResponse	ServletRequestEvent
ServletContextListener	ServletContextEvent
RequestDispatcher	ServletRequestAttributeEvent
SingleThreadModel	ServletContextAttributeEvent
Filter	ServletException
FilterConfig	UnavailableException
FilterChain	GenericServlet
ServletRequestListener	

Some Important Classes and Interface of javax.servlet.http

CLASSES and INTERFACES	
HttpServlet	HttpServletRequest
HttpServletResponse	HttpSessionAttributeListener
HttpSession	HttpSessionListener
Cookie	HttpSessionEvent

Servlet Interface

Servlet Interface provides five methods. Out of these five methods, three methods are **Servlet life cycle** methods and rest two are non life cycle methods.



GenericServlet Class

GenericServlet is an abstract class that provides implementation of most of the basic servlet methods. This is a very important class.

Methods of GenericServlet class

- `public void init(ServletConfig)`
- `public abstract void service(ServletRequest request, ServletResponse response)`
- `public void destroy()`
- `public ServletConfig getServletConfig()`
- `public String getServletInfo()`
- `public ServletContext getServletContext()`
- `public String getInitParameter(String name)`
- `public Enumeration getInitParameterNames()`
- `public String getServletName()`
- `public void log(String msg)`
- `public void log(String msg, Throwable t)`

HttpServlet class

HttpServlet is also an abstract class. This class gives implementation of various `service()` methods of **Servlet** interface.

To create a servlet, we should create a class that extends **HttpServlet** abstract class. The Servlet class that we will create, must not override `service()` method. Our servlet class will override only the `doGet()` and/or `doPost()` methods.

The `service()` method of **HttpServlet** class listens to the Http methods (GET, POST etc) from request stream and invokes `doGet()` or `doPost()` methods based on Http Method type.

How a Servlet Application works

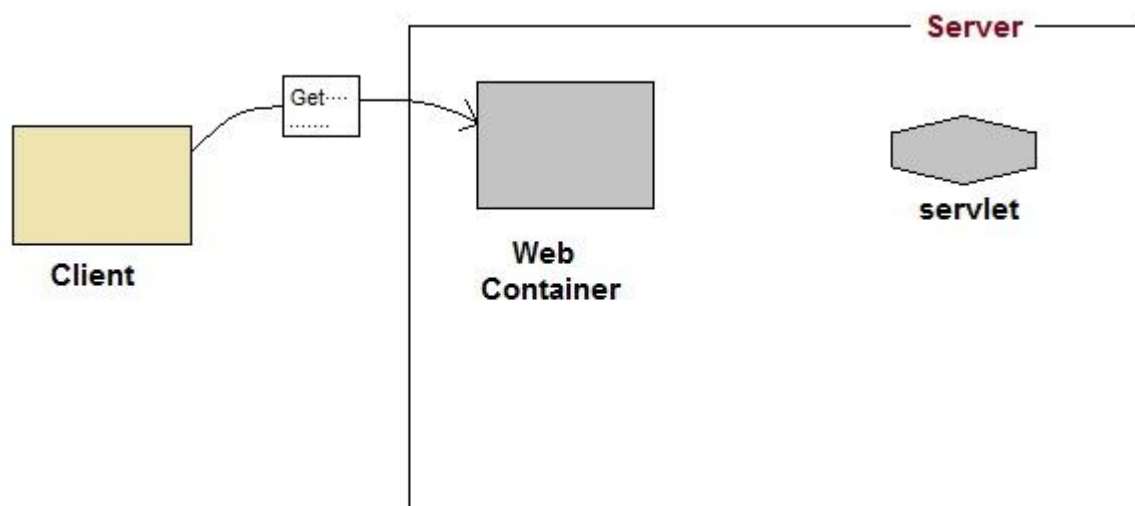
Web container is responsible for managing execution of servlets and JSP pages for Java EE application.

When a request comes in for a servlet, the server hands the request to the Web Container. **Web Container** is responsible for instantiating the servlet or creating a new thread to handle the request. Its the job of Web Container to get the request and response to the servlet. The container creates multiple threads to process multiple requests to a single servlet.

Servlets don't have a main() method. Web Container manages the life cycle of a Servlet instance.

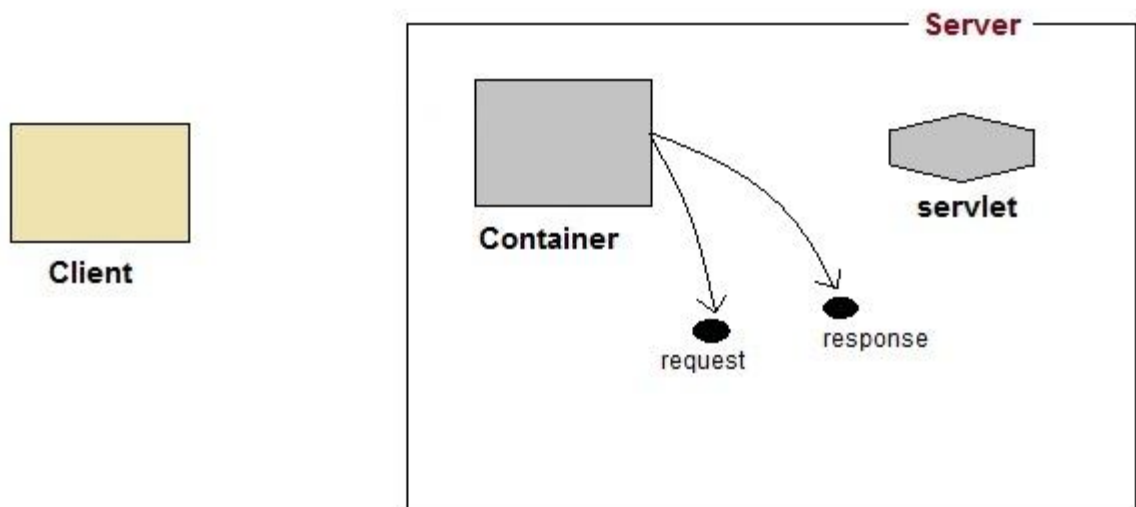
Quick Revision on How a Servlet works

1. User sends request for a servlet by clicking a link that has URL to a servlet.

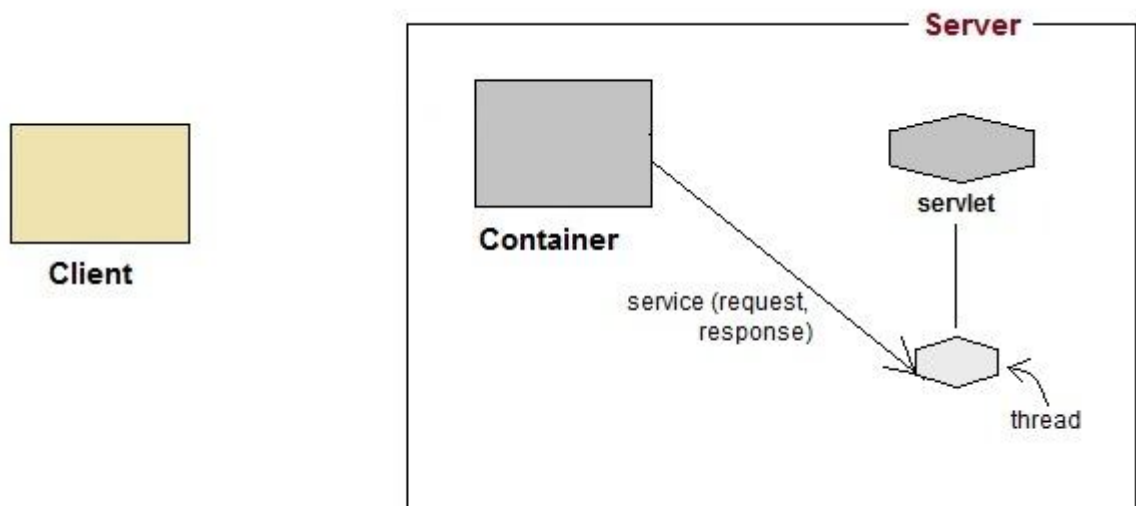


2. The container finds the servlet using **deployment descriptor** and creates two objects :

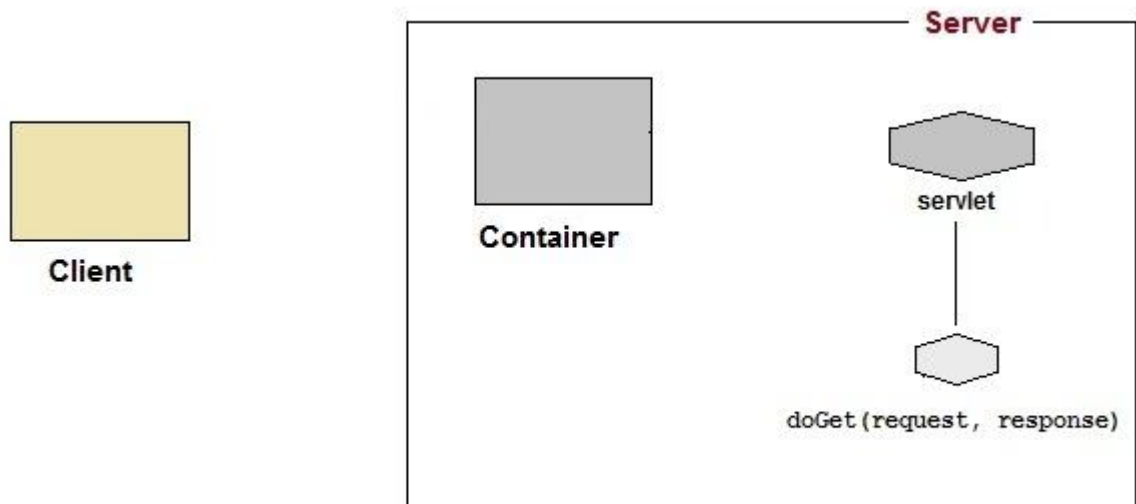
- a. **HttpServletRequest**
- b. **HttpServletResponse**



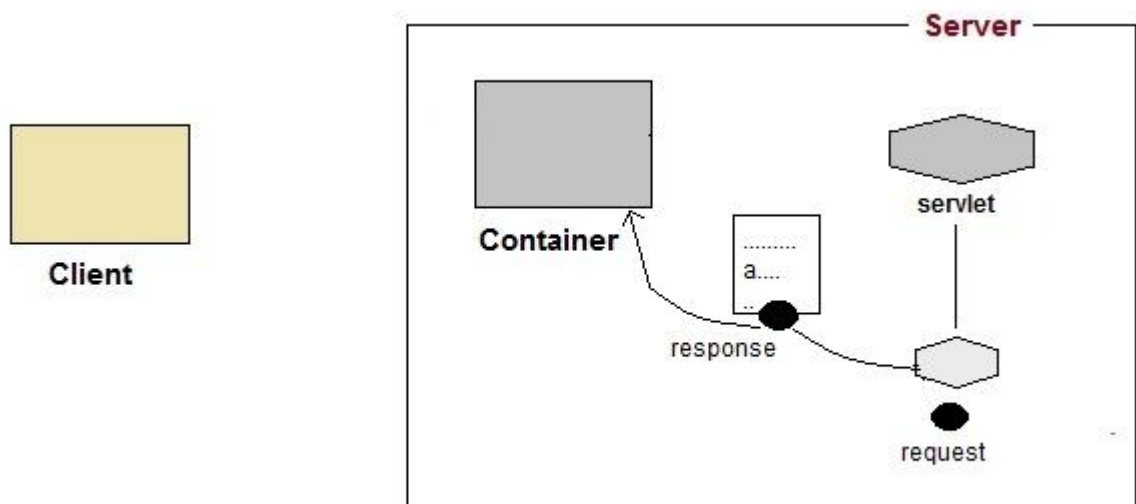
3. Then the container creates or allocates a thread for that request and calls the Servlet's **service()** method and passes the **request**, **response** objects as arguments.



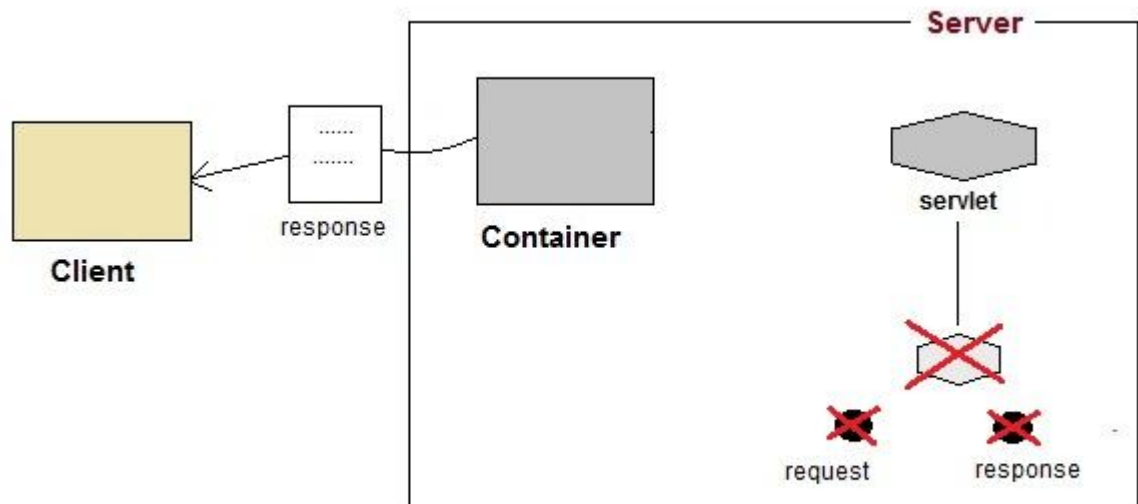
4. The **service()** method, then decides which servlet method, **doGet()** or **doPost()** to call, based on **HTTP Request Method**(Get, Post etc) sent by the client. Suppose the client sent an HTTP GET request, so the **service()** will call Servlet's **doGet()** method.



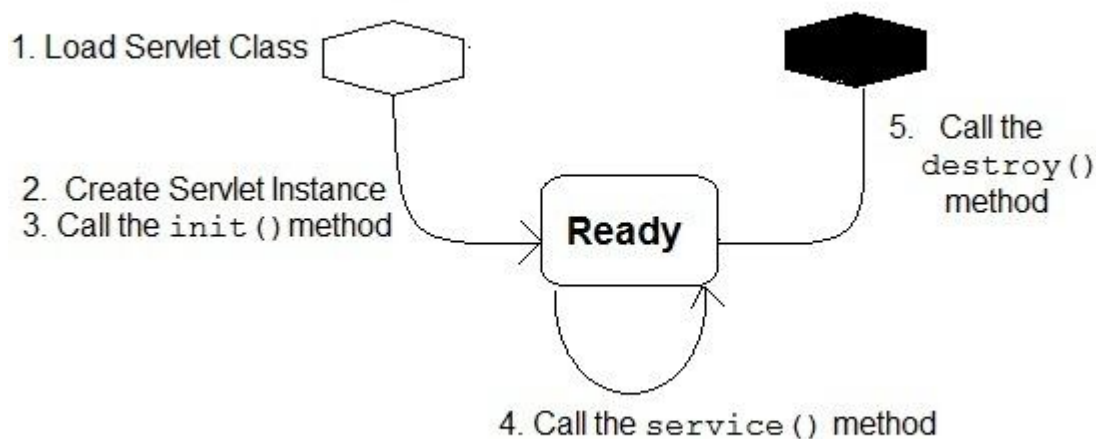
5. Then the Servlet uses response object to write the response back to the client.



6. After the `service()` method is completed the **thread** dies. And the request and response objects are ready for **garbage collection**.



Servlet Life Cycle



1. **Loading Servlet Class** : A Servlet class is loaded when first request for the servlet is received by the Web Container.
2. **Servlet instance creation** :After the Servlet class is loaded, Web Container creates the instance of it. Servlet instance is created only once in the life cycle.
3. **Call to the `init()` method** : `init()` method is called by the Web Container on servlet instance to initialize the servlet.

Signature of `init()` method :

```
public void init(ServletConfig config) throws ServletException
```

4. **Call to the service() method** : The containers call the **service()** method each time the request for servlet is received. The service() method will then call the **doGet()** or **doPost()** methods based on the type of the HTTP request, as explained in previous lessons.

Signature of service() method :

```
public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException
```

5. **Call to destroy() method**: The Web Container call the **destroy()** method before removing servlet instance, giving it a chance for cleanup activity.

Creating First Servlet Application using Netbeans IDE

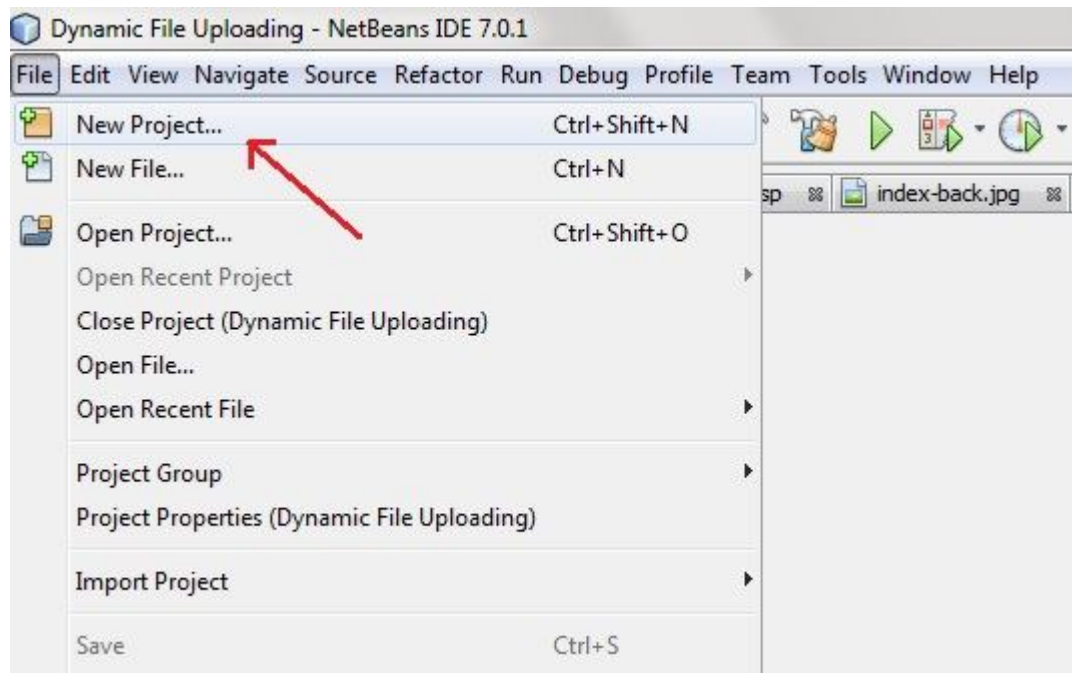
In the last lesson we created our first Servlet Application but without using any IDE. An IDE is Integrated Development Environment, and it makes creating applications a lot easier. We will learn how to create Servlet applications on NetBeans IDE and Eclipse IDE. Then you can decide which one, you want to use.

Using Integrated Development Environment(IDE) is the easiest way to create Servlet Applications. An IDE is a software application that provides facilities to computer programmers for software development. **Eclipse, MyEclipse, Netbeans** are example of some popular Java IDE.

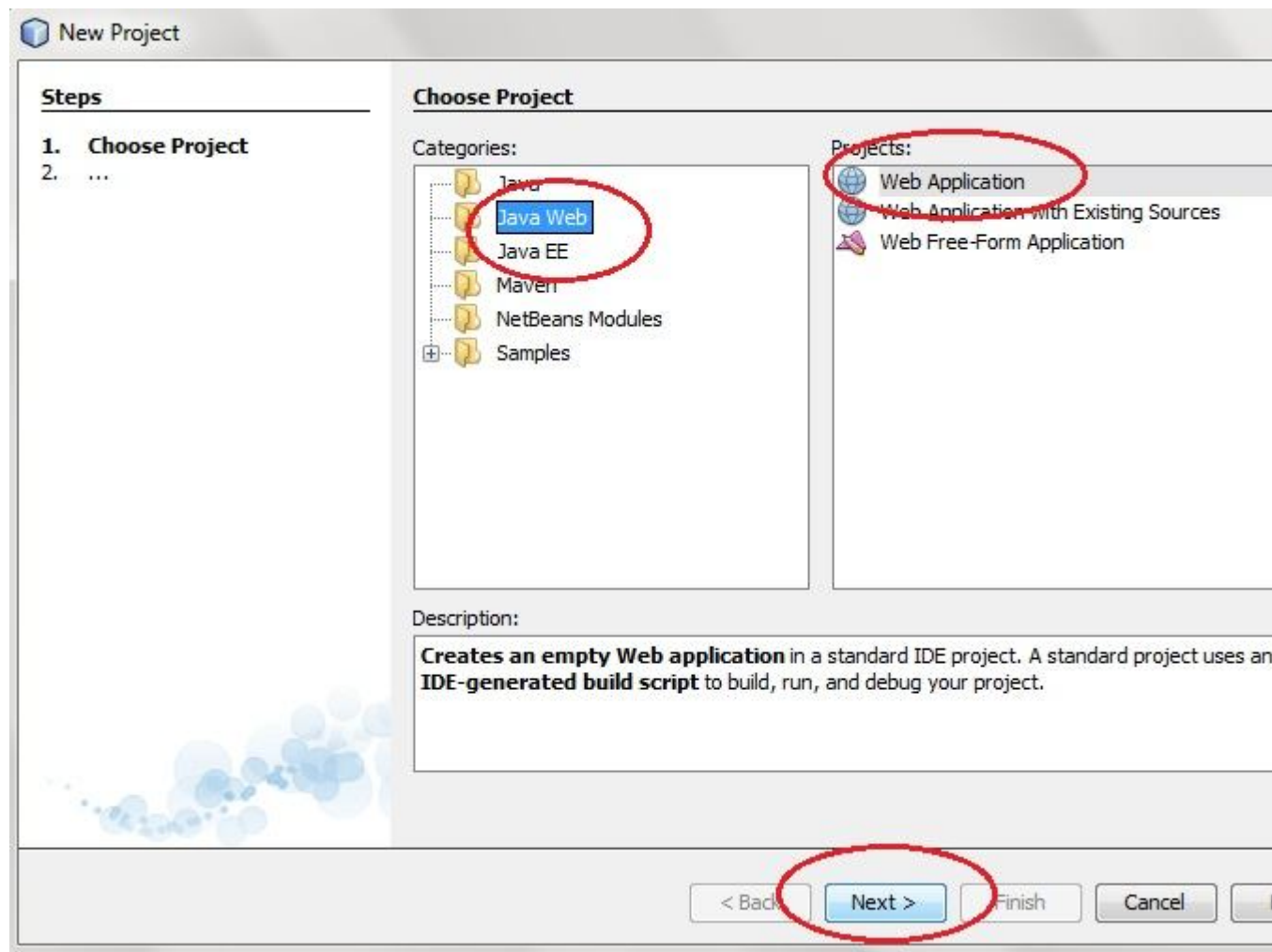
Steps to Create Servlet Application in Netbeans IDE

To create a servlet application in Netbeans IDE, you will need to follow the following (simple) steps :

1. Open Netbeans IDE, Select **File -> New Project**



-
2. Select **Java Web** -> **Web Application**, then click on Next,



-
3. Give a name to your project and click on Next,

New Web Application

Steps

1. Choose Project
- 2. Name and Location**
3. Server and Settings
4. Frameworks

Name and Location

Project Name:

Project Location:

Project Folder:

☐ Use Dedicated Folder for Storing Libraries

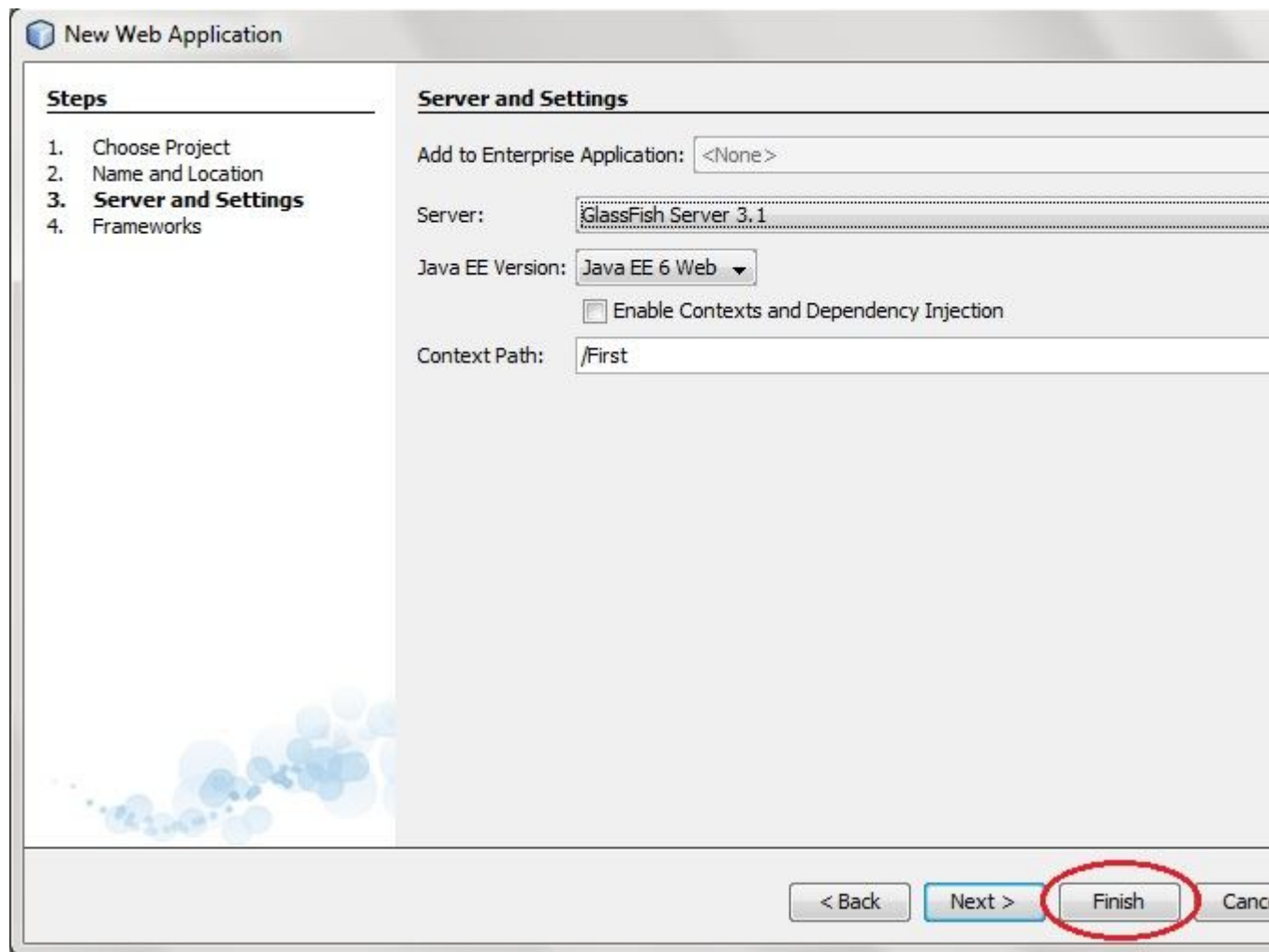
Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

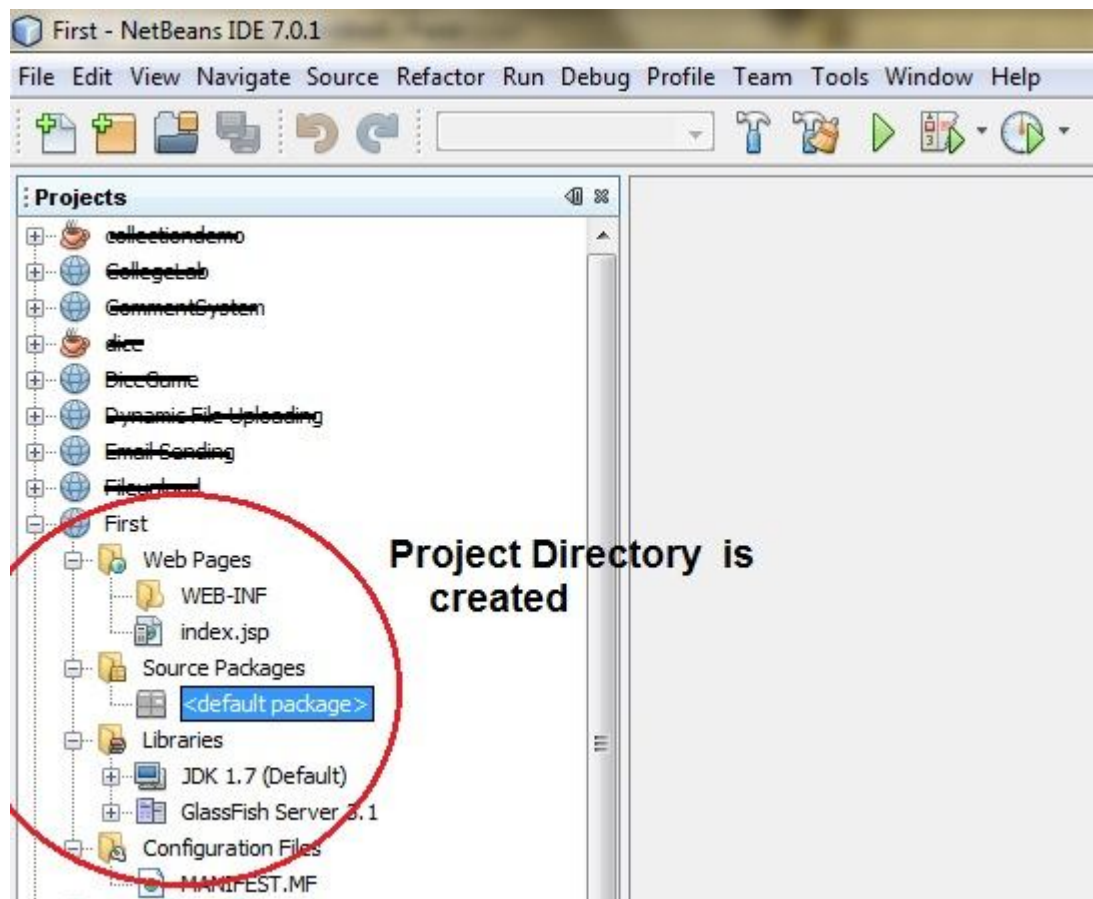
☒ Set as Main Project

< Back **Next >** Finish Cancel

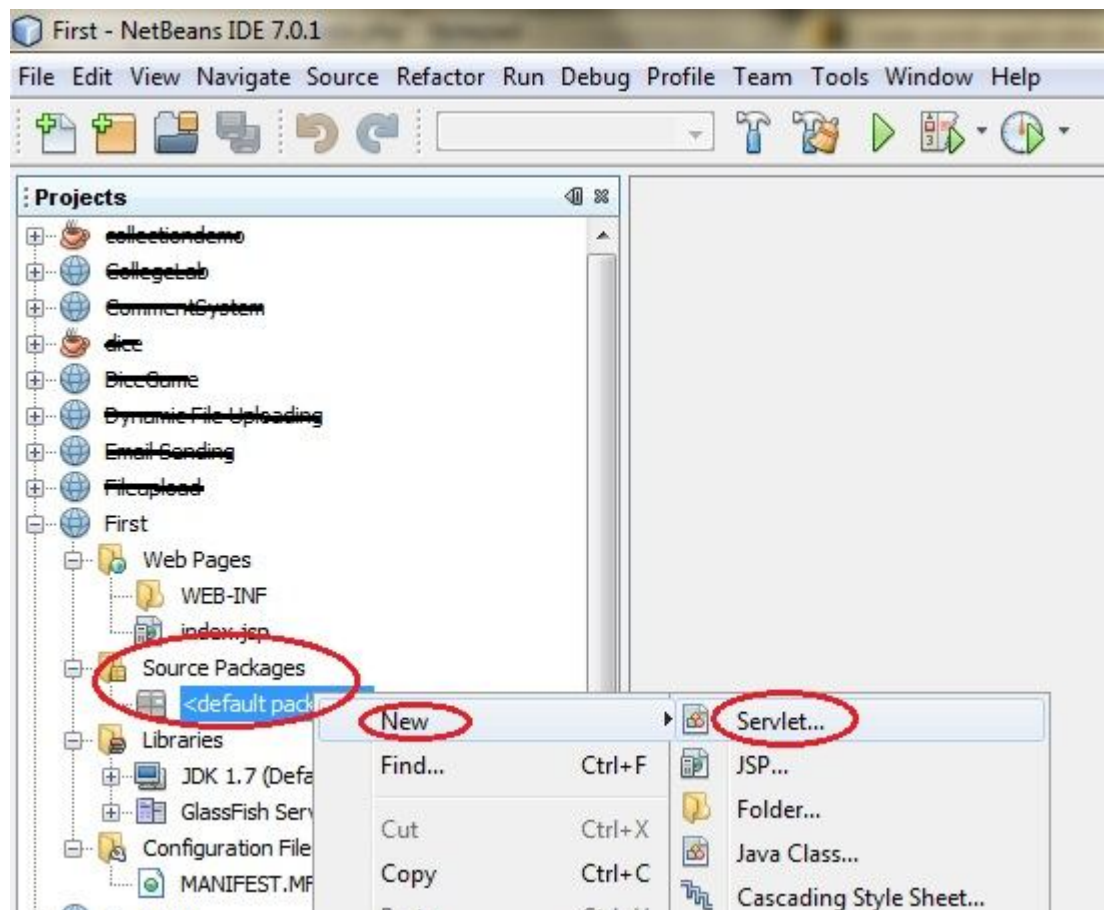
4. and then, Click **Finish**



-
5. The complete directory structure required for the Servlet Application will be created automatically by the IDE.



6. To create a Servlet, open **Source Package**, right click on **default packages** -> **New** -> **Servlet**.



7. Give a Name to your Servlet class file,

New Servlet

Steps

1. Choose File Type

2. Name and Location

3. Configure Servlet Deployment

Name and Location

Class Name

MyServlet

Project:

First

Location:

Source Packages

Package:

Created File:

C:\Users\Abhijit\Documents\NetBeansProjects\First\src\java\MySer

Warning: It is highly recommended that you do NOT place Java classes in the default package.

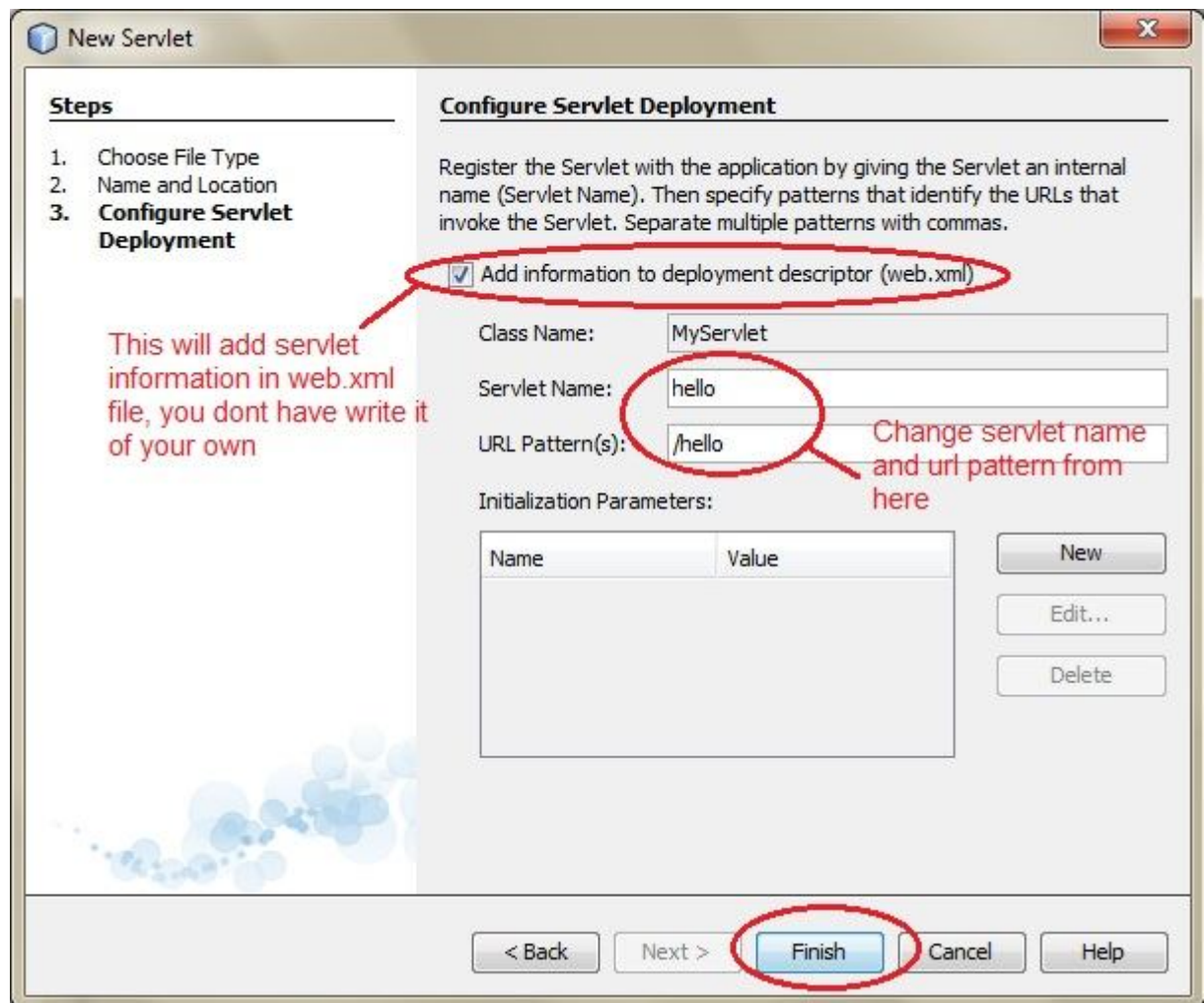
< Back

Next >

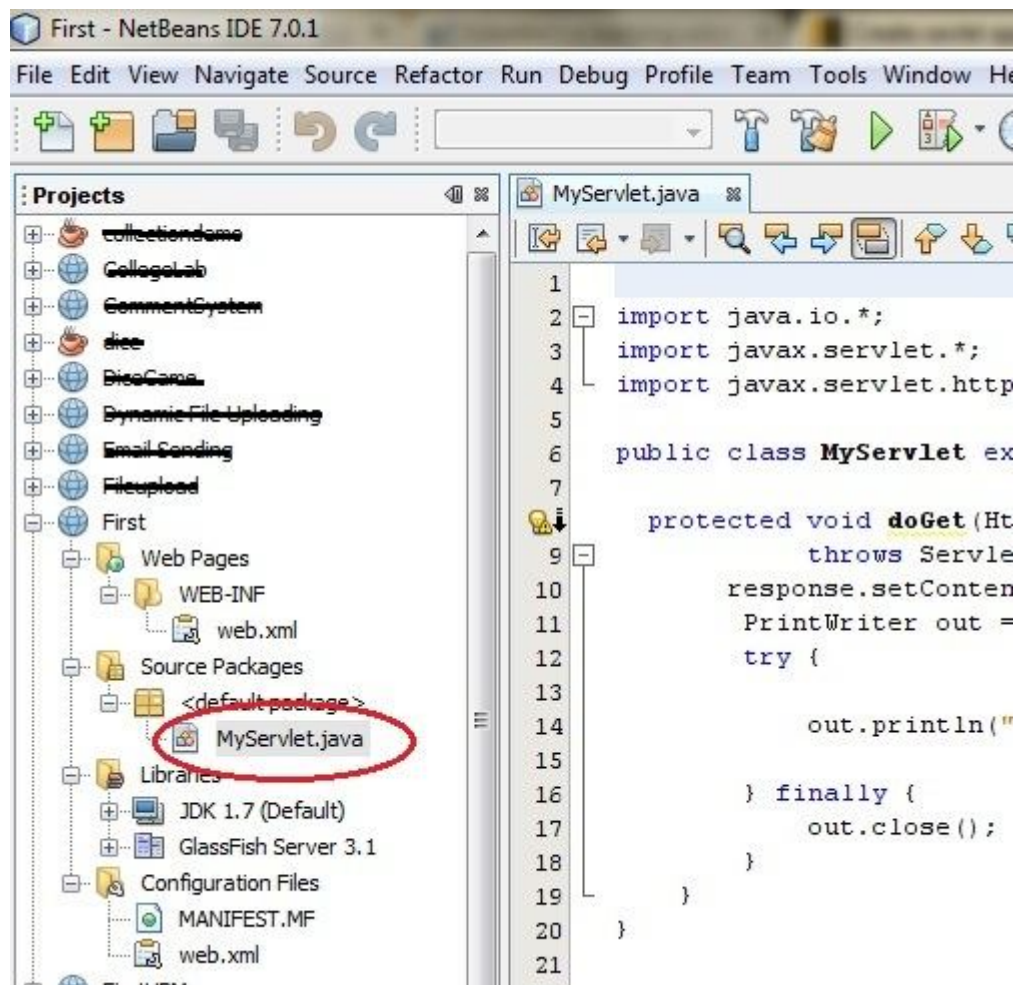
Finish

Cancel

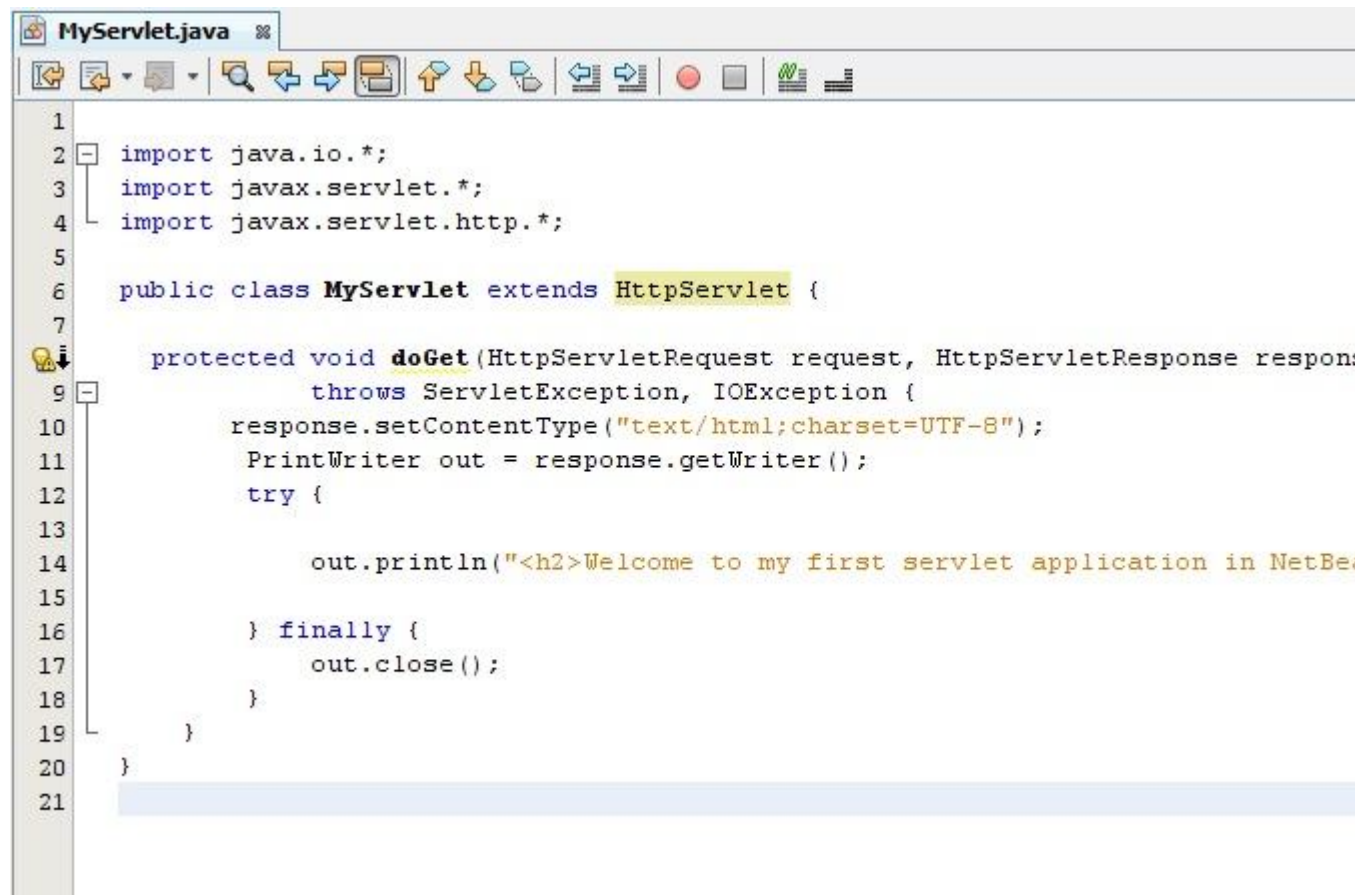
Help



-
8. Now, your Servlet class is ready, and you just need to change the method definitions and you will good to go.

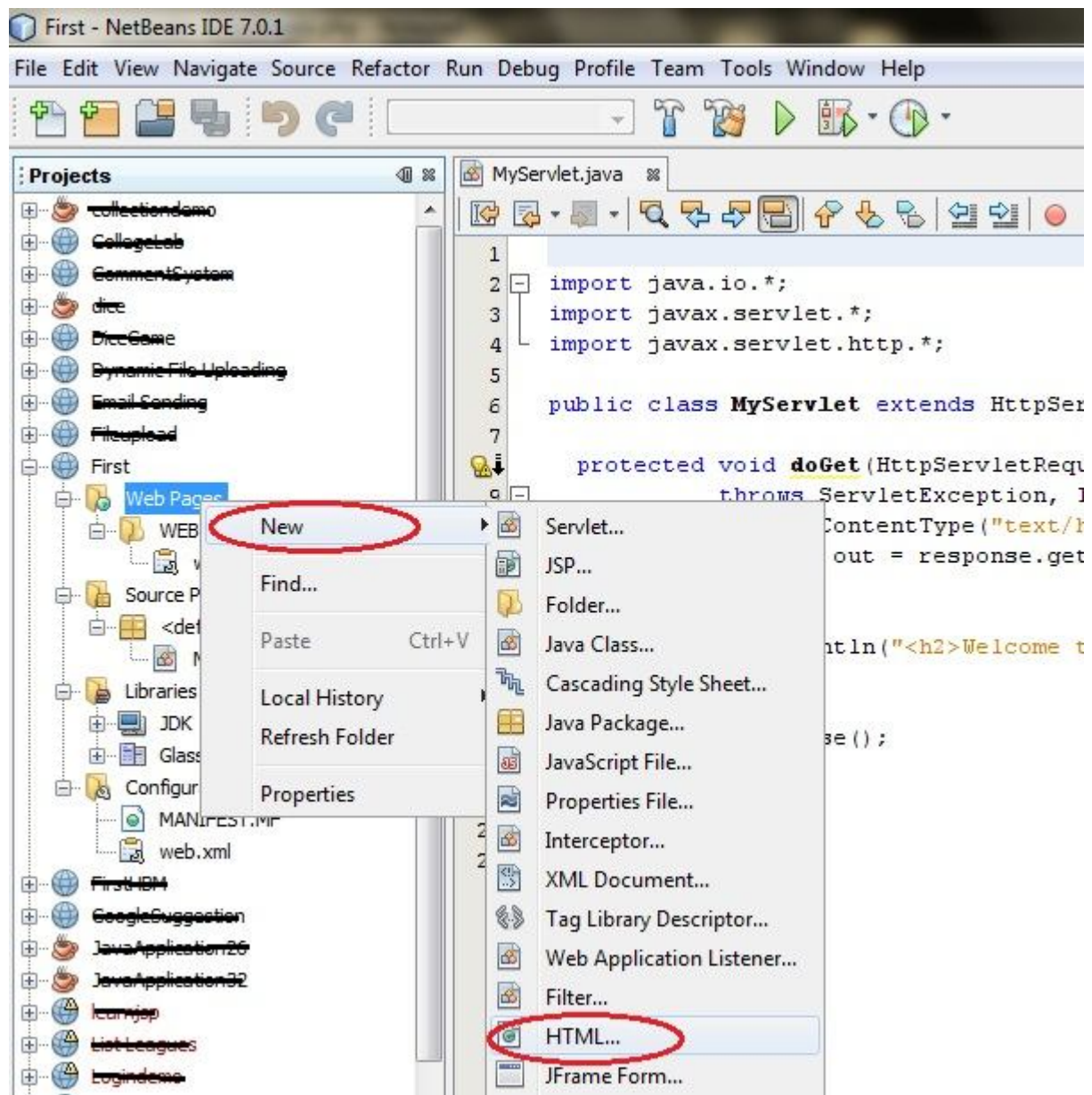


9. Write some code inside your Servlet class.

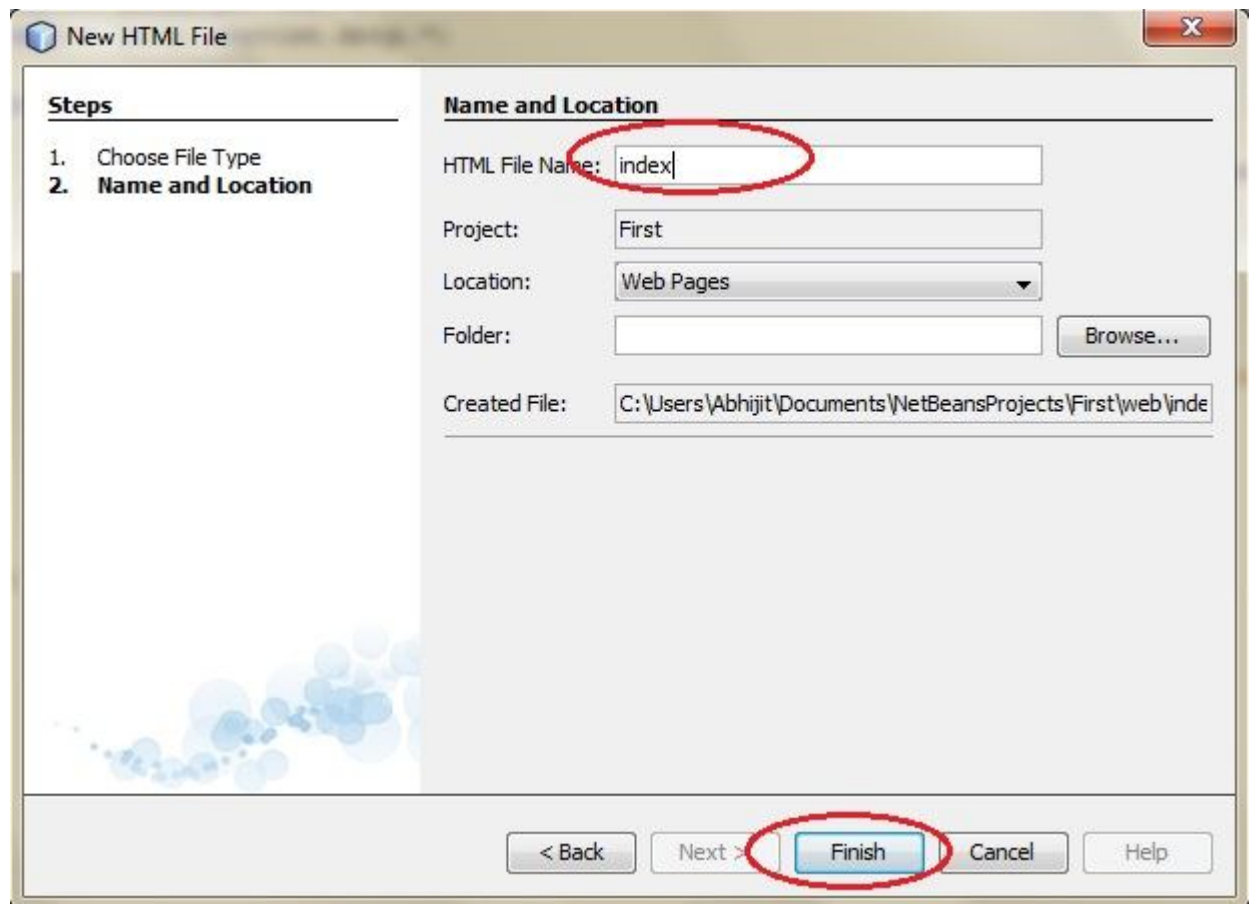


```
1
2 import java.io.*;
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5
6 public class MyServlet extends HttpServlet {
7
8     protected void doGet(HttpServletRequest request, HttpServletResponse response)
9         throws ServletException, IOException {
10         response.setContentType("text/html; charset=UTF-8");
11         PrintWriter out = response.getWriter();
12         try {
13
14             out.println("<h2>Welcome to my first servlet application in NetBeans");
15
16         } finally {
17             out.close();
18         }
19     }
20 }
21
```

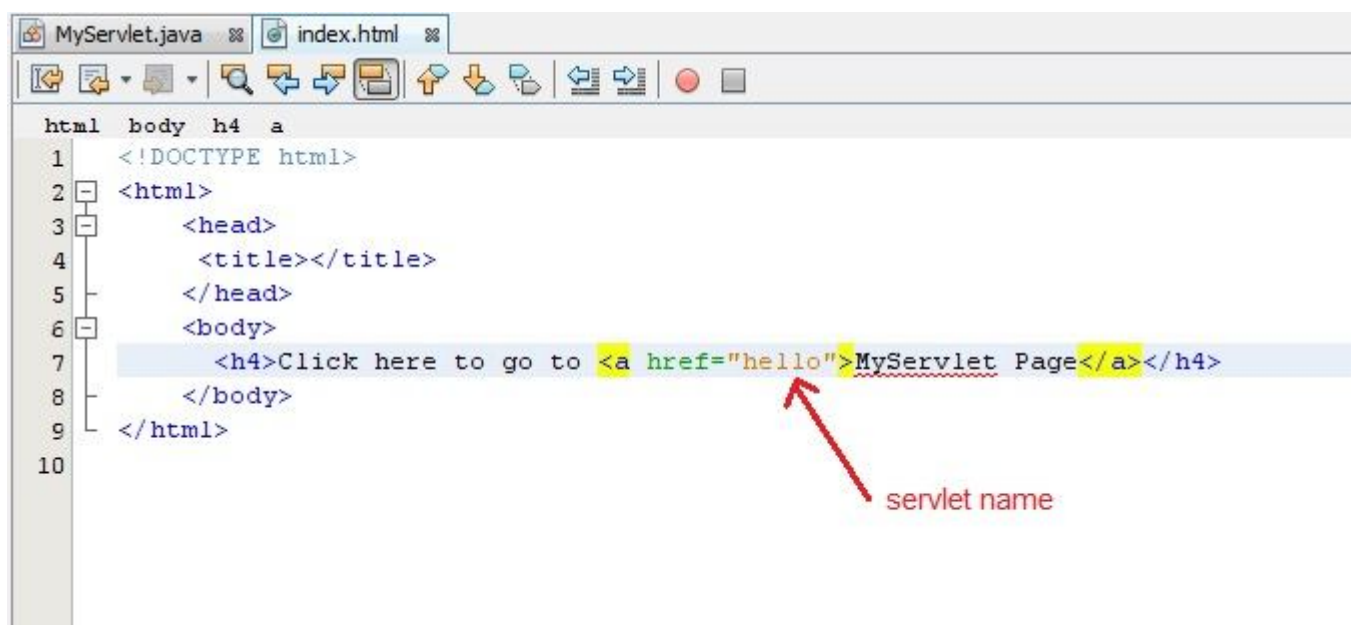
10. Create an HTML file, right click on **Web Pages** -> **New** -> **HTML**



11. Give it a name. We recommend you to name it `index`, because browser will always pick up the `index.html` file automatically from a directory. Index file is read as the first page of the web application.



12. Write some code inside your HTML file. We have created a hyperlink to our Servlet in our HTML file.



Create Deployment Descriptor

Deployment Descriptor(DD) is an XML document that is used by Web Container to run Servlets and JSP pages. DD is used for several important purposes such as:

- Mapping URL to Servlet class.
- Initializing parameters.
- Defining Error page.
- Security roles.
- Declaring tag libraries.

We will discuss about all these in details later. Now we will see how to create a simple **web.xml** file for our web application.

```
<?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>hello</servlet-name>
    <servlet-class>MyServlet</servlet-class>
  </servlet>

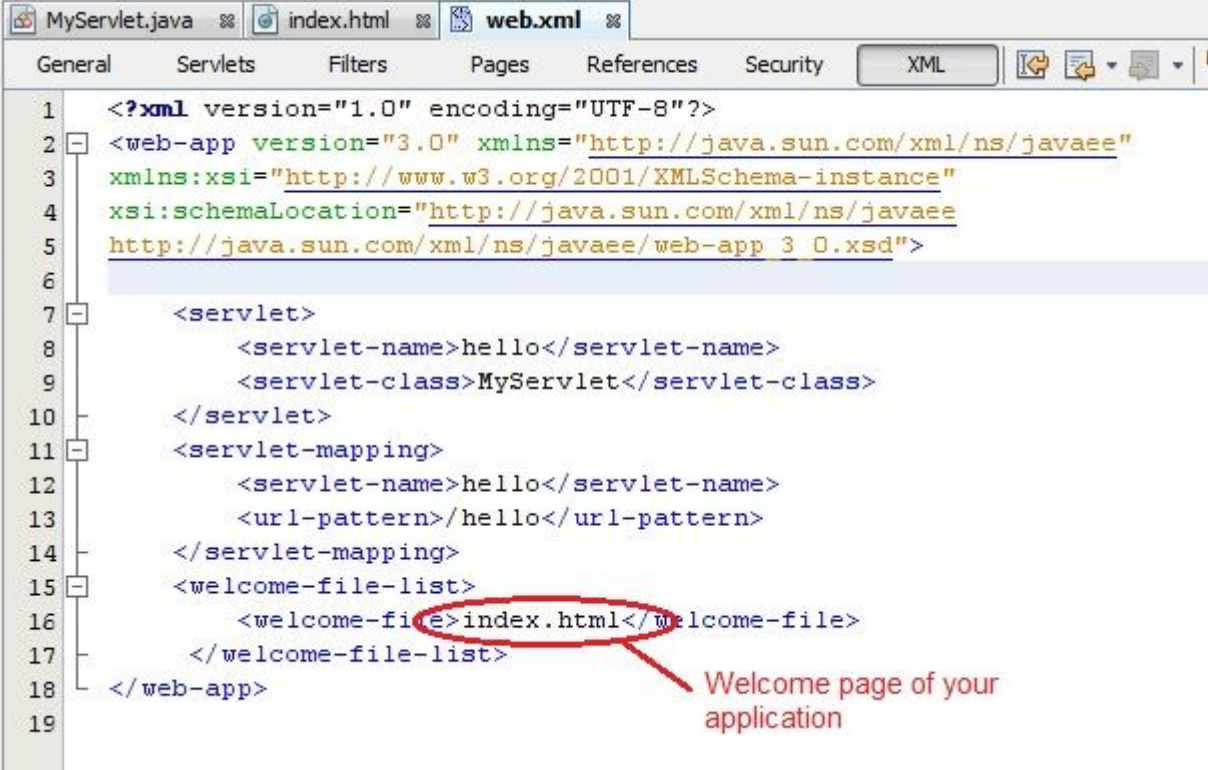
  <servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>

</web-app>
```

Annotations in the diagram:

- First line of any xml document**: points to the XML declaration line.
- root tag of web.xml file. All other tag come inside it**: points to the `<web-app>` tag.
- this tag maps internal name to fully qualified class name**: points to the `<servlet>` tag.
- Give a internal name to your servlet**: points to the `<servlet-name>` element.
- servlet class that you have created**: points to the `<servlet-class>` element.
- this tag maps internal name to public URL name**: points to the `<servlet-mapping>` tag.
- URL name. This is what the user will see to get to the servlet.**: points to the `<url-pattern>` element.

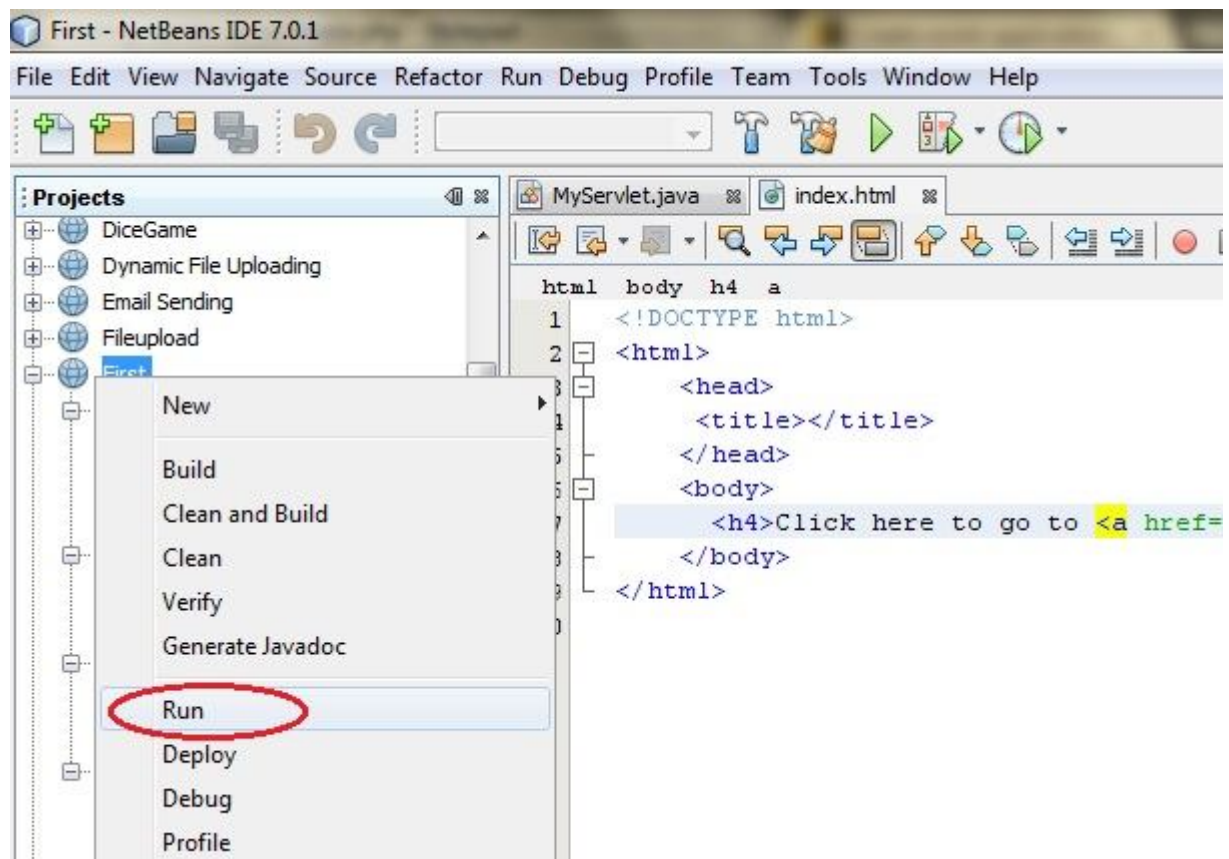
13. Edit **web.xml** file. In the web.xml file you can see, we have specified the **url-pattern** and the **servlet-name**, this means when **hello** url is accessed our Servlet file will be executed.



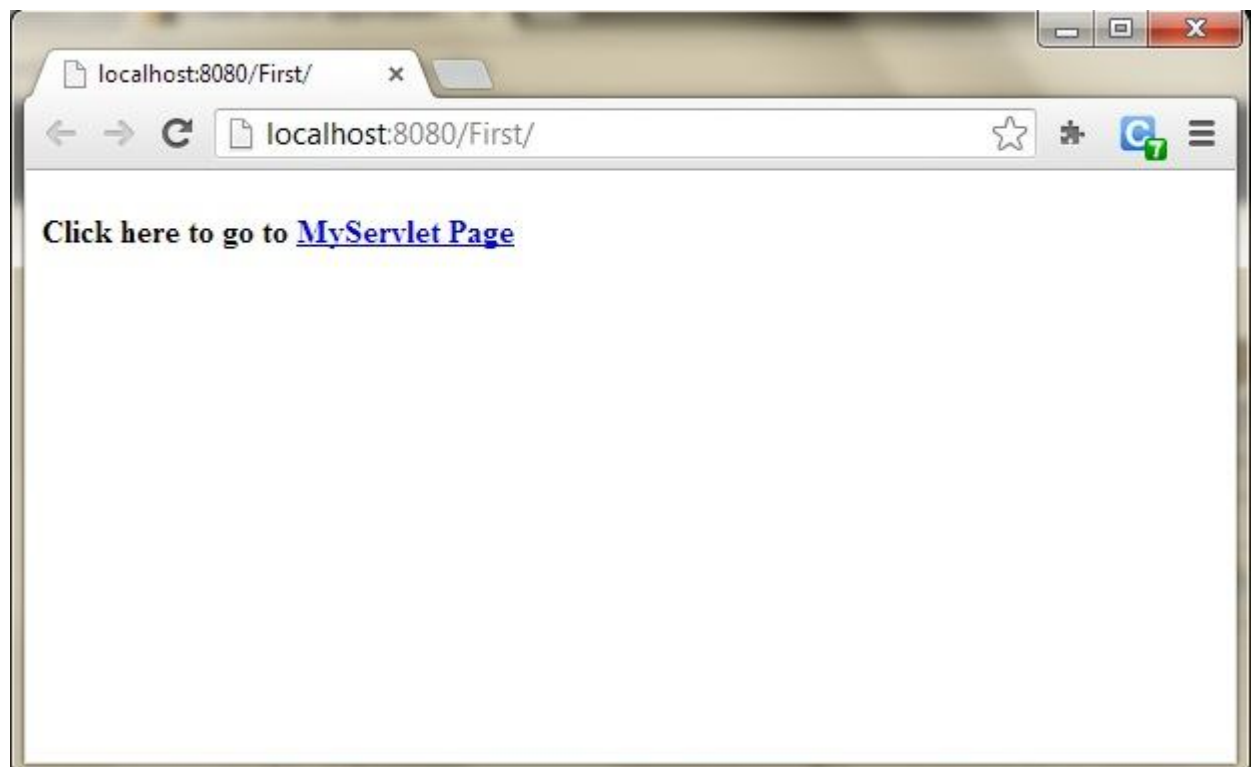
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
3  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
6
7      <servlet>
8          <servlet-name>hello</servlet-name>
9          <servlet-class>MyServlet</servlet-class>
10     </servlet>
11     <servlet-mapping>
12         <servlet-name>hello</servlet-name>
13         <url-pattern>/hello</url-pattern>
14     </servlet-mapping>
15     <welcome-file-list>
16         <welcome-file>index.html</welcome-file>
17     </welcome-file-list>
18 </web-app>
19
```

Welcome page of your application

14. Run your application, right click on your Project and select **Run**



15. Click on the link created, to open your Servlet.



Introduction to Servlet Request

True job of a Servlet is to handle client request. Servlet API provides two important interfaces **javax.servlet.ServletRequest** and **javax.servlet.http.HttpServletRequest** to encapsulate client request. Implementation of these interfaces provide important information about client request to a servlet.

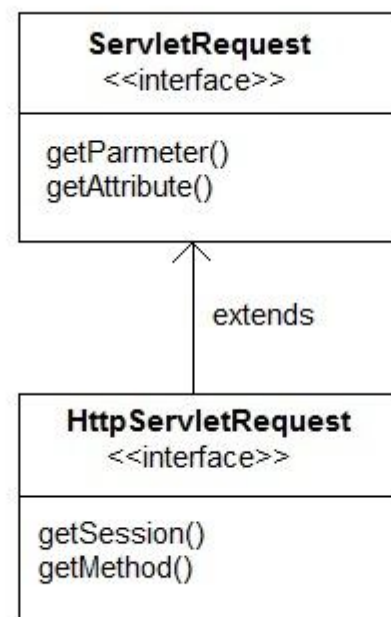
Some Important Methods of ServletRequest

Methods	Description
Object <code>getAttribute(String name)</code>	return attribute set on request object by name
Enumeration <code>getAttributeNames()</code>	return an Enumeration containing the names of the attributes available in this request
int <code>getContentTypeLength()</code>	return size of request body
int <code>getContentType()</code>	return media type of request content
ServletInputStream <code>getInputStream()</code>	returns a input stream for reading binary data
String <code>getParameter(String name)</code>	returns value of parameter by name
String <code>getLocalAddr()</code>	returns the Internet Protocol(IP) address of the interface on which the request was received
Enumeration <code>getParameterNames()</code>	returns an enumeration of all parameter names
String[] <code>getParameterValues(String name)</code>	returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist

ServletContext <code>getServletContext()</code>	return the servlet context of current request.
String <code>getServerName()</code>	returns the host name of the server to which the request was sent
int <code>getServerPort()</code>	returns the port number to which the request was sent
boolean <code>isSecure()</code>	returns a boolean indicating whether this request was made using a secure channel, such as HTTPS.
void <code>removeAttribute(String name)</code>	removes an attribute from this request
void <code>setAttribute(String name, Object o)</code>	stores an attribute in this request.

HttpServletRequest interface

HttpServletRequest interface adds the methods that relates to the **HTTP** protocol.



Some important methods of HttpServletRequest

Methods	Description
String <code>getContextPath()</code>	returns the portion of the request URI that indicates the context of the request
Cookies <code>getCookies()</code>	returns an array containing all of the Cookie objects the client sent with this request
String <code>getQueryString()</code>	returns the query string that is contained in the request URL after the path
HttpSession <code>getSession()</code>	returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session
String <code>getMethod()</code>	Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
Part <code>getPart(String name)</code>	gets the Part with the given name
String <code>getPathInfo()</code>	returns any extra path information associated with the URL the client sent when it made this request.
String <code>getServletPath()</code>	returns the part of this request's URL that calls the servlet

Example demonstrating Servlet Request

In this example, we will show how a parameter is passed to a Servlet in a request object from HTML page.

index.html

```
<form method="post" action="check">
Name <input type="text" name="user" >
<input type="submit" value="submit">
```

```
</form>
```

web.xml

```
<servlet>
    <servlet-name>check</servlet-name>
    <servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>check</servlet-name>
    <url-pattern>/check</url-pattern>
</servlet-mapping>
```

MyServlet.java

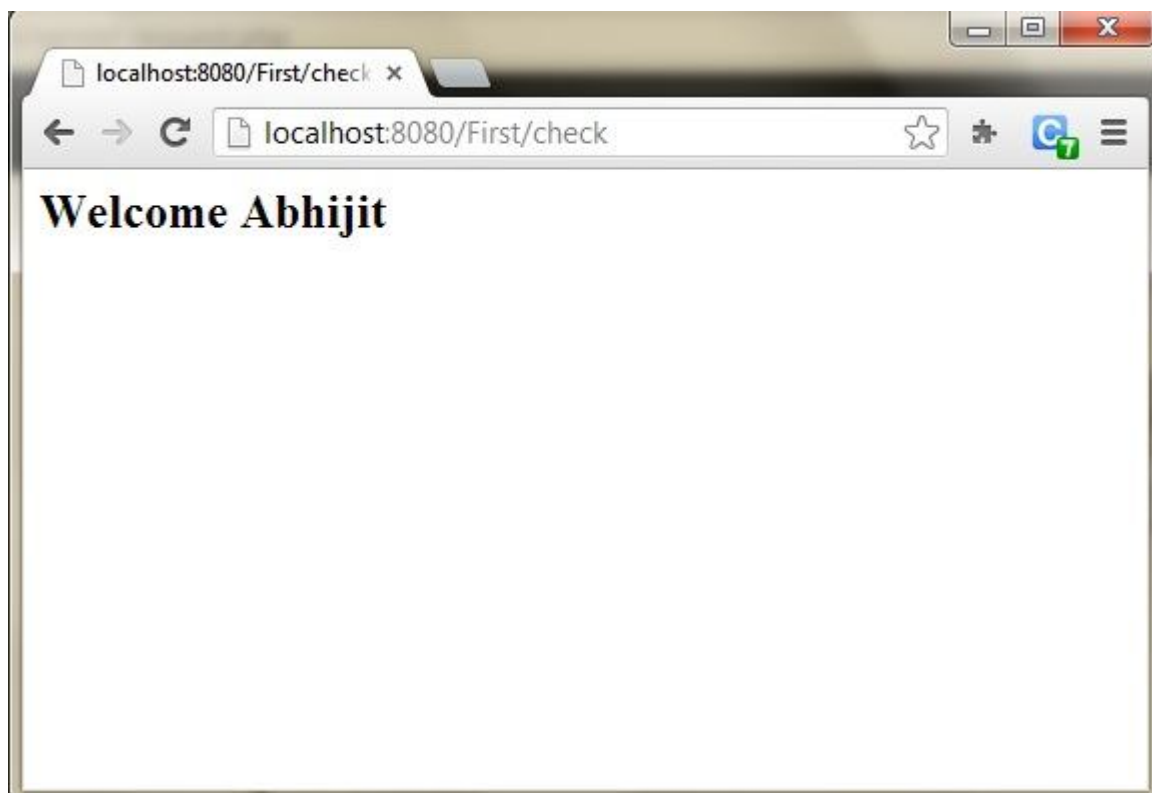
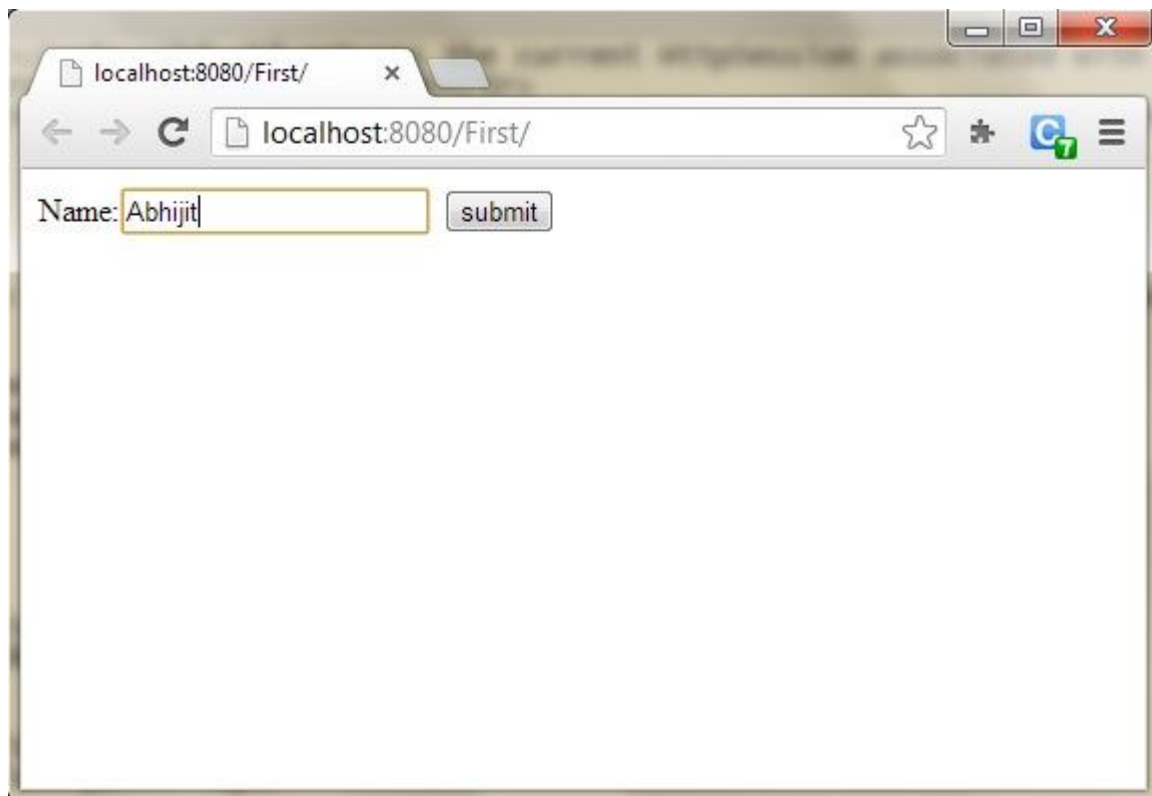
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {

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

            String user=request.getParameter("user");
            out.println("<h2> Welcome "+user+"</h2>");
        } finally {
            out.close();
        }
    }
}
```

Output :



Introduction to Servlet Response

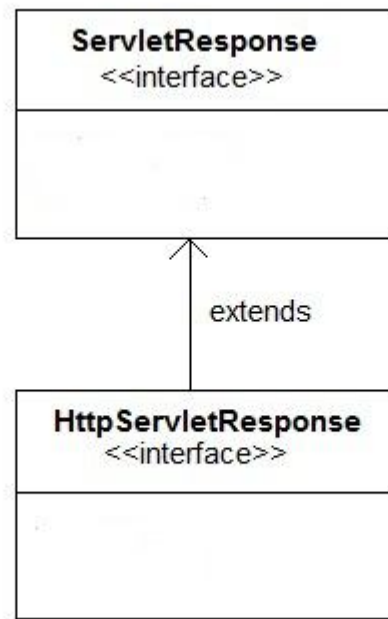
Servlet API provides two important interfaces **ServletResponse** and **HttpServletResponse** to assist in sending response to client.

Some Important Methods of ServletResponse

Methods	Description
PrintWriter <code>getWriter()</code>	returns a PrintWriter object that can send character text to the client.
void <code>setBufferSize(int size)</code>	Sets the preferred buffer size for the body of the response
void <code>setContentLength(int len)</code>	Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header
void <code>.setContentType(String type)</code>	sets the content type of the response being sent to the client before sending the respond.
void <code>setBufferSize(int size)</code>	sets the preferred buffer size for the body of the response.
boolean <code>isCommitted()</code>	returns a boolean indicating if the response has been committed
void <code>setLocale(Locale loc)</code>	sets the locale of the response, if the response has not been committed yet.

HttpServletResponse Interface

HttpServletResponse interface adds the methods that relates to the **HTTP** response.



Some Important Methods of HttpServletResponse

Methods	Description
<code>void addCookie(Cookie cookie)</code>	adds the specified cookie to the response.
<code>void sendRedirect(String location)</code>	Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer
<code>int getStatus()</code>	gets the current status code of this response
<code>String getHeader(String name)</code>	gets the value of the response header with the given name.
<code>void setHeader(String name, String value)</code>	sets a response header with the given name and value
<code>void setStatus(int sc)</code>	sets the status code for this response

void <code>sendError(int sc, String msg)</code>	sends an error response to the client using the specified status and clears the buffer
---	--

Introduction to Request Dispatcher

RequestDispatcher is an interface, implementation of which defines an object which can dispatch request to any resources(such as HTML, Image, JSP, Servlet) on the server.

Methods of RequestDispatcher

RequestDispatcher interface provides two important methods

Methods	Description
void <code>forward(ServletRequest request, ServletResponse response)</code>	forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server
void <code>include(ServletRequest request, ServletResponse response)</code>	includes the content of a resource (servlet, JSP page, HTML file) in the response

How to get an Object of RequestDispatcher

`getRequestDispatcher()` method of **ServletRequest** returns the object of **RequestDispatcher**.

```
RequestDispatcher rs = request.getRequestDispatcher("hello.html");
rs.forward(request, response);
```

ServletRequest object **resource name**

```
RequestDispatcher rs = request.getRequestDispatcher("hello.html");  
  
rs.forward(request, response);
```

forward the request and response to "hello.html" page

OR

```
RequestDispatcher rs = request.getRequestDispatcher("hello.html");  
rs.include(request, response);
```

ServletRequest object **Resource name**

```
RequestDispatcher rs = request.getRequestDispatcher("first.html");  
  
rs.include(request, response);
```

include the response of "first.html" page in current servlet response

Example demonstrating usage of RequestDispatcher

In this example, we will show you how RequestDispatcher is used to **forward** or **include** response of a resource in a Servlet. Here we are using **index.html** to get username and password from the user, **Validate** Servlet will validate the password entered by the user, if the user has entered "studytonight" as password, then he will be forwarded to **Welcome** Servlet else the user will stay on the index.html page and an error message will be displayed.

Files to be created :

- **index.html** will have form fields to get user information.
- **Validate.java** will validate the data entered by the user.

- **Welcome.java** will be the welcome page.
- **web.xml** , the deployment descriptor.

index.html

```
<form method="post" action="Validate">
Name: <input type="text" name="user" /><br/>
Password: <input type="password" name="pass" /><br/>
<input type="submit" value="submit">
</form>
```

Validate.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Validate extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            String name = request.getParameter("user");
            String password = request.getParameter("pass");

            if(password.equals("studytonight"))
            {
                RequestDispatcher rd = request.getRequestDispatcher("Welcome");
                rd.forward(request, response);
            }
            else
            {
                out.println("<font color='red'><b>You have entered incorrect password</b></font>");
                RequestDispatcher rd = request.getRequestDispatcher("index.html");
                rd.include(request, response);
            }
        } finally {
            out.close();
        }
    }
}
```

```
}  
}
```

Welcome.java

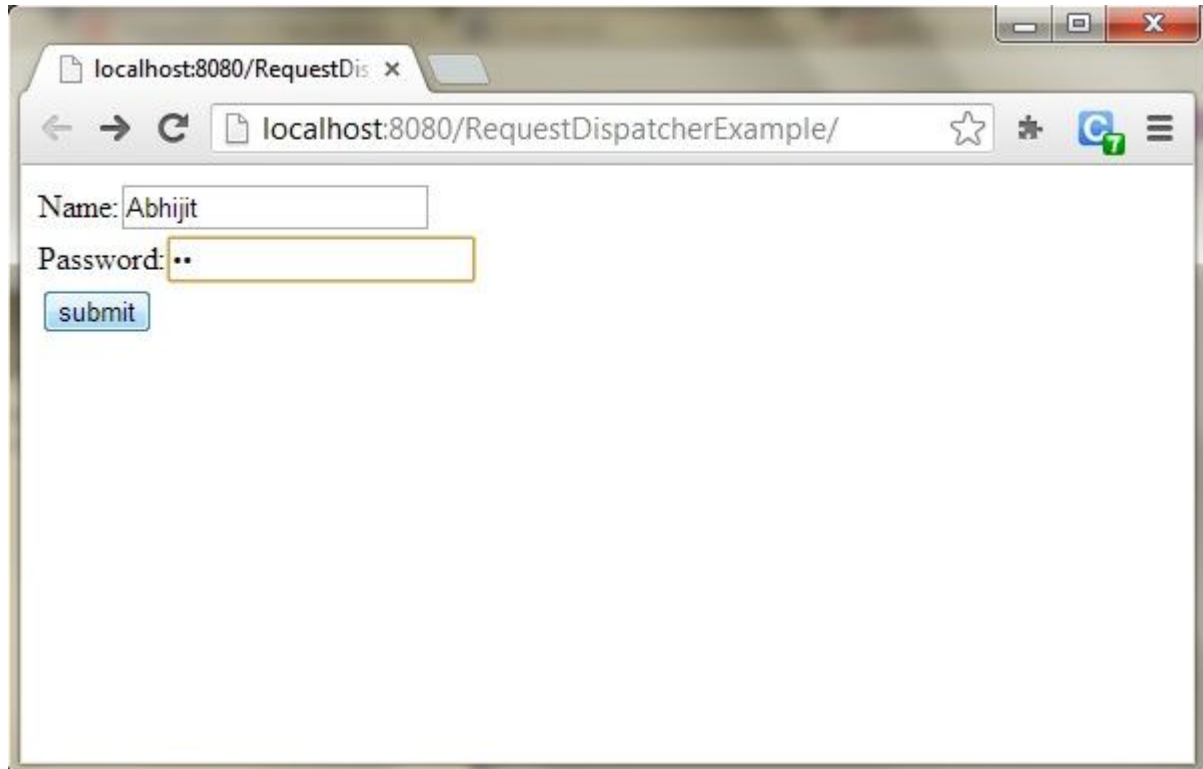
```
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
public class Welcome extends HttpServlet {  
  
    protected void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html; charset=UTF-8");  
        PrintWriter out = response.getWriter();  
        try {  
  
            out.println("<h2>Welcome user</h2>");  
        } finally {  
            out.close();  
        }  
    }  
}
```

web.xml

```
<web-app>  
    <servlet>  
        <servlet-name>Validate</servlet-name>  
        <servlet-class>Validate</servlet-class>  
    </servlet>  
    <servlet>  
        <servlet-name>Welcome</servlet-name>  
        <servlet-class>Welcome</servlet-class>  
    </servlet>  
    <servlet-mapping>  
        <servlet-name>Validate</servlet-name>  
        <url-pattern>/Validate</url-pattern>  
    </servlet-mapping>  
    <servlet-mapping>  
        <servlet-name>Welcome</servlet-name>  
        <url-pattern>/Welcome</url-pattern>  
    </servlet-mapping>  
    <welcome-file-list>  
        <welcome-file>index.html</welcome-file>
```

```
</welcome-file-list>  
</web-app>
```

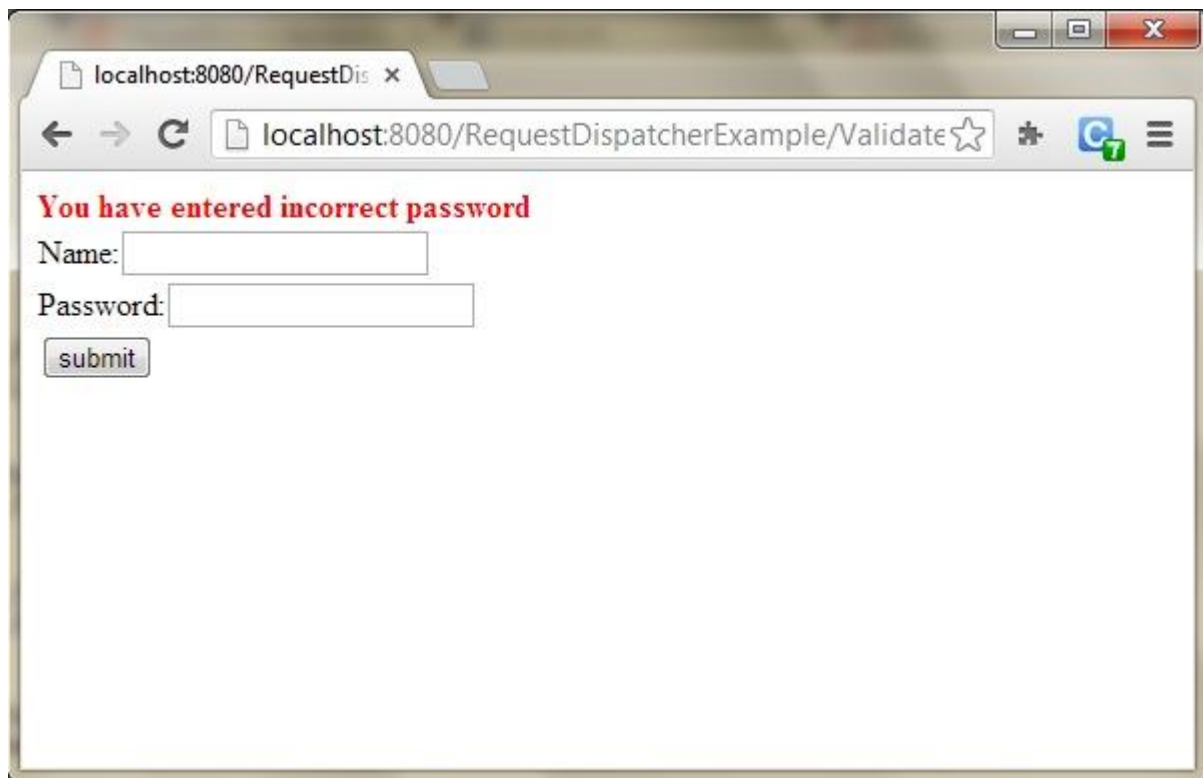
This will be the first screen. You can enter your Username and Password here.



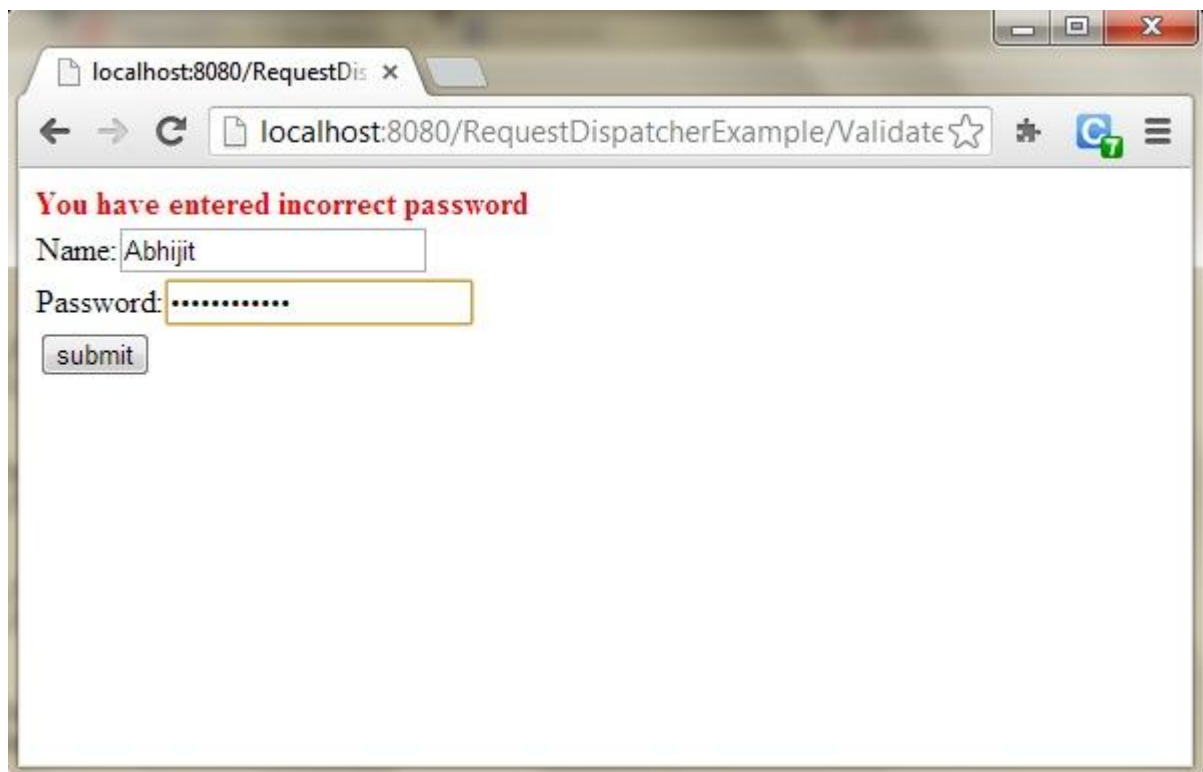
The screenshot shows a web browser window with the address bar displaying 'localhost:8080/RequestDispatcherExample/'. The page contains a login form with the following elements:

- A text input field labeled 'Name:' containing the text 'Abhijit'.
- A text input field labeled 'Password:' containing two dots '••'.
- A blue button labeled 'submit'.

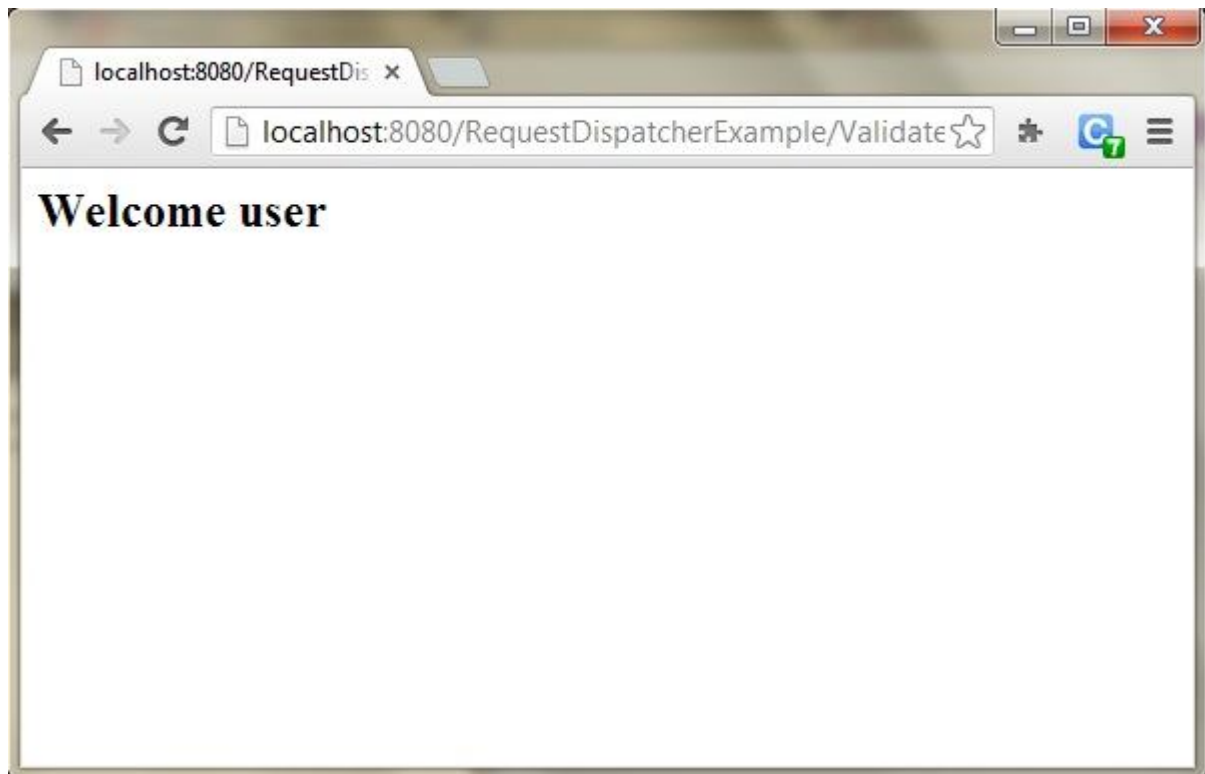
When you click on Submit, Password will be validated, if it is not 'studytonight' , error message will be displayed.



Enter any Username, but enter 'studytonight' as password.



Password will be successfully validated and you will be directed to the Welcome Servlet.



Introduction to sendRedirect() Method

`sendRedirect()` method redirects the response to another resource. This method actually makes the client(browser) to create a new request to get to the resource. The client can see the new url in the browser.

`sendRedirect()` accepts relative **URL**, so it can go for resources inside or outside the server.

sendRedirect() and Request Dispatcher

The main difference between a **redirection** and a **request dispatching** is that, redirection makes the client(browser) create a new request to get to the resource, the user can see the new URL while request dispatch get the resource in same request and URL does not changes.

Also, another very important difference is that, `sendRedirect()` works on **response** object while request dispatch work on **request** object.

Example demonstrating usage of sendRedirect()

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class MyServlet extends HttpServlet {  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html; charset=UTF-8");  
        PrintWriter out = response.getWriter();  
        try {  
            response.sendRedirect("http: //www. studytonight. com");  
        } finally {  
            out.close();  
        }  
    }  
}
```

Introduction to ServletConfig interface

When the **Web Container** initializes a servlet, it creates a **ServletConfig** object for the servlet. ServletConfig object is used to pass information to a servlet during initialization by getting configuration information from **web.xml**(Deployment Descriptor).

Methods of ServletConfig

- String *getInitParameter(String name)*: returns a String value initialized parameter, or NULL if the parameter does not exist.
 - Enumeration *getInitParameterNames()*: returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.
 - ServletContext *getServletContext()*: returns a reference to the ServletContext
 - String *getServletName()*: returns the name of the servlet instance
-

How to Initialize a Servlet inside web.xml

In the Deployment Descriptor(web.xml) file,


```

<servlet>
  <servlet-name>check</servlet-name>
  <servlet-class>MyServlet</servlet-class>
  <init-param>
    <param-name>email</param-name>
    <param-value>we@studytonight.com</param-value>
  </init-param>
</servlet>

```

Or, Inside the Servlet class, using following code,

```

ServletConfig sc = getServletConfig();
out.println(sc.getInitParameter("email"));

```

Example demonstrating usage of ServletConfig

web.xml

```

<web-app ... >
  <servlet>
    <servlet-name>check</servlet-name>
    <servlet-class>MyServlet</servlet-class>
    <init-param>
      <param-name>email</param-name>
      <param-value>we@studytonight.com</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>check</servlet-name>
    <url-pattern>/check</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

```

```
</welcome-file-list>
</web-app>
```

MyServlet class :

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        ServletConfig sc=getServletConfig();
        out.println(sc.getInitParameter("email"));
    }
}
```

Introduction to ServletContext Interface

For every **Web application** a **ServletContext** object is created by the web container. ServletContext object is used to get configuration information from **Deployment Descriptor**(web.xml) which will be available to any servlet or JSPs that are part of the web app.

Some Important method of ServletContext

Methods	Description
Object <code>getAttribute(String name)</code>	returns the container attribute with the given name, or NULL if there is no attribute by that name.
String <code>getInitParameter(String name)</code>	returns parameter value for the specified parameter name, or NULL if the parameter does not exist
Enumeration <code>getInitParameterNames()</code>	returns the names of the context's initialization parameters as an Enumeration of String objects

void <code>setAttribute(String name, Object obj)</code>	set an object with the given attribute name in the application scope
void <code>removeAttribute(String name)</code>	removes the attribute with the specified name from the application context

How Context Parameter is Initialized inside web.xml

```

<web-app ...>
  <context-param>
    <param-name>driverName</param-name>
    <param-value>sun.jdbc.JdbcOdbcDriver</param-value>
  </context-param>
  <servlet>
    ....
  </servlet>
</web-app>

```

The `<context-param>` is for whole application, so it is put inside the `<web-app>` tag but outside any `<servlet>` declaration

Parameter name

Parameter value

How to get the Object of ServletContext

```

ServletContext app = getServletContext();
OR
ServletContext app = getServletConfig().getServletContext();

```

Advantages of ServletContext

- Provides communication between servlets
 - Available to all servlets and JSPs that are part of the web app
 - Used to get configuration information from web.xml
-

Difference between Context Init Parameters and Servlet Init Parameter

Context Init parameters	Servlet Init parameter
Available to all servlets and JSPs that are part of web	Available to only servlet for which the <code><init-param></code> was configured
Context Init parameters are initialized within the <code><web-app></code> not within a specific <code><servlet></code> elements	Initialized within the <code><servlet></code> for each specific servlet.
ServletContext object is used to get Context Init parameters	ServletConfig object is used to get Servlet Init parameters
Only one ServletContext object for entire web app	Each servlet has its own ServletConfig object

Example demonstrating usage of ServletContext

web.xml

```
<web-app ...>

    <context-param>
        <param-name>driverName</param-name>
        <param-value>sun.jdbc.JdbcOdbcDriver</param-value>
    </context-param>

    <servlet>
        <servlet-name>hello</servlet-name>
        <servlet-class>MyServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
```

```
</web-app>
```

MyServlet class :

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        ServletContext sc = getServletContext();
        out.println(sc.getInitParameter("driverName"));
    }
}
```

Introduction to Attribute

An **attribute** is an object that is used to share information in a web app. Attribute allows Servlets to share information among themselves. Attributes can be SET and GET from one of the following scopes :

1. request
2. session
3. application

Application Scope:

```
ServletContext sc=getServletContext();
sc.setAttribute("user","Abhijit");
sc.getAttribute("user");
sc.removeAttribute("user");
```

Diagram annotations for Application Scope:

- `getServletContext()`: context object
- `"user"` in `setAttribute`: attribute name
- `"Abhijit"`: attribute value
- `getAttribute("user")`: getting an attribute
- `removeAttribute("user")`: removing attribute

request Scope:

```
request.setAttribute("user","Abhijit");
request.getAttribute("user");
request.removeAttribute("user");
```

Diagram annotations for request Scope:

- `request` (circled in all three lines): ServletRequest object
- `setAttribute("user","Abhijit")`: setting an attribute on request scope
- `getAttribute("user")`: getting an attribute
- `removeAttribute("user")`: removing an attribute

How to SET an Attribute

`public void setAttribute(String name, Object obj)` method is used to SET an Attribute.

Example demonstrating Setting Attribute

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class First extends HttpServlet {

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

```
ServletContext sc = getServletContext();  
sc.setAttribute("user", "Abhijit"); //setting attribute on context scope  
}  
}
```

How to GET an Attribute

Object `getAttribute(String name)` method is used to GET an attribute.

Example demonstrating getting a value of set Attribute

```
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
  
public class Second extends HttpServlet {  
  
    protected void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html; charset=UTF-8");  
        PrintWriter out = response.getWriter();  
        ServletContext sc = getServletContext();  
  
        String str = sc.getAttribute("user"); //getting attribute from context scope  
  
        out.println("Welcome"+str); // Prints : Welcome Abhijit  
    }  
}
```

16. Hurray! Our First Servlet class is running.

