Scripts are often utilized to create dynamic web applications nowadays because they are linked to web development. Server-Side Scripting Languages and Client-Side Scripting Languages are the two types of scripting languages. Python, PHP, and Perl are examples of server-side scripting languages, while JavaScript is the greatest example of a client-side scripting language.

Servlets

Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).

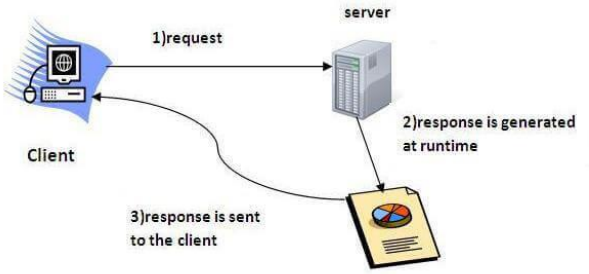
Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology.

There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.

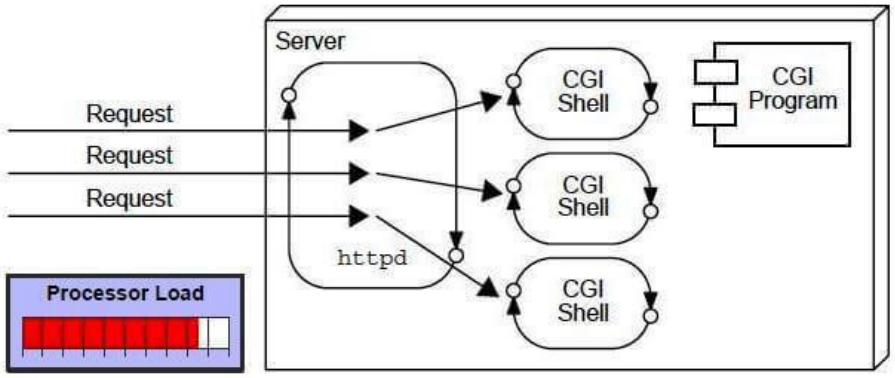


What is a web application?

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

CGI (Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



Disadvantages of CGI

There are many problems in CGI technology:

1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.

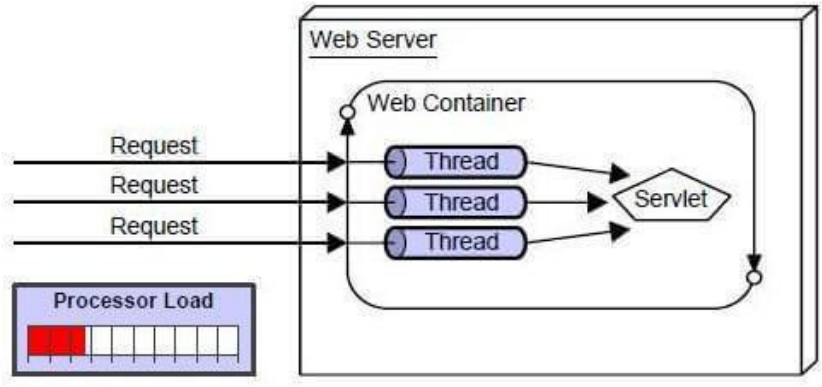
3. It uses platform dependent language e.g. C, C++, perl

Advantages of Servlet

There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. Web Container is used in java for dynamically generating the web pages on the server side.

The advantages of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.



Servlet API

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Interfaces in javax.servlet package

There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletRequestWrapper
5. ServletResponseWrapper
6. ServletRequestEvent
7. ServletContextEvent
8. ServletRequestAttributeEvent
9. ServletContextAttributeEvent
10. ServletException
11. UnavailableException

Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener

7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

Servlet Interface

Servlet interface provides common behavior to all the servlets. Servlet interface defines methods that all servlets must implement.

Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

Method	Description
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

GenericServlet class

GenericServlet class implements **Servlet**, **ServletConfig** and **Serializable** interfaces. It provides the implementation of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet class

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.

5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.

Servlet Example by inheriting the GenericServlet class

File: First.java

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

public class First extends GenericServlet
{
   public void service(ServletRequest req,ServletResponse res)  throws IOException,ServletException
   {
            res.setContentType("text/html");
           PrintWriter out=res.getWriter();

          out.print("<html><body>");
          out.print("<b>hello generic servlet</b>");
          out.print("</body></html>");
    }
}
```

HttpServlet class

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

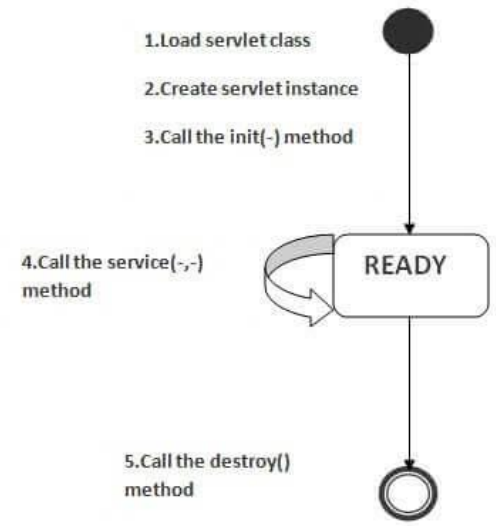
Methods of HttpServlet class

1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.

Life Cycle of a Servlet (Servlet Life Cycle)

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.



As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

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

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

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

5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

```
public void destroy()
```

Steps to create a servlet example

There are given 6 steps to create a **servlet example**. These steps are required for all the servers.
The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.
Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

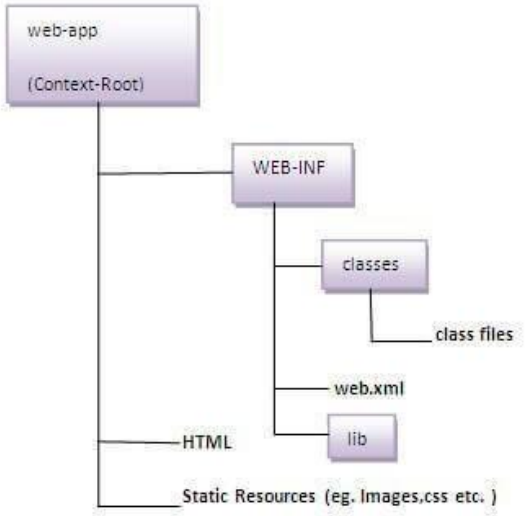
1. Create a directory structure

2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

1) Create a directory structures

The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet. As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.



2)lCreate a Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc. In this example we are going to create a servlet that extends the HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request.

DemoServlet.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException, IOException
{
res.setContentType("text/html");//setting the content type
PrintWriter pw=res.getWriter();//get the stream to write the data
```

```
//writing html in the stream
pw.println("<html><body>");
pw.println("Welcome to servlet");
pw.println("</body></html>");
  pw.close();//closing the stream
}}
```

3)Compile the servlet

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

Jar file	Server
1) servlet-api.jar	Apache Tomcat
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

4)Create the deployment descriptor (web.xml file)

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and

Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

web.xml file

```
<web-app>
  <servlet>
<servlet-name>mrec</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>mrec</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
  </web-app>
```

Description of the elements of web.xml file

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

<web-app> represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

5)Start the Server and deploy the project

To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.

One Time Configuration for Apache Tomcat Server

You need to perform 2 tasks:

1. set JAVA_HOME or JRE_HOME in environment variable (It is required to start server).
2. Change the port number of tomcat (optional). It is required if another server is running on same port (8080).

How Servlet works?

It is important to learn how servlet works for understanding the servlet well. Here, we are going to get the internal detail about the first servlet program.

The server checks if the servlet is requested **for the first time**.

If yes, web container does the following tasks:

a) loads the servlet class, b) instantiates the servlet class, c) calls the init method passing the ServletConfig object

Else : calls the service method passing request and response objects

The web container calls the destroy method when it needs to remove the servlet such as at time of stopping server or undeploying the project.

How web container handles the servlet request?

The web container is responsible to handle the request. Let's see how it handles the request.

- maps the request with the servlet in the web.xml file.
- creates request and response objects for this request
- calls the service method on the thread
- The public service method internally calls the protected service method
- The protected service method calls the doGet method depending on the type of request.
- The doGet method generates the response and it is passed to the client.
- After sending the response, the web container deletes the request and response objects. The thread is contained in the thread pool or deleted depends on the server implementation.

The public service method converts the ServletRequest object into the HttpServletRequest type and ServletResponse object into the HttpServletResponse type. Then, calls the service method passing these objects. The protected service method checks the type of request, if request type is get, it calls doGet method, if request type is post, it calls doPost method, so on.

Creating Servlet Example in Eclipse

Eclipse is an open-source ide for developing JavaSE and JavaEE (J2EE) applications. You can download the eclipse ide from the eclipse website http://www.eclipse.org/downloads/.

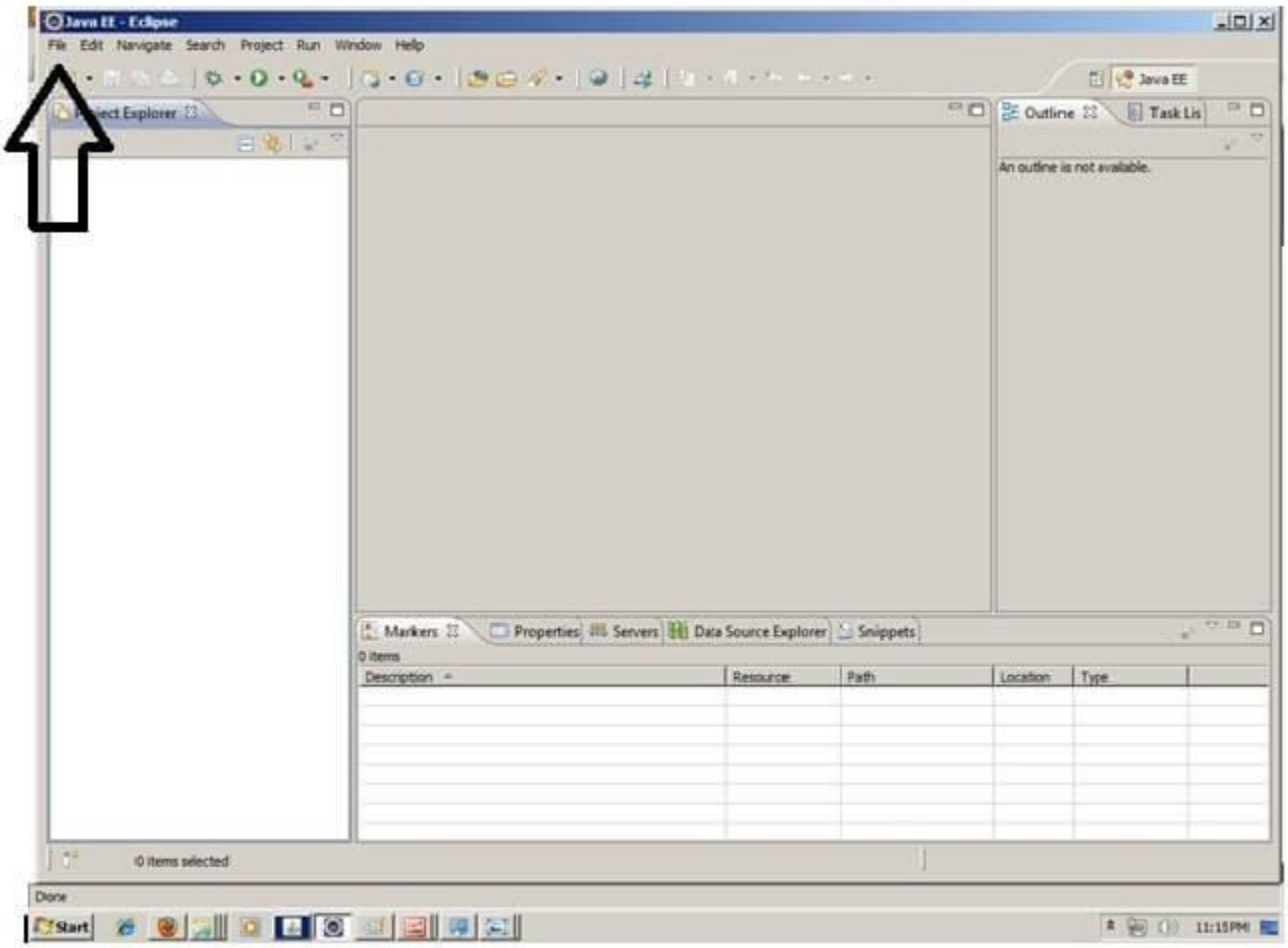
You need to download the eclipse ide for JavaEE developers.

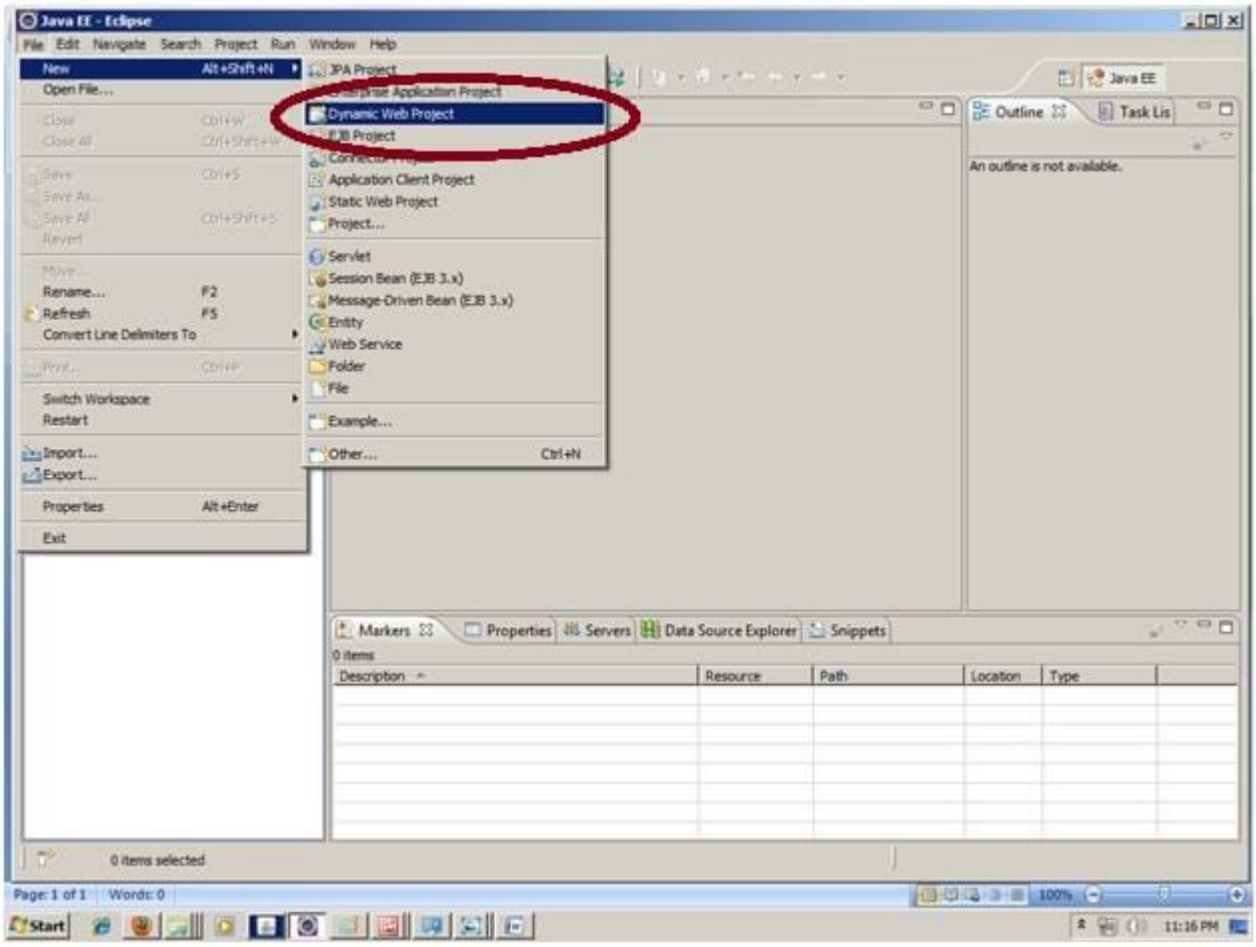
Creating **servlet example in eclipse ide**, saves a lot of work to be done. It is easy and simple to create a servlet example. Let's see the steps, you need to follow to create the first servlet example.

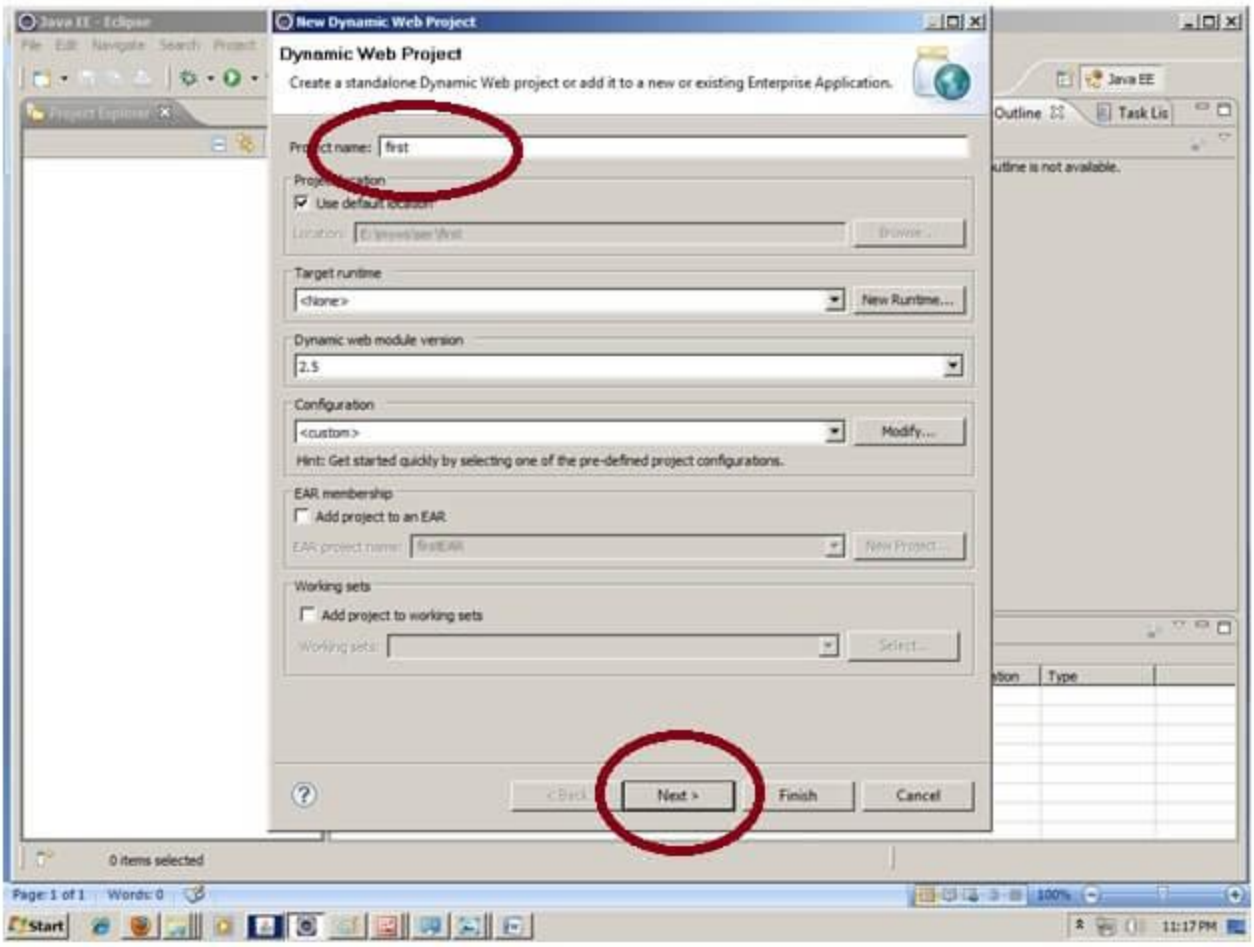
1. Create a Dynamic web project
2. create a servlet
3. add servlet-api.jar file
4. Run the servlet

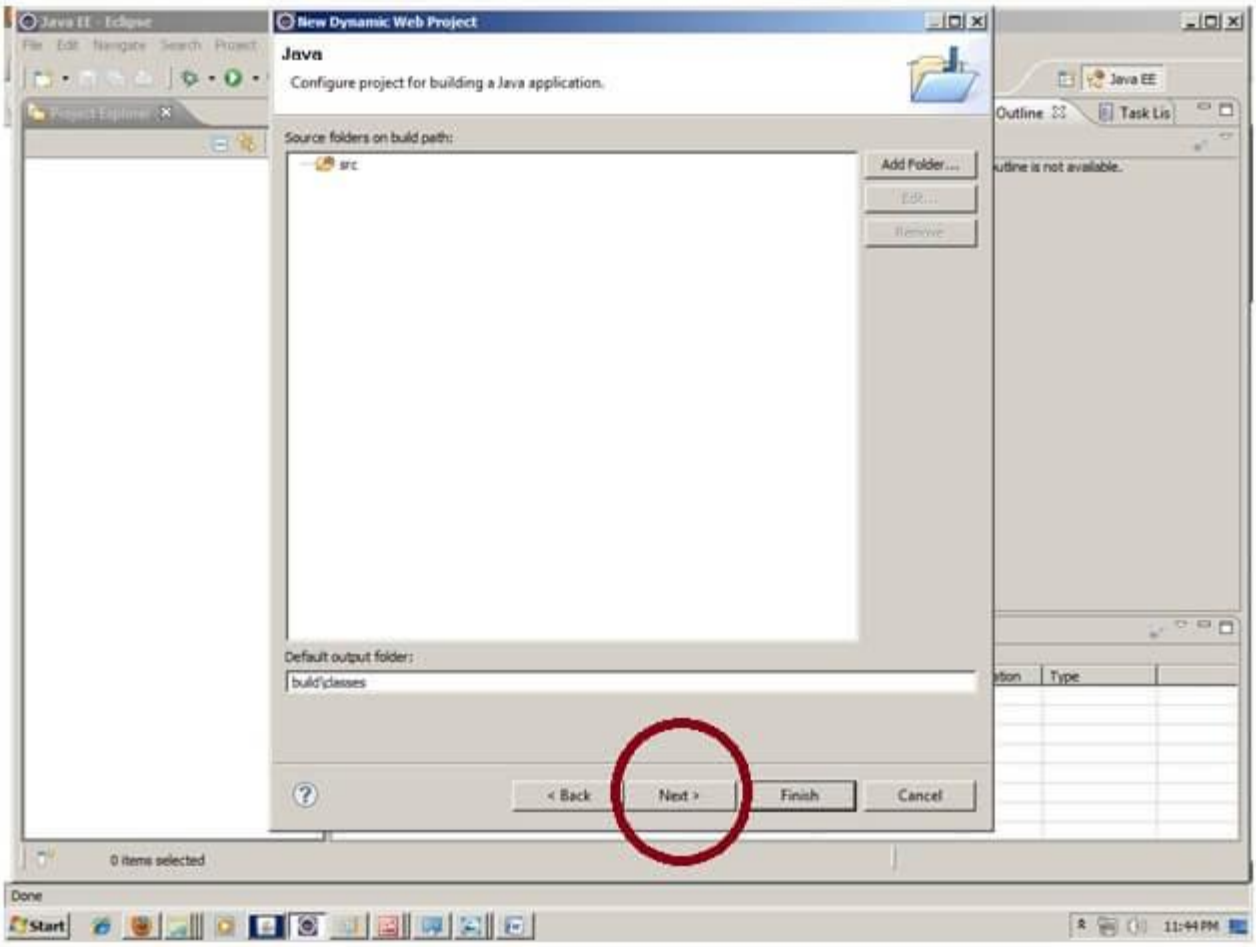
1) Create the dynamic web project:

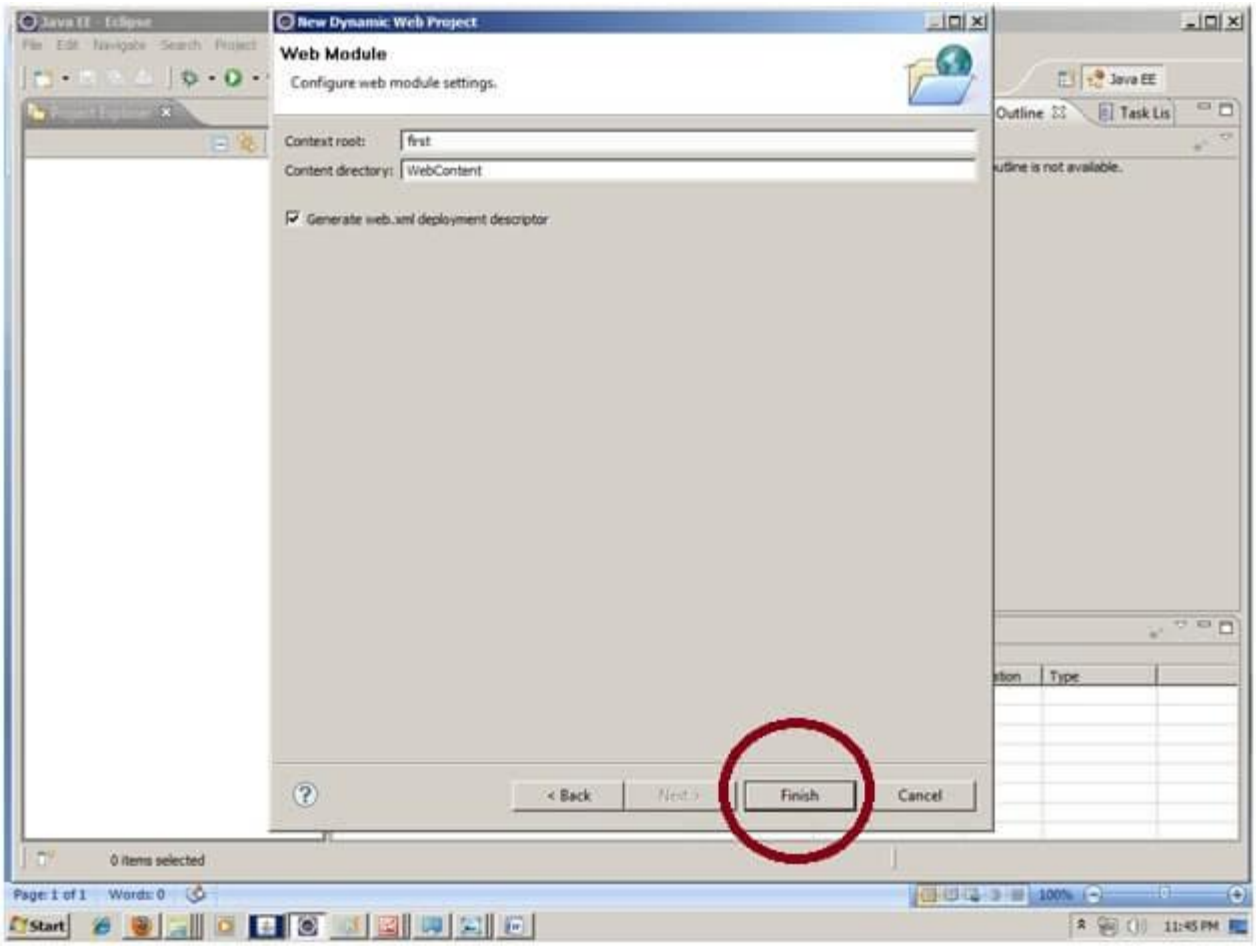
For creating a dynamic web project **click on File Menu -> New -> Project..-> Web -> dynamic web project -> write your project name e.g. first -> Finish**.

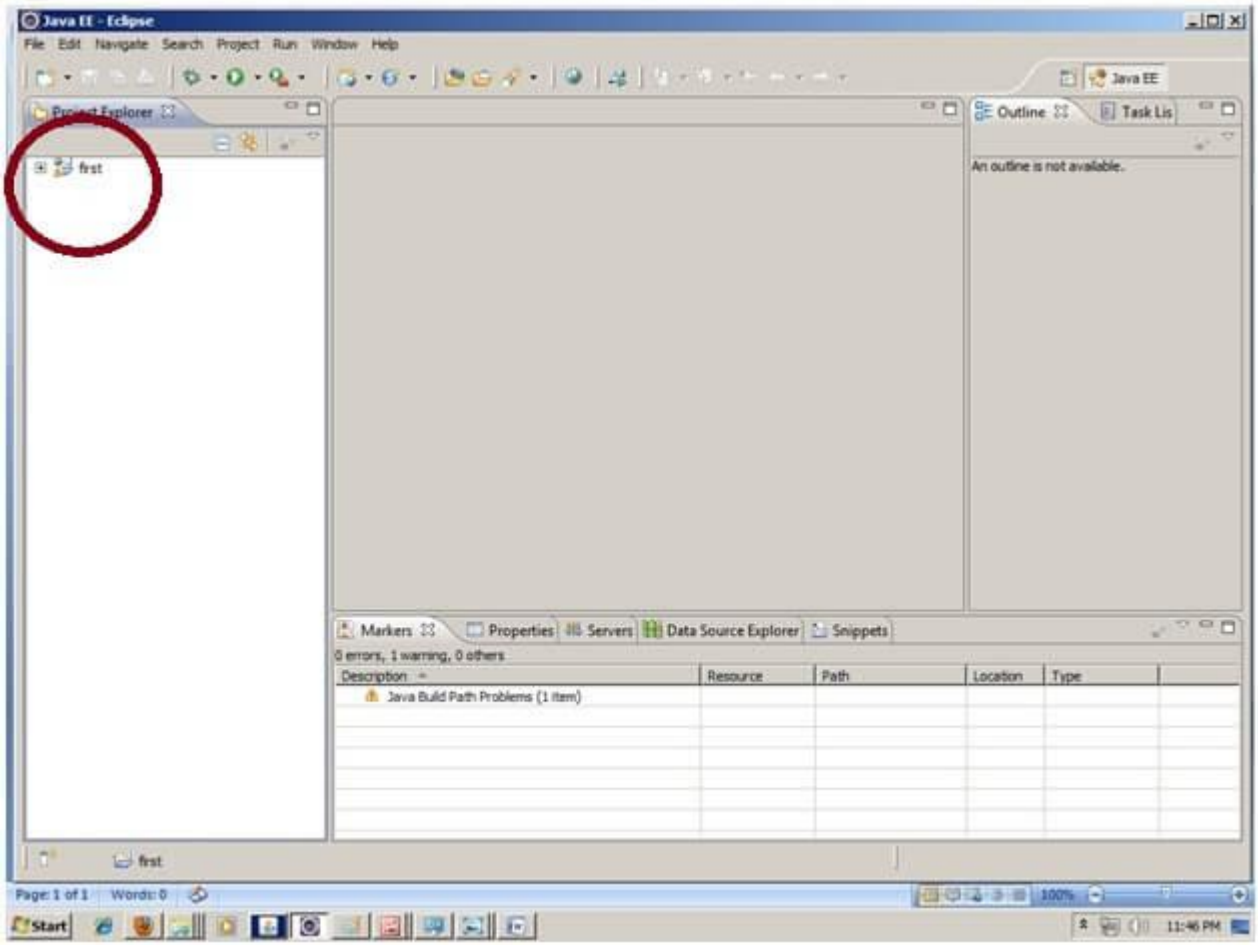



Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
Exit
JPA Project
Enterprise Application Project
Dynamic Web Project
EJB Project
Connector Project
Application Client Project
Static Web Project
Project...
Servlet
Session Bean (EJB 3.x)
Message-Driven Bean (EJB 3.x)
Entity
Web Service
Folder
File
Example...
Other... Ctrl+N
Java EE
Outline
Task List
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
0 items
Description
Resource
Path
Location
Type
0 items selected
Page: 1 of 1
Words: 0
100%
Start
11:16 PM


New Dynamic Web Project
Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.
Project name: first
Use default location
Target runtime
<none>
New Runtime...
Dynamic web module version
2.5
Configuration
<custom>
Modify...
Hint: Get started quickly by selecting one of the pre-defined project configurations.
EAR membership
Add project to an EAR
Working sets
Add project to working sets
Next >
Finish
Cancel
0 items selected

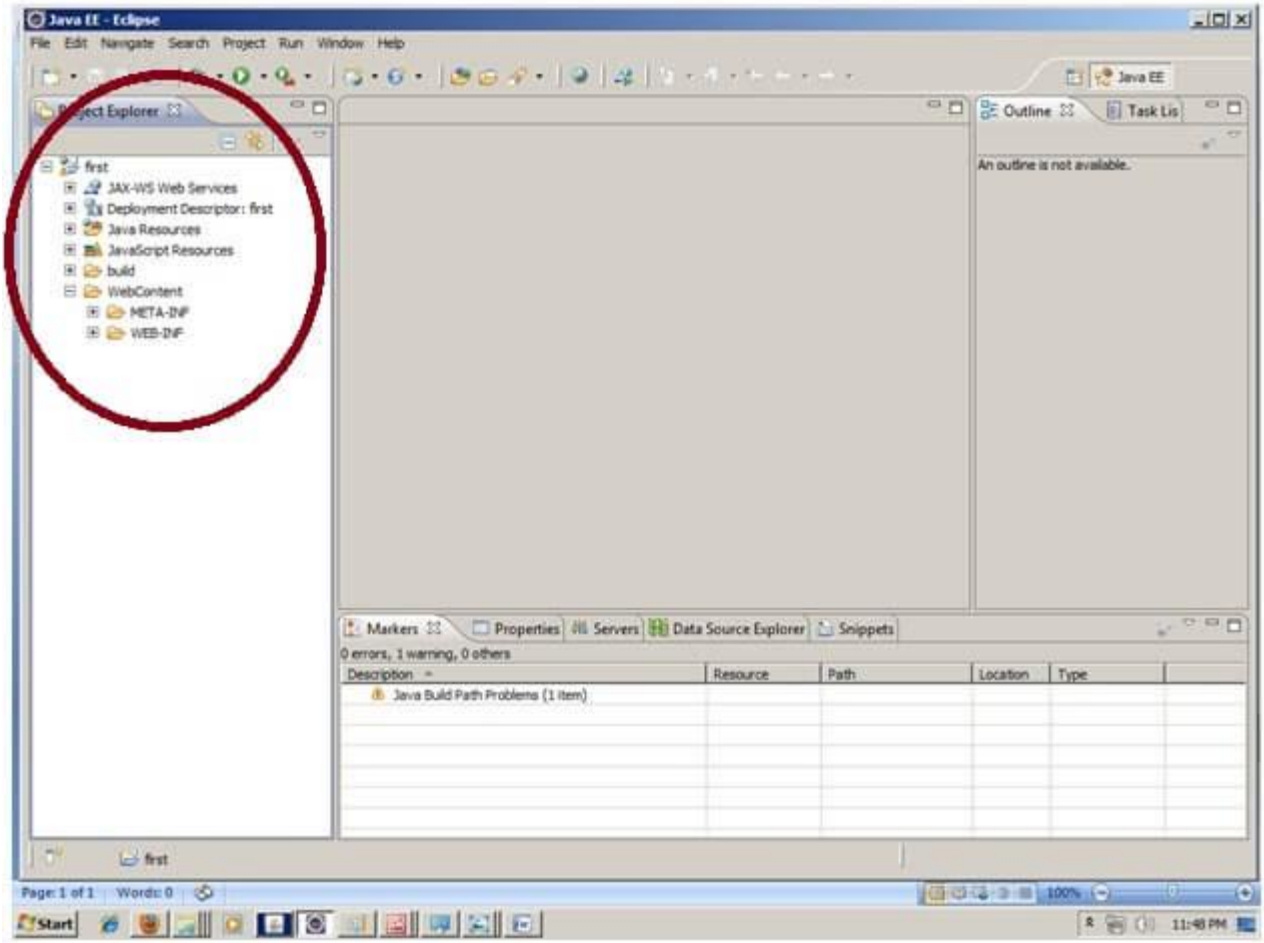

New Dynamic Web Project
Java
Configure project for building a Java application.
Source folders on build path:
src
Add Folder...
Edit...
Remove
Default output folder:
build\classes
< Back
Next >
Finish
Cancel
Java EE
Outline
Task Lis
utline is not available.
Type
0 items selected
Done
Start
11:44 PM

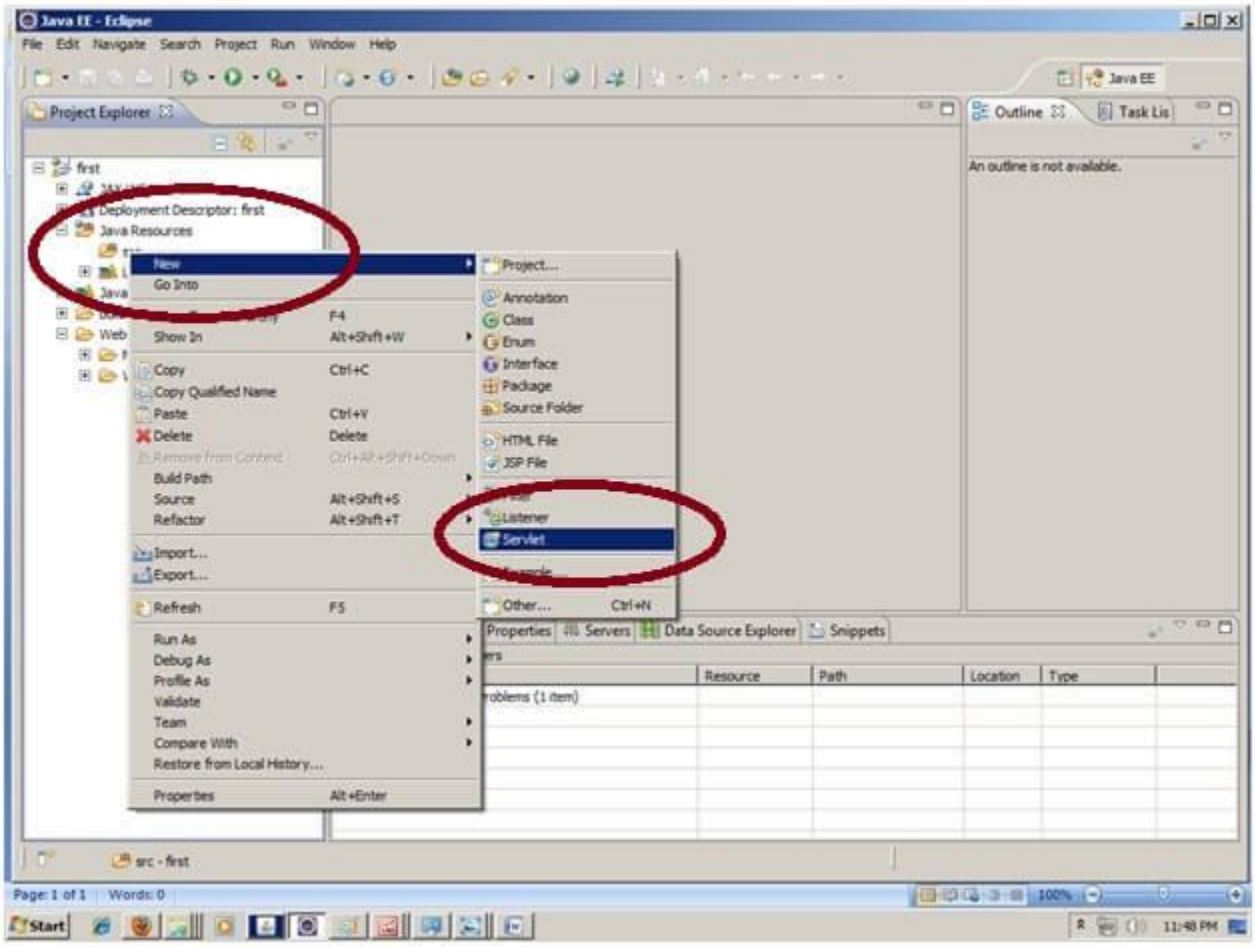

New Dynamic Web Project
Web Module
Configure web module settings.
Context root:
first
Content directory:
WebContent
Generate web.xml deployment descriptor
< Back
Next >
Finish
Cancel
Outline
Task Lis
utline is not available.
Type
0 items selected
Page: 1 of 1
Words: 0
100%
Start
11:45 PM

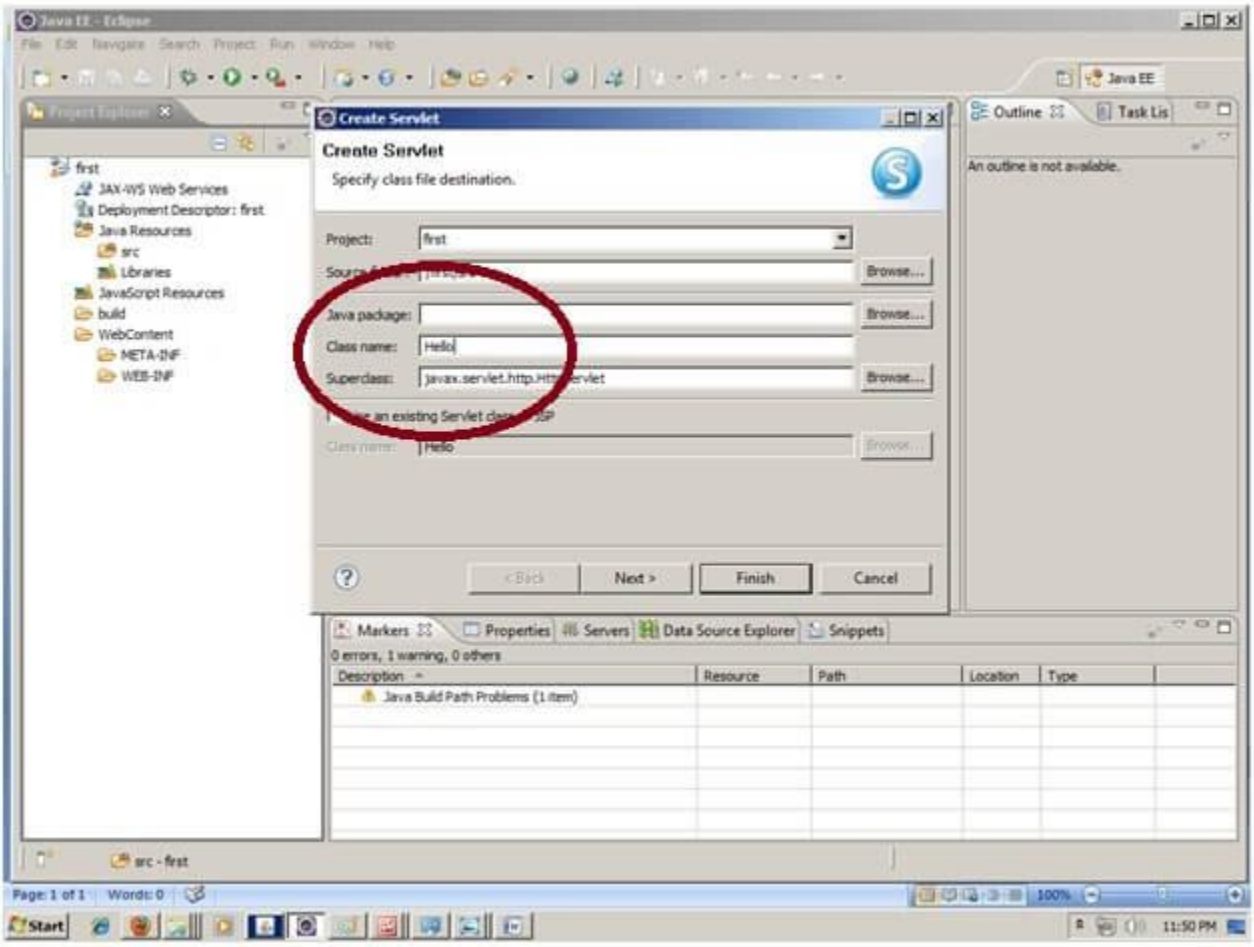


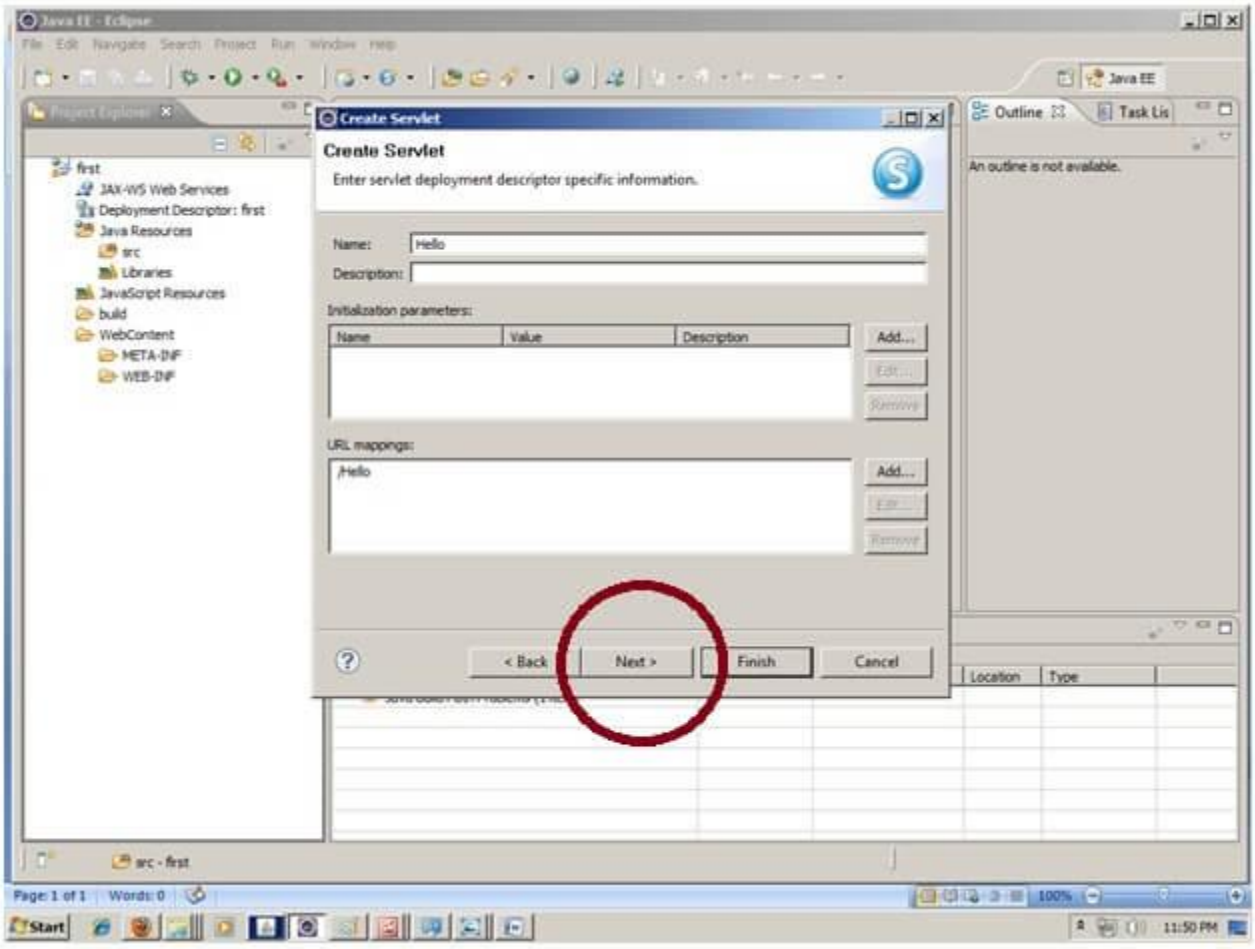
2) Create the servlet in eclipse IDE:

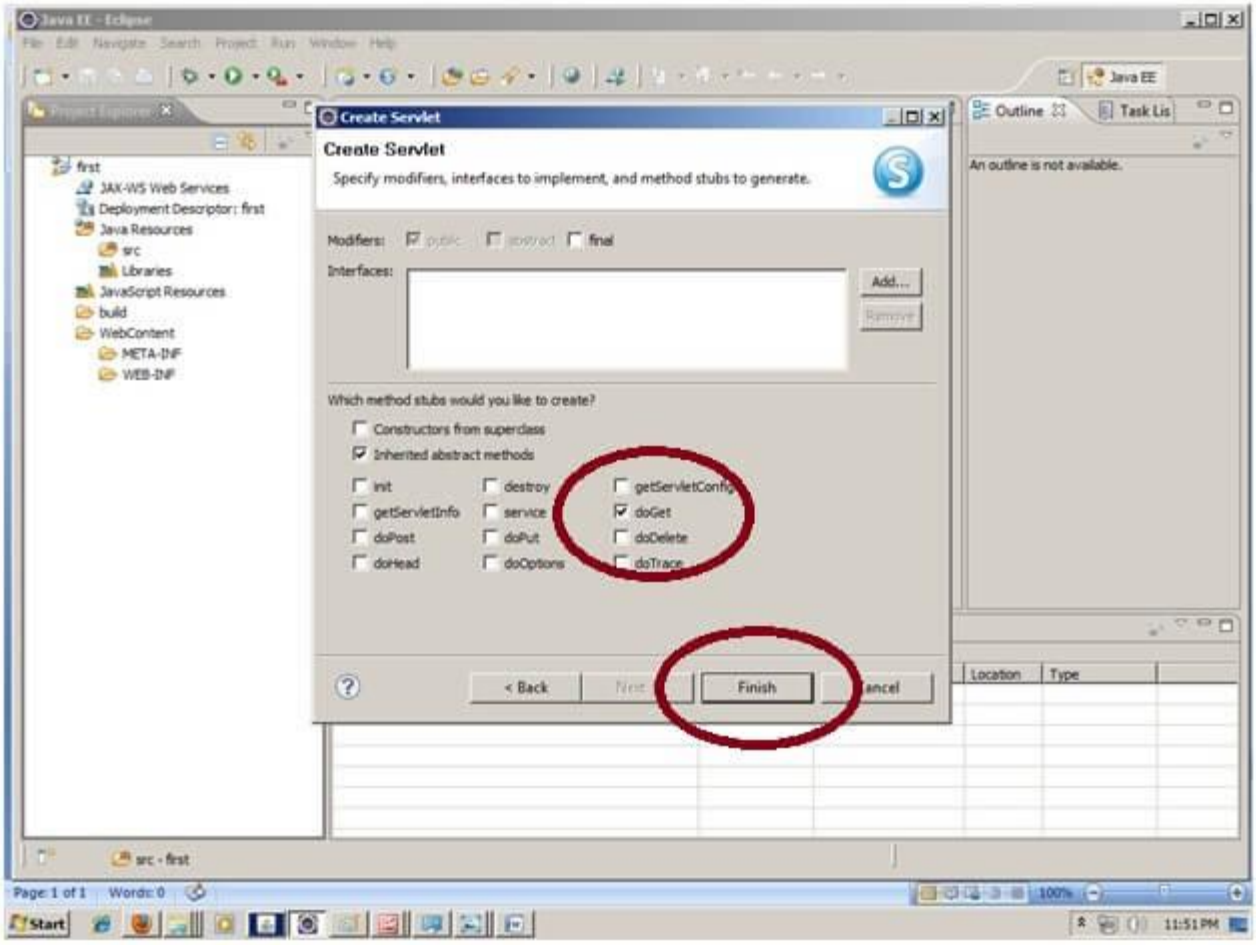
For creating a servlet, **explore the project by clicking the + icon -> explore the Java Resources -> right click on src -> New -> servlet -> write your servlet name e.g. Hello -> uncheck all the checkboxes except doGet() -> next -> Finish**.

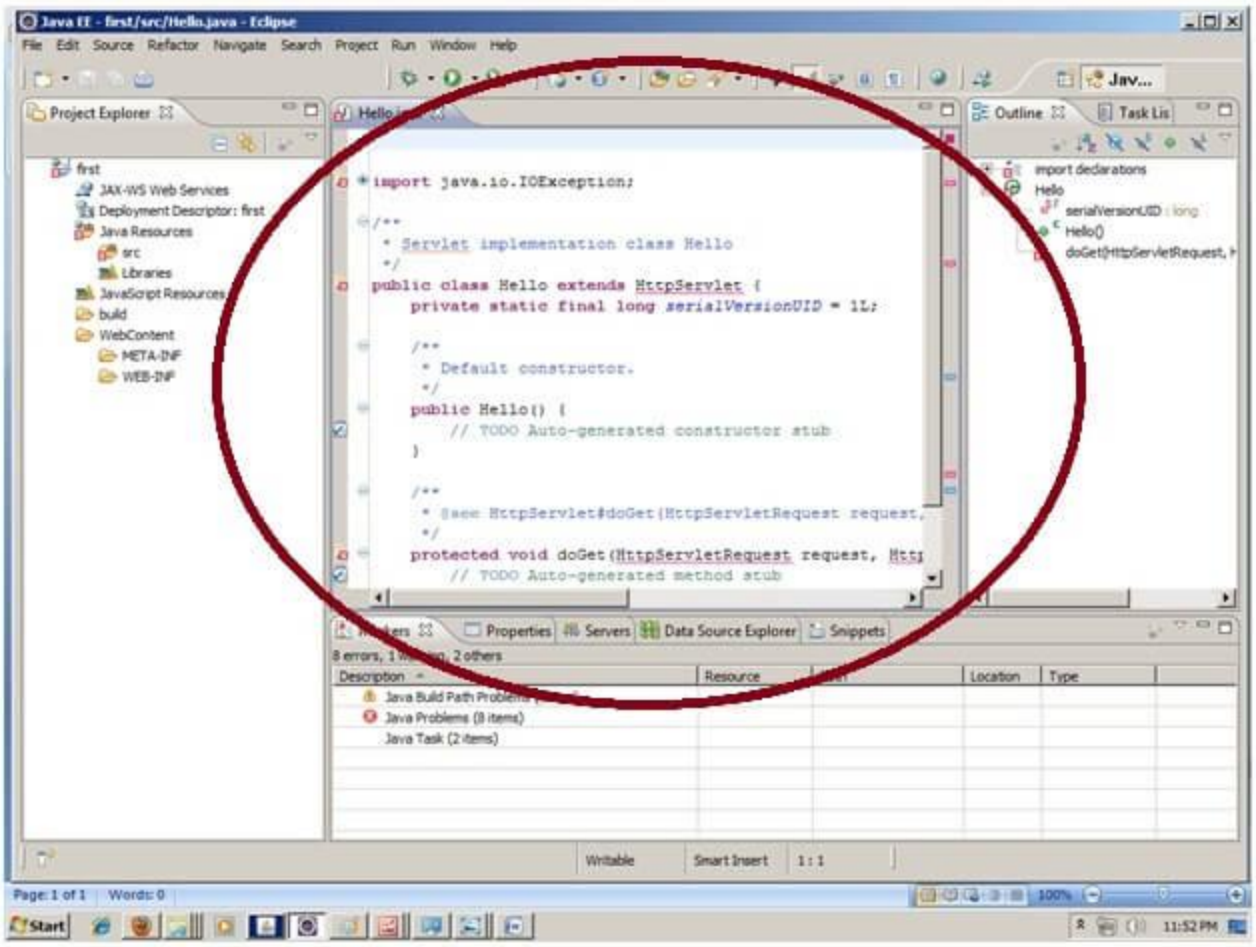

Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
Java EE
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
JavaScript Resources
build
WebContent
META-INF
WEB-INF
Outline
Task List
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
0 errors, 1 warning, 0 others
Description
Resource
Path
Location
Type
Java Build Path Problems (1 item)
first
Page: 1 of 1
Words: 0
100%
Start
11:48 PM


Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
Java EE
Project Explorer
Outline
Task Lis
An outline is not available.
first
Deployment Descriptor: first
Java Resources
New
Go Into
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Run As
Debug As
Profile As
Validate
Team
Compare With
Restore from Local History...
Properties Alt+Enter
Project...
Annotation
Class
Enum
Interface
Package
Source Folder
HTML File
JSP File
Listener
Servlet
Other... Ctrl+N
Properties
Servers
Data Source Explorer
Snippets
Resource
Path
Location
Type
Problems (1 item)
src - first
Page: 1 of 1
Words: 0
100%
Start
11:48 PM


Create Servlet
Create Servlet
Specify class file destination.
Project:
first
Java package:
Class name:
Hello
Superclass:
Browse...
Next >
Finish
Cancel
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
Outline
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
0 errors, 1 warning, 0 others
Description
Resource
Path
Location
Type
Java Build Path Problems (1 item)
src - first

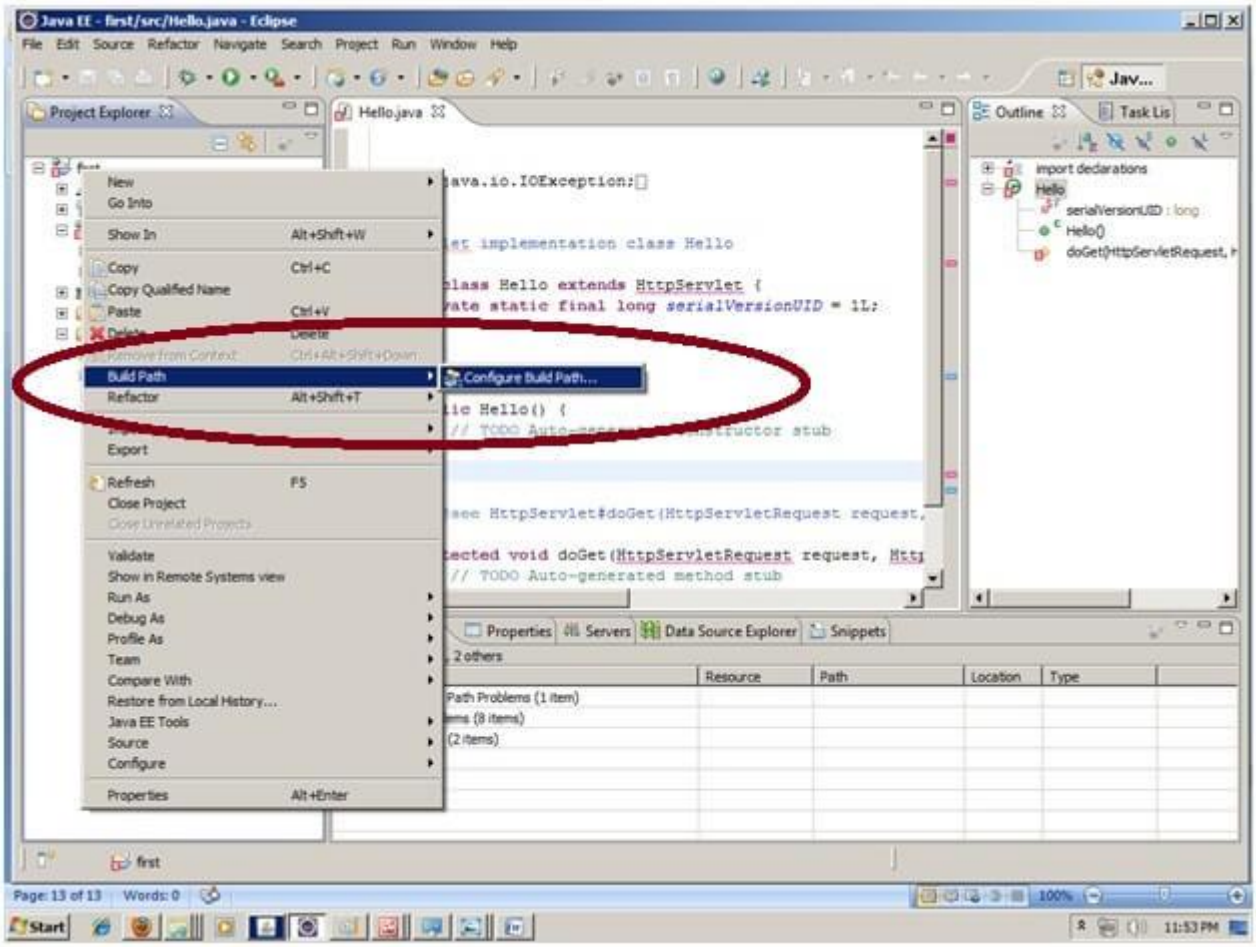

Create Servlet
Create Servlet
Enter servlet deployment descriptor specific information.
Name: Hello
Description:
Initialization parameters:
Name
Value
Description
Add...
Edit...
Remove
URL mappings:
/Hello
Add...
Edit...
Remove
< Back
Next >
Finish
Cancel
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
Outline
Task Lis
An outline is not available.
Location
Type
src - first
Page: 1 of 1
Words: 0
100%
Start
11:50 PM

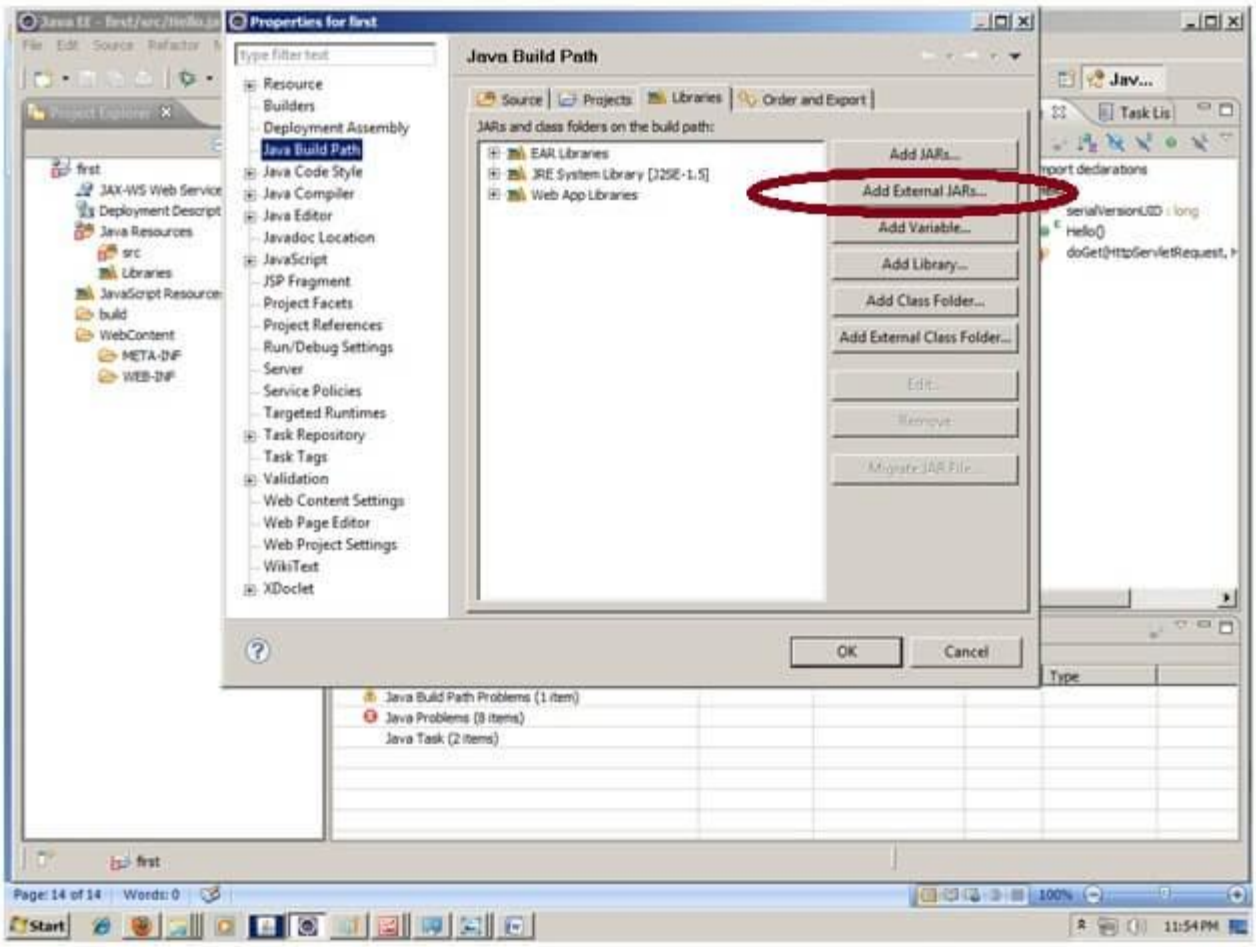

Create Servlet
Create Servlet
Specify modifiers, interfaces to implement, and method stubs to generate.
Modifiers:
public
abstract
final
Interfaces:
Add...
Remove
Which method stubs would you like to create?
Constructors from superclass
Inherited abstract methods
init
destroy
getServletConfig
getServletInfo
service
doGet
doPost
doPut
doDelete
doHead
doOptions
doTrace
< Back
Next >
Finish
Cancel
Outline
Task List
An outline is not available.
Location
Type
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
src - first
Page: 1 of 1
Words: 0
Start
11:51 PM



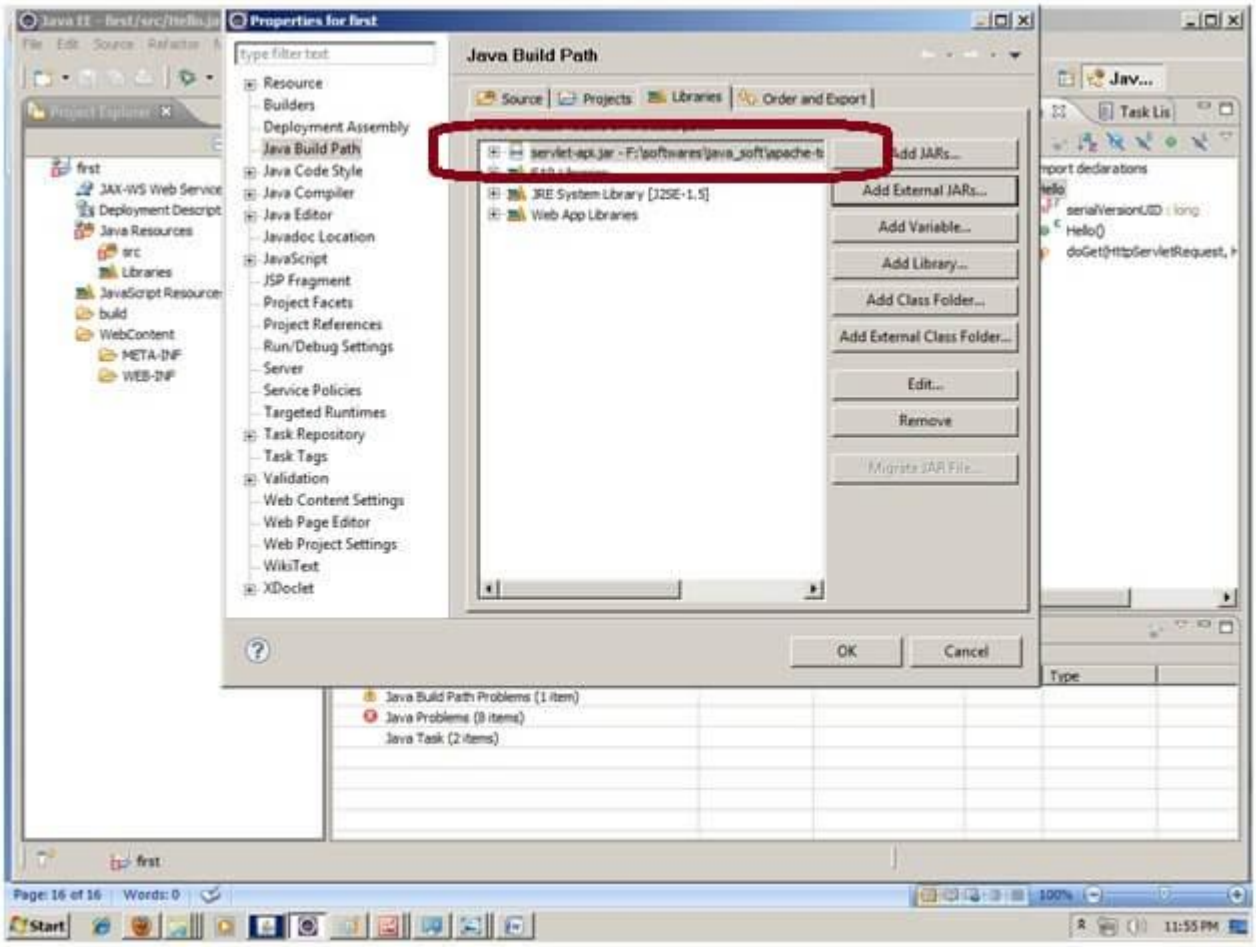
3) add jar file in eclipse IDE:

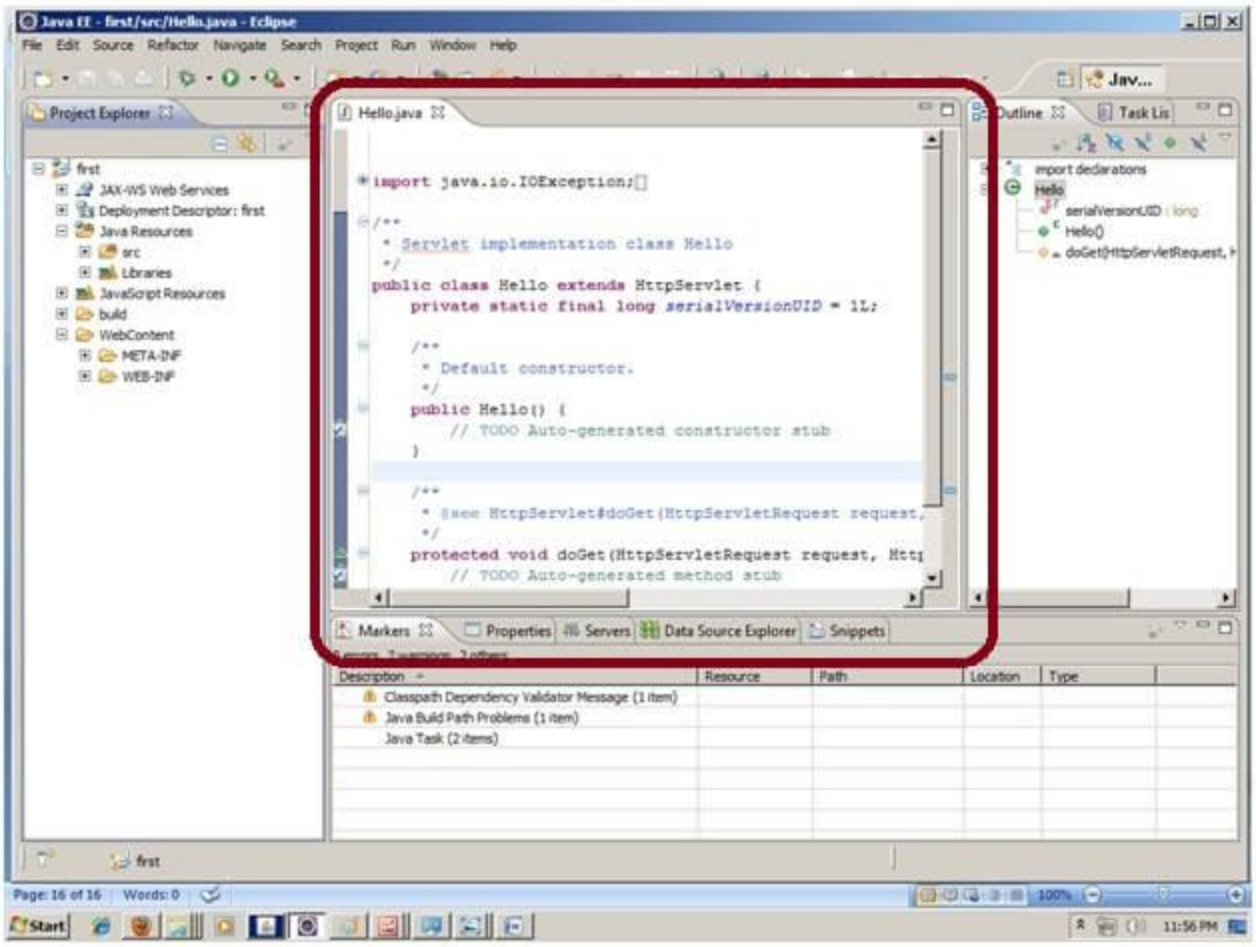
For adding a jar file, **right click on your project -> Build Path -> Configure Build Path -> click on Libraries tab in Java Build Path -> click on Add External JARs button -> select the servlet-api.jar file under tomcat/lib -> ok.**

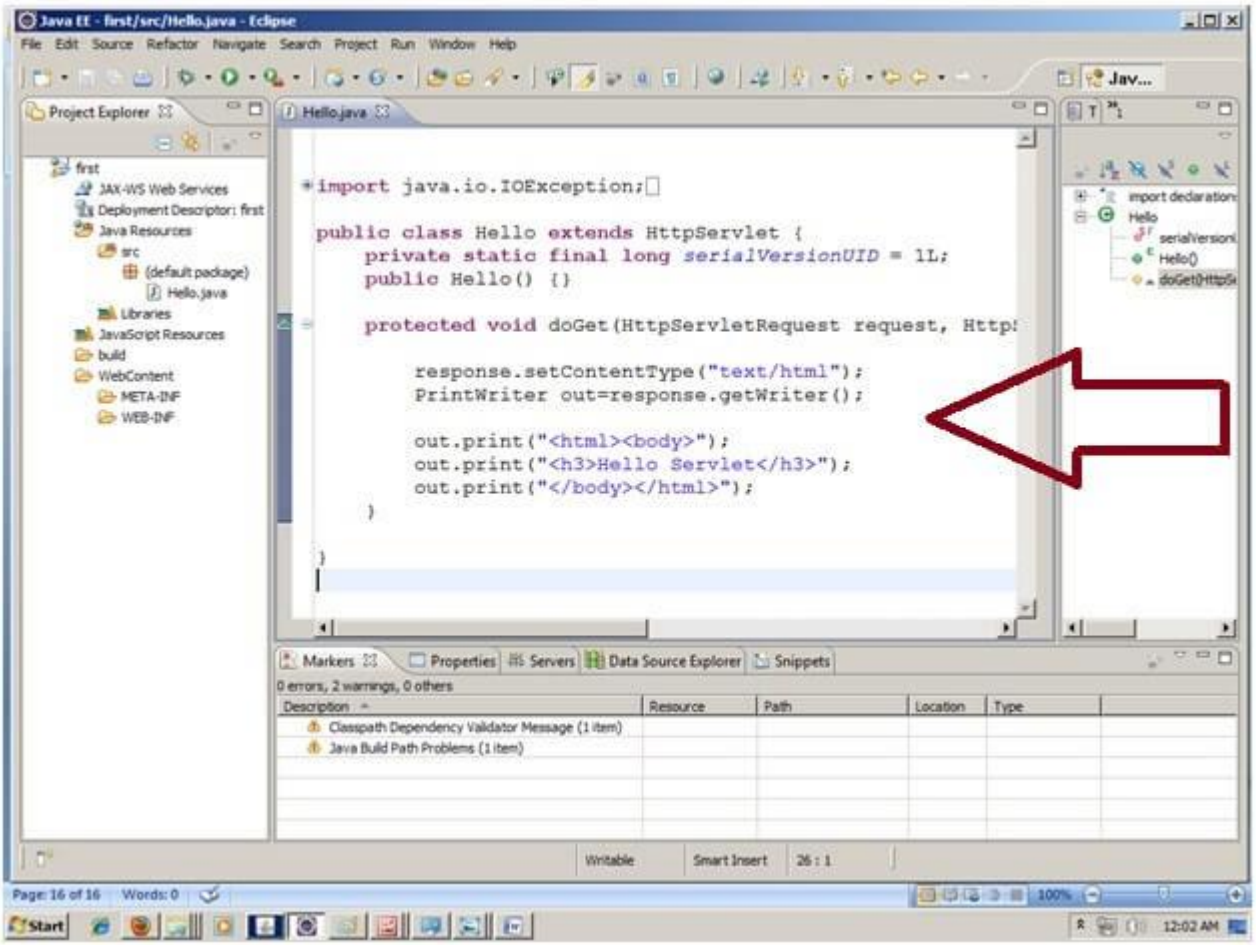

Java EE - first/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Hello.java
Outline
Task Lis
import declarations
Hello
serialVersionUID : long
Hello()
doGet(HttpServletRequest, h
New
Go Into
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Refactor Alt+Shift+T
Export
Refresh F5
Close Project
Close Unrelated Projects
Validate
Show in Remote Systems view
Run As
Debug As
Profile As
Team
Compare With
Restore from Local History...
Java EE Tools
Source
Configure
Properties Alt+Enter
Configure Build Path...
Properties
Servers
Data Source Explorer
Snippets
2 others
Resource
Path
Location
Type
Path Problems (1 item)
(2 items)
first
Page: 13 of 13
Words: 0
100%
Start
11:53 PM


Properties for first
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
EAR Libraries
JRE System Library [J2SE-1.5]
Web App Libraries
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
OK
Cancel
Resource
Builders
Deployment Assembly
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
JavaScript
JSP Fragment
Project Facets
Project References
Run/Debug Settings
Server
Service Policies
Targeted Runtimes
Task Repository
Task Tags
Validation
Web Content Settings
Web Page Editor
Web Project Settings
WikiText
XDoclet
first
JAX-WS Web Service
Deployment Descript
Java Resources
src
Libraries
JavaScript Resource
build
WebContent
META-INF
WEB-INF
Java Build Path Problems (1 item)
Java Problems (8 items)
Java Task (2 items)
Type
Page: 14 of 14
Words: 0
100%
Start
11:54 PM


Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
EAR Libraries
JRE System Library [J2SE-1.5]
Web App Libraries
Add JARs...
Add External JARs...
Add Variable...
JAR Selection
apache-tomcat-7.0.23 ▸ lib
Search lib
Organize
New folder
Desktop
Downloads
Recent Places
Libraries
Documents
Music
Pictures
Videos
Computer
Local Disk (C:)
Works (D:)
Local Disk (G:)
Name
Date modified
Type
annotations-api 11/20/2011 7:38 PM Executab
catalina 11/20/2011 7:38 PM Executab
catalina-ant 11/20/2011 7:38 PM Executab
catalina-ha 11/20/2011 7:38 PM Executab
catalina-tribes 11/20/2011 7:38 PM Executab
ecj-3.7 11/20/2011 7:38 PM Executab
el-api 11/20/2011 7:38 PM Executab
jasper 11/20/2011 7:38 PM Executab
jasper-el 11/20/2011 7:38 PM Executab
jsp-api 11/20/2011 7:38 PM Executab
servlet-api 11/20/2011 7:38 PM Executab
Date modified: 11/20/2011 7:38 PM
File name: servlet-api
Open
Cancel
first
Page: 14 of 14
Words: 0
100%
Start
11:55 PM

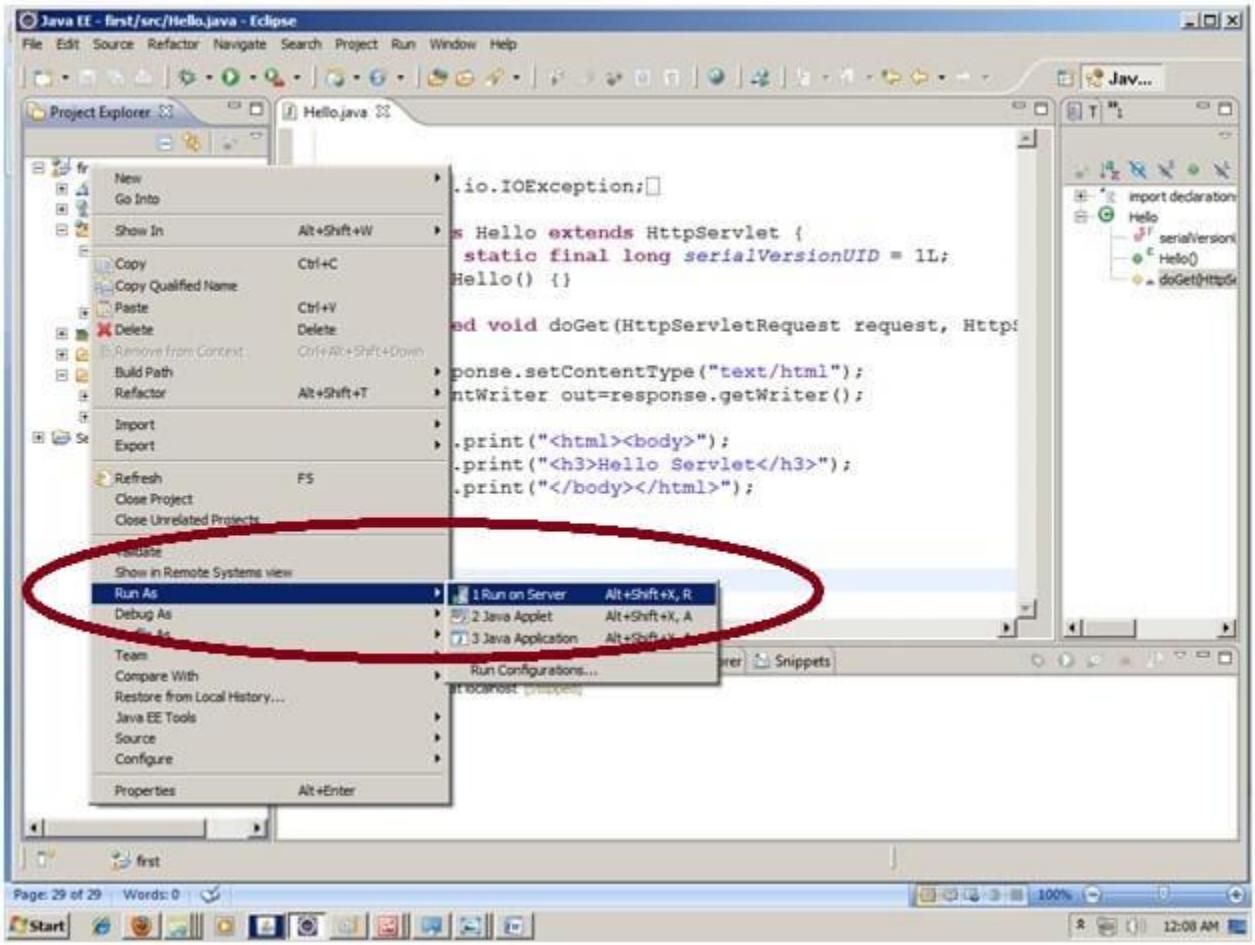

Properties for first
Java Build Path
Source
Projects
Libraries
Order and Export
servlet-api.jar - F:\softwares\java_soft\apache-t
JRE System Library [J2SE-1.5]
Web App Libraries
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
OK
Cancel
Resource
Builders
Deployment Assembly
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
JavaScript
JSP Fragment
Project Facets
Project References
Run/Debug Settings
Server
Service Policies
Targeted Runtimes
Task Repository
Task Tags
Validation
Web Content Settings
Web Page Editor
Web Project Settings
WikiText
XDoclet
first
JAX-WS Web Service
Deployment Descript
Java Resources
src
Libraries
JavaScript Resource
build
WebContent
META-INF
WEB-INF
Java Build Path Problems (1 item)
Java Problems (8 items)
Java Task (2 items)
Page: 16 of 16
Words: 0
Start
11:55 PM

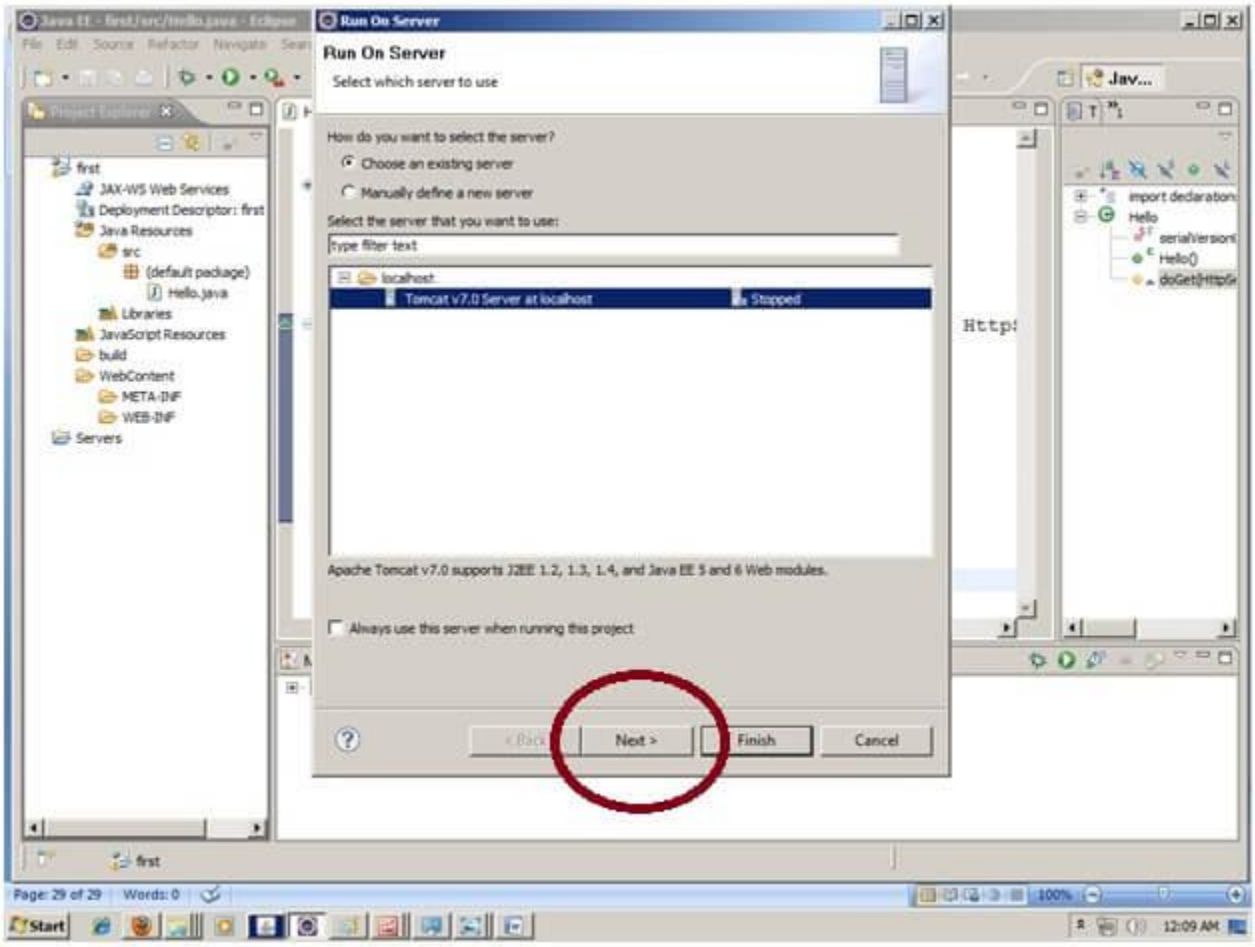

Java EE - first/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
Hello.java
import java.io.IOException;
/**
* Servlet implementation class Hello
*/
public class Hello extends HttpServlet {
private static final long serialVersionUID = 1L;
/**
* Default constructor.
*/
public Hello() {
// TODO Auto-generated constructor stub
}
/**
* @see HttpServlet#doGet(HttpServletRequest request,
*/
protected void doGet(HttpServletRequest request, Htt
// TODO Auto-generated method stub
Outline
Task Lis
import declarations
Hello
serialVersionUID : long
Hello()
doGet(HttpServletRequest,
Markers
Properties
Servers
Data Source Explorer
Snippets
Description
Resource
Path
Location
Type
Classpath Dependency Validator Message (1 item)
Java Build Path Problems (1 item)
Java Task (2 items)
first
Page: 16 of 16
Words: 0
100%
Start
11:56 PM

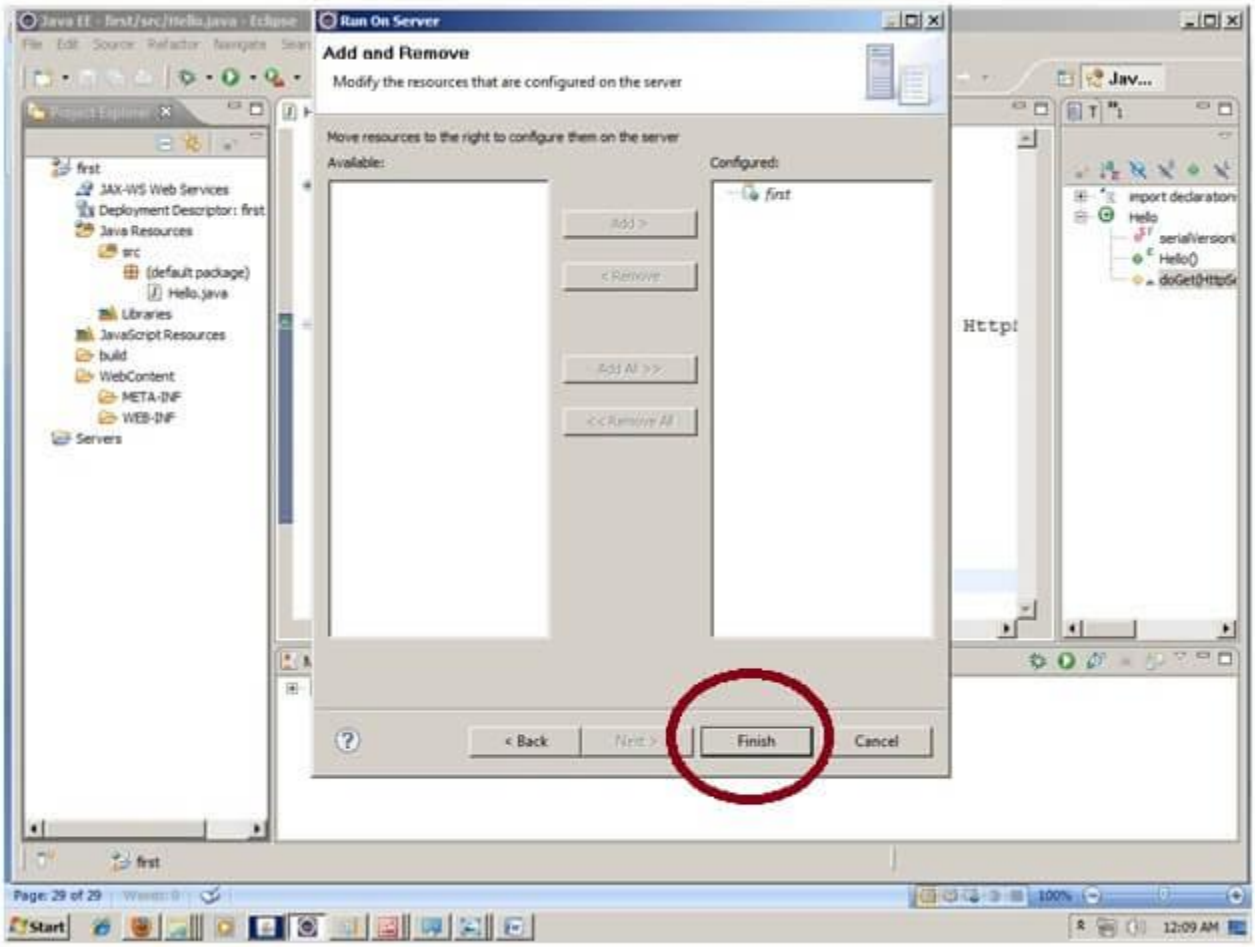


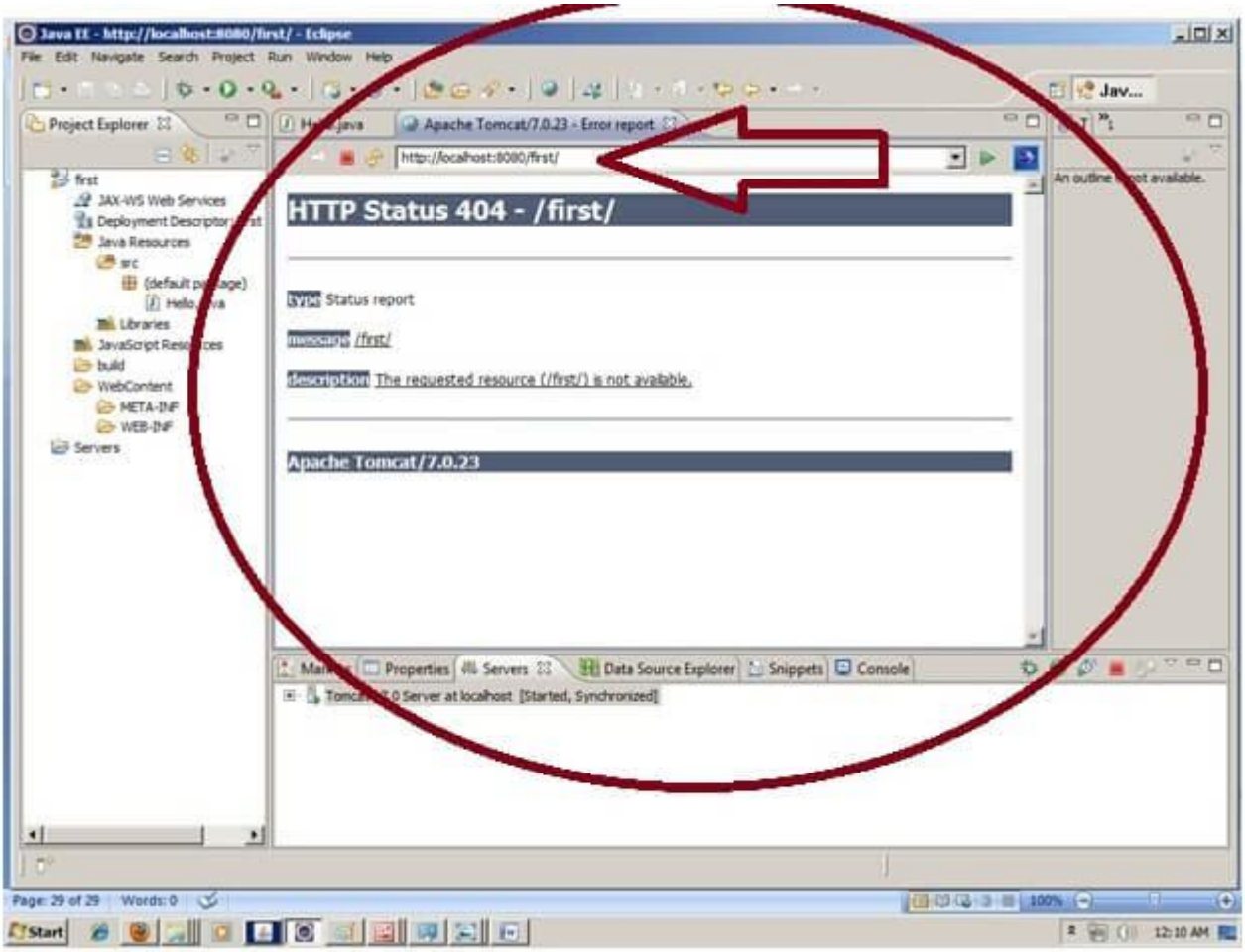
4) Start the server and deploy the project:

For starting the server and deploying the project in one step, **Right click on your project -> Run As -> Run on Server -> choose tomcat server -> next -> addAll -> finish.**

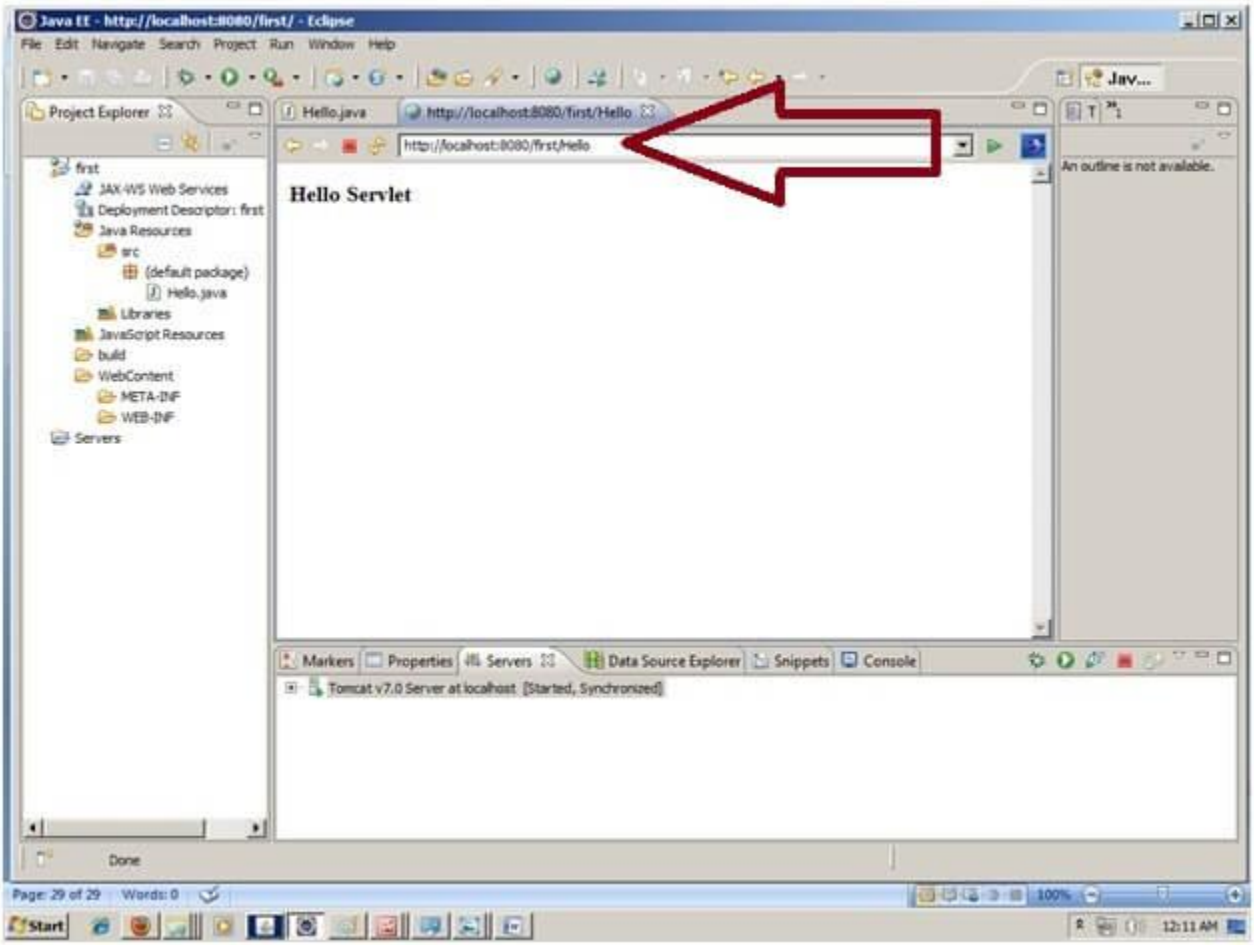

Java EE - first/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Hello.java
.io.IOException;
s Hello extends HttpServlet {
static final long serialVersionUID = 1L;
Hello() {}
ed void doGet(HttpServletRequest request, Https
ponse.setContentType("text/html");
ntWriter out=response.getWriter();
.print("<html><body>");
.print("<h3>Hello Servlet</h3>");
.print("</body></html>");
New
Go Into
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Refactor Alt+Shift+T
Import
Export
Refresh F5
Close Project
Close Unrelated Projects
Show in Remote Systems view
Run As
Debug As
Team
Compare With
Restore from Local History...
Java EE Tools
Source
Configure
Properties Alt+Enter
1 Run on Server Alt+Shift+X, R
2 Java Applet Alt+Shift+X, A
3 Java Application
Run Configurations...
Snippets
first
Page: 29 of 29 Words: 0
100%
Start
12:08 AM


Run On Server
Run On Server
Select which server to use
How do you want to select the server?
Choose an existing server
Manually define a new server
Select the server that you want to use:
type filter text
localhost
Tomcat v7.0 Server at localhost
Stopped
Apache Tomcat v7.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5 and 6 Web modules.
Always use this server when running this project
< Back
Next >
Finish
Cancel
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
Servers
Page: 29 of 29
Words: 0
100%
Start
12:09 AM


Run On Server
Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
first
Add >
< Remove
Add All >>
<< Remove All
< Back
Next >
Finish
Cancel
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
Servers
Hello()
Page: 29 of 29
Start
12:09 AM



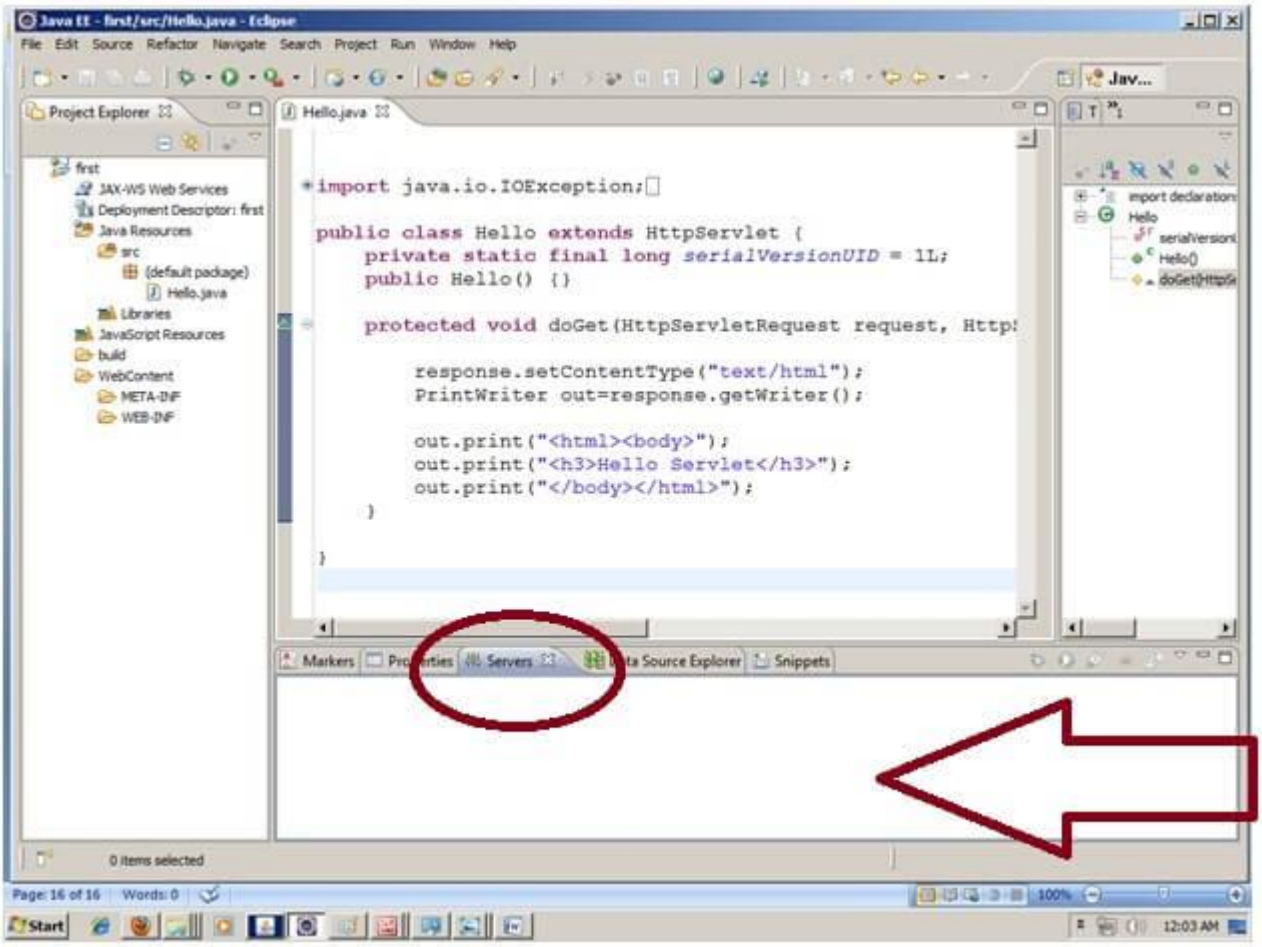
Now tomcat server has been started and project is deployed. To access the servlet write the url pattern name in the URL bar of the browser. In this case Hello then enter.

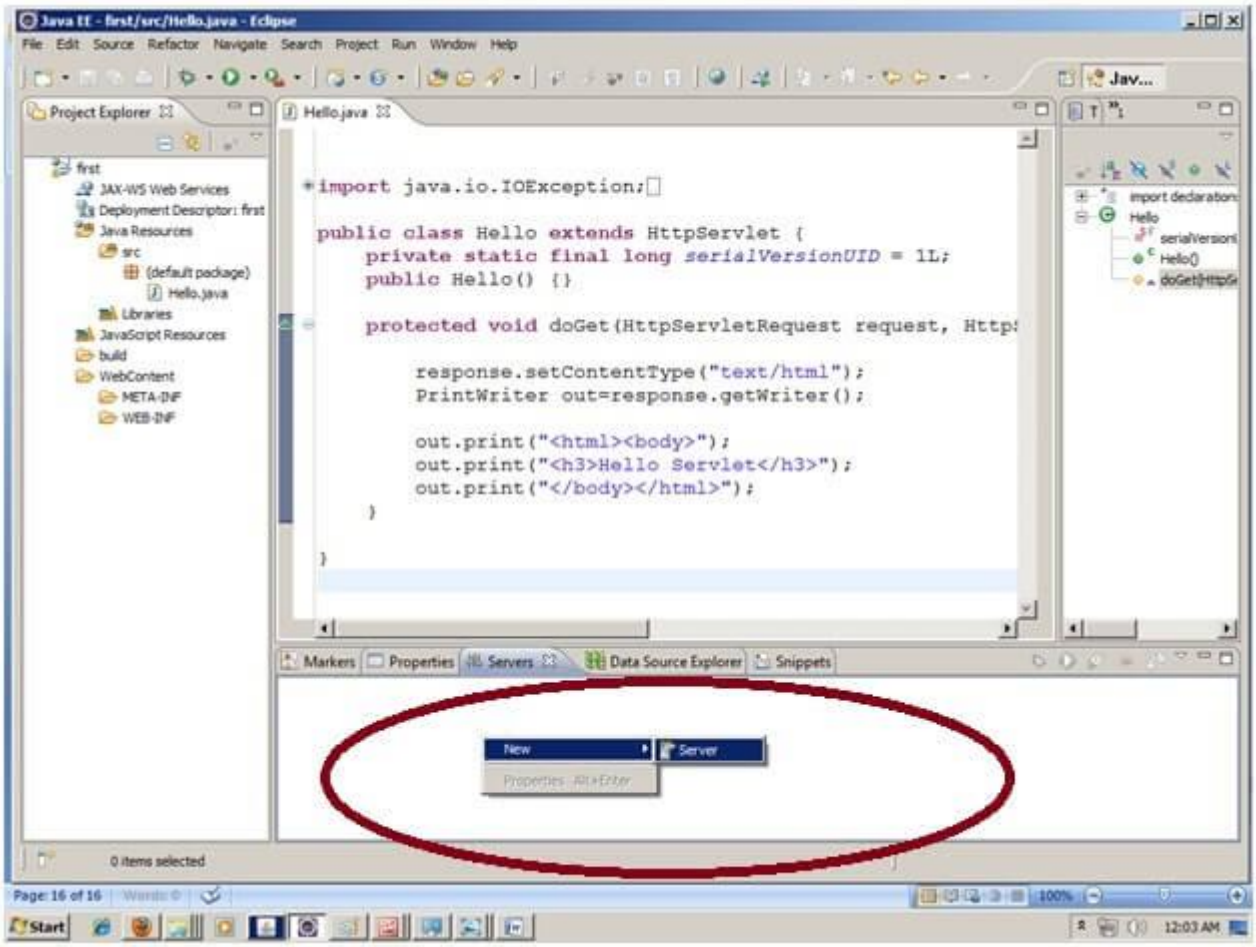


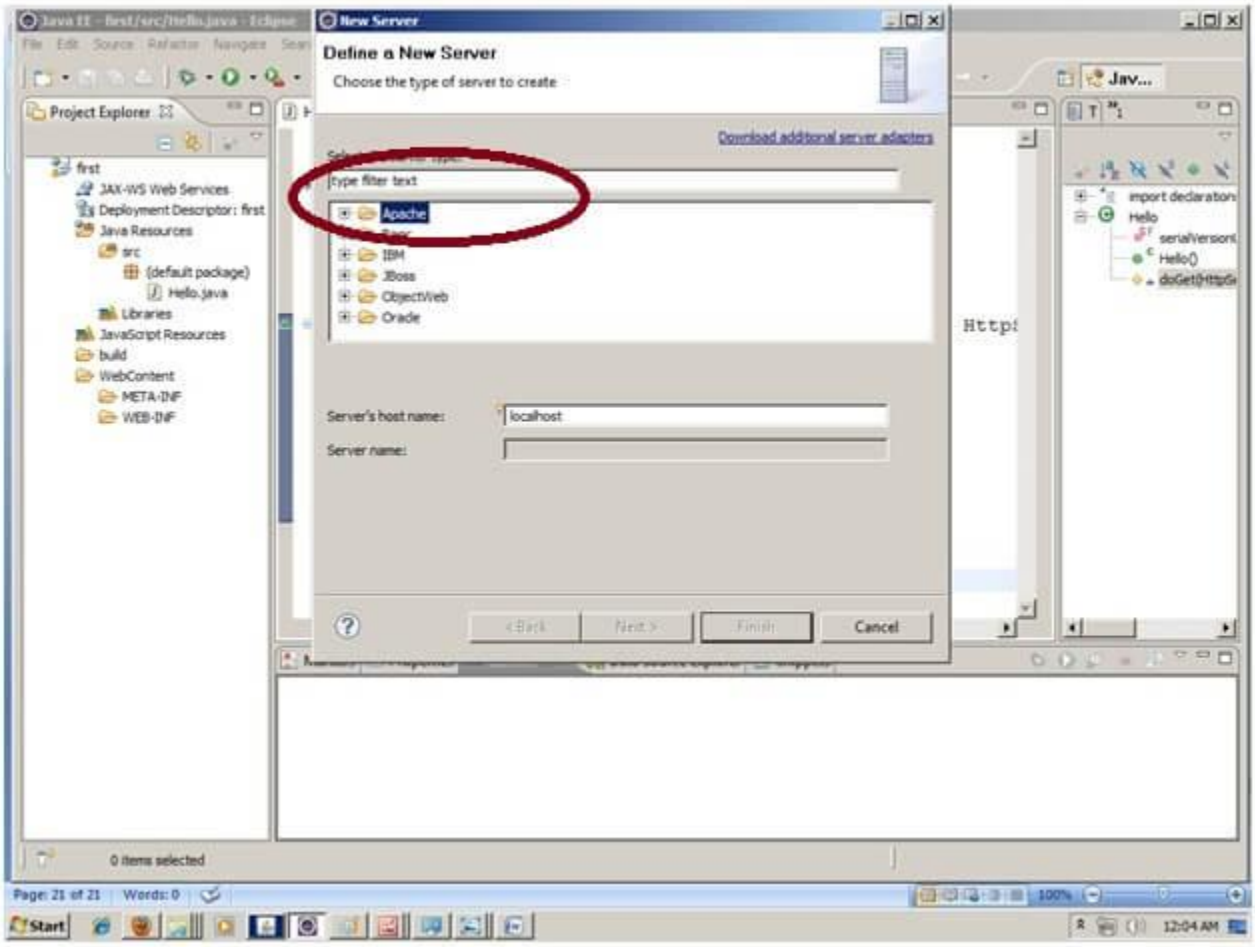
How to configure tomcat server in Eclipse ? (One time Requirement)

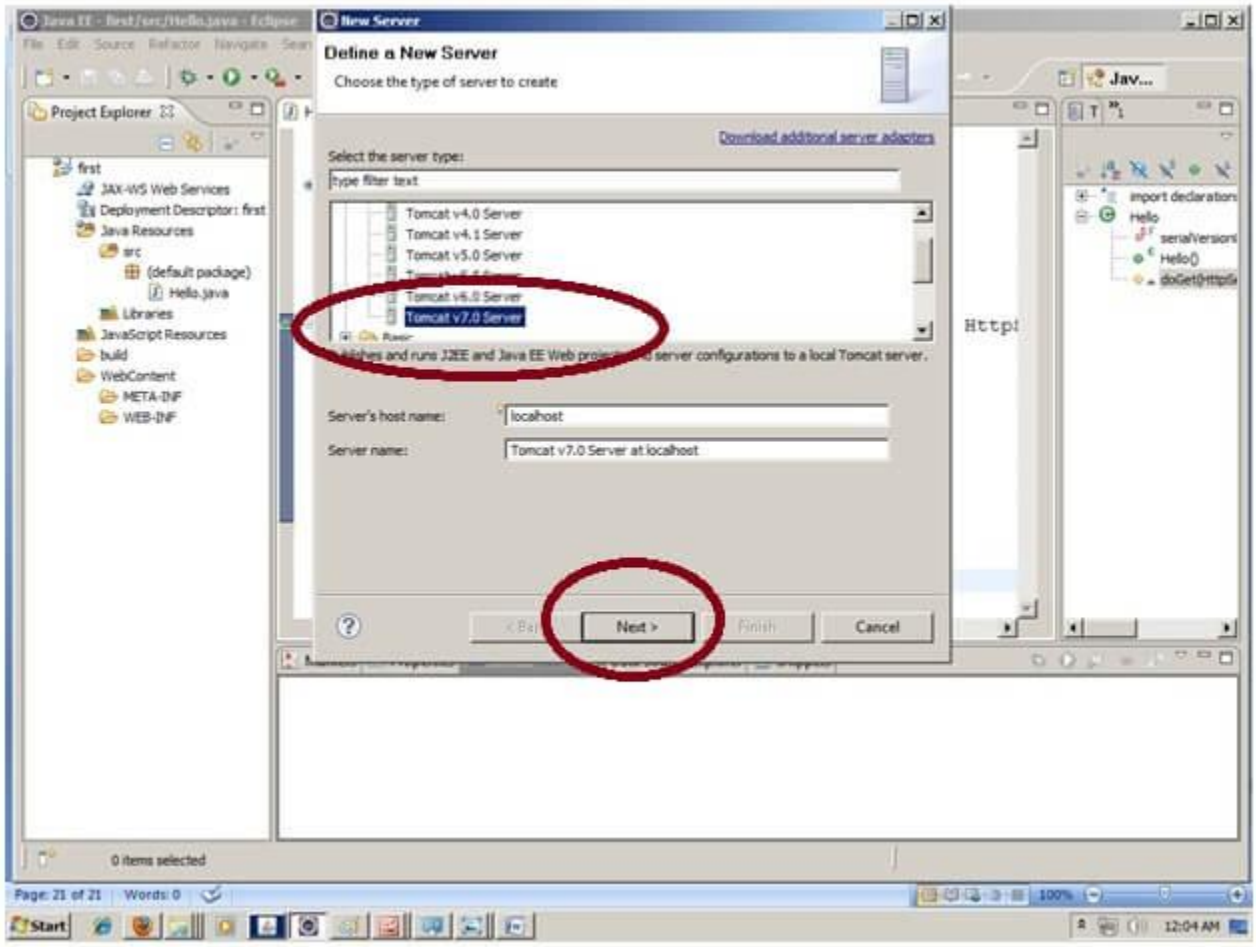
If you are using **Eclipse IDE first time**, you need to configure the tomcat server First.

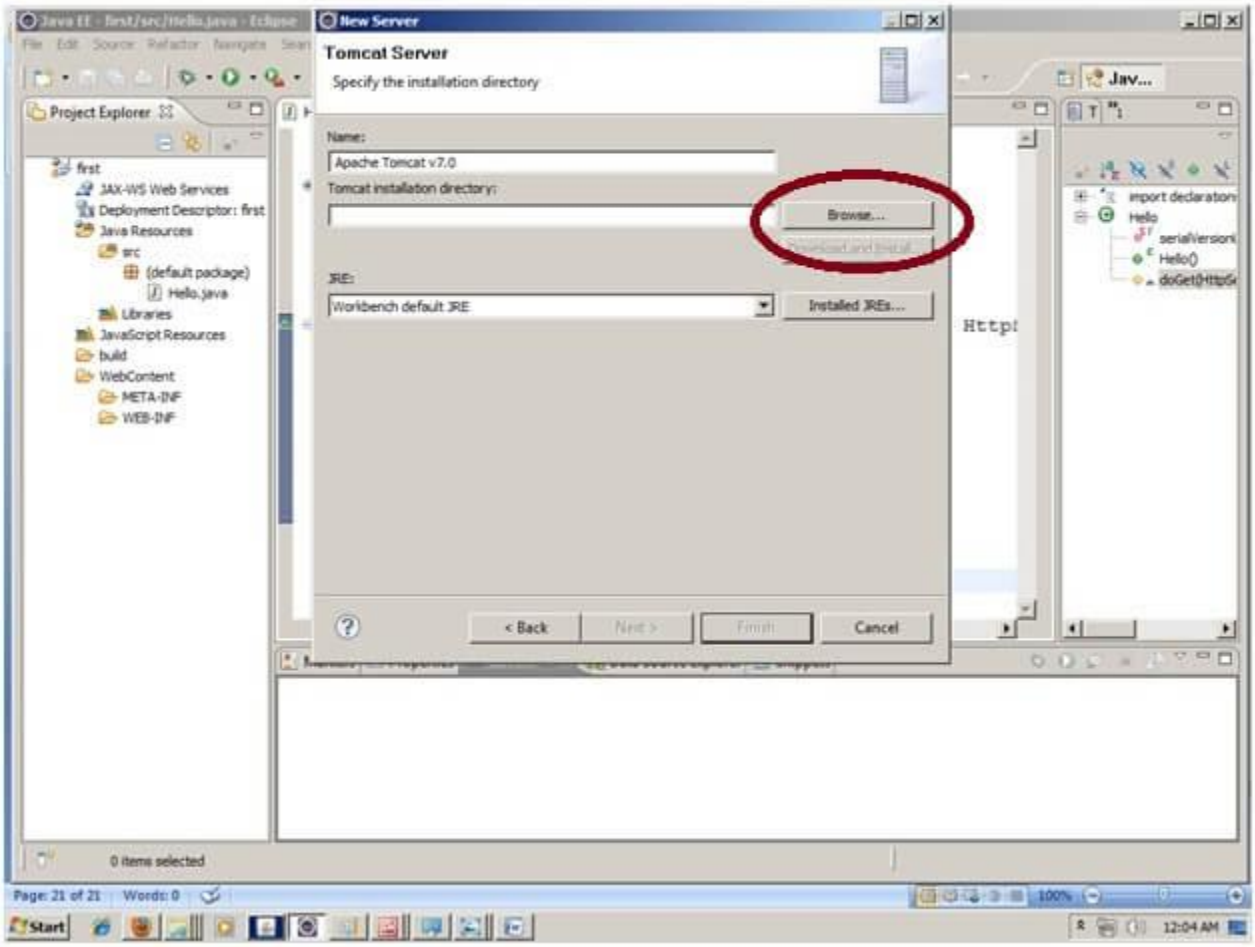
For configuring the tomcat server in eclipse IDE, **click on servers tab at the bottom side of the IDE -> right click on blank area -> New -> Servers -> choose tomcat then its version -> next -> click on Browse button -> select the apache tomcat root folder previous to bin -> next -> addAll -> Finish.**

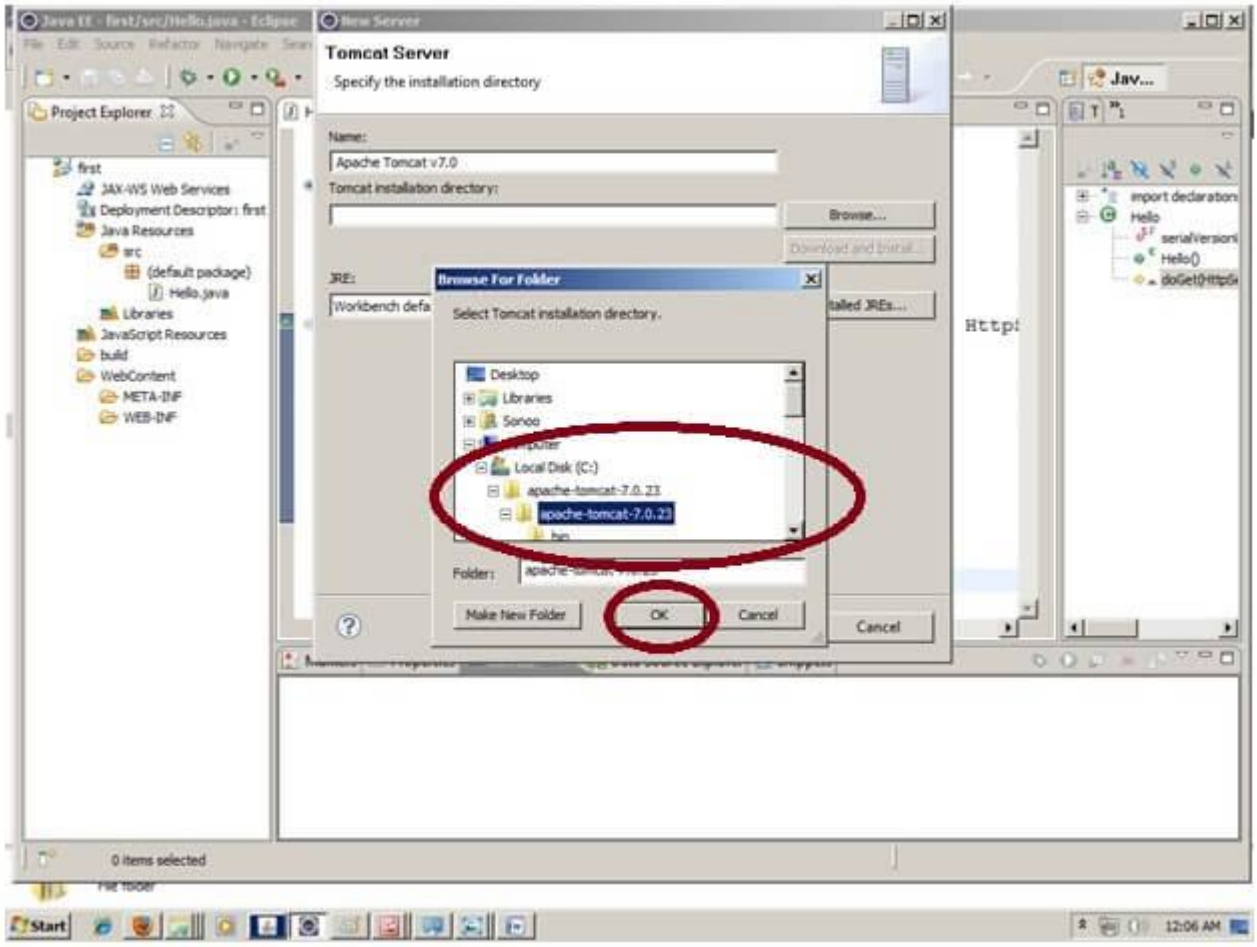

Java EE - first/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
Hello.java
import java.io.IOException;
public class Hello extends HttpServlet {
private static final long serialVersionUID = 1L;
public Hello() {}
protected void doGet(HttpServletRequest request, Http
response.setContentType("text/html");
PrintWriter out=response.getWriter();
out.print("<html><body>");
out.print("<h3>Hello Servlet</h3>");
out.print("</body></html>");
}
}
Markers
Servers
Snippets
0 items selected
Page: 16 of 16
Words: 0
100%
Start
12:03 AM

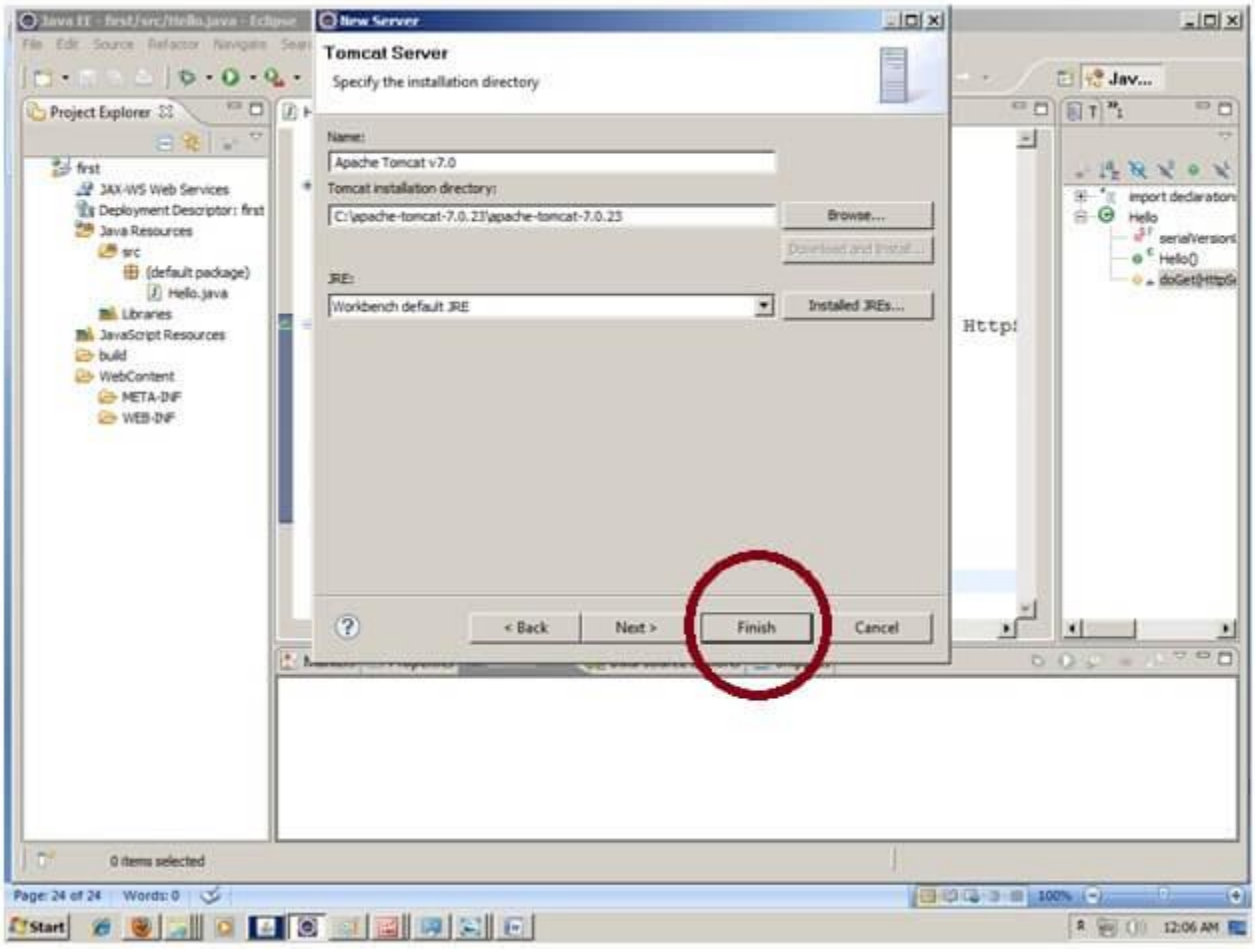

Java EE - first/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
Hello.java
import java.io.IOException;
public class Hello extends HttpServlet {
private static final long serialVersionUID = 1L;
public Hello() {}
protected void doGet(HttpServletRequest request, Https
response.setContentType("text/html");
PrintWriter out=response.getWriter();
out.print("<html><body>");
out.print("<h3>Hello Servlet</h3>");
out.print("</body></html>");
}
}
Markers
Properties
Servers
Data Source Explorer
Snippets
New
Server
Properties Alt+Enter
0 items selected
Page: 16 of 16
100%
Start
12:03 AM

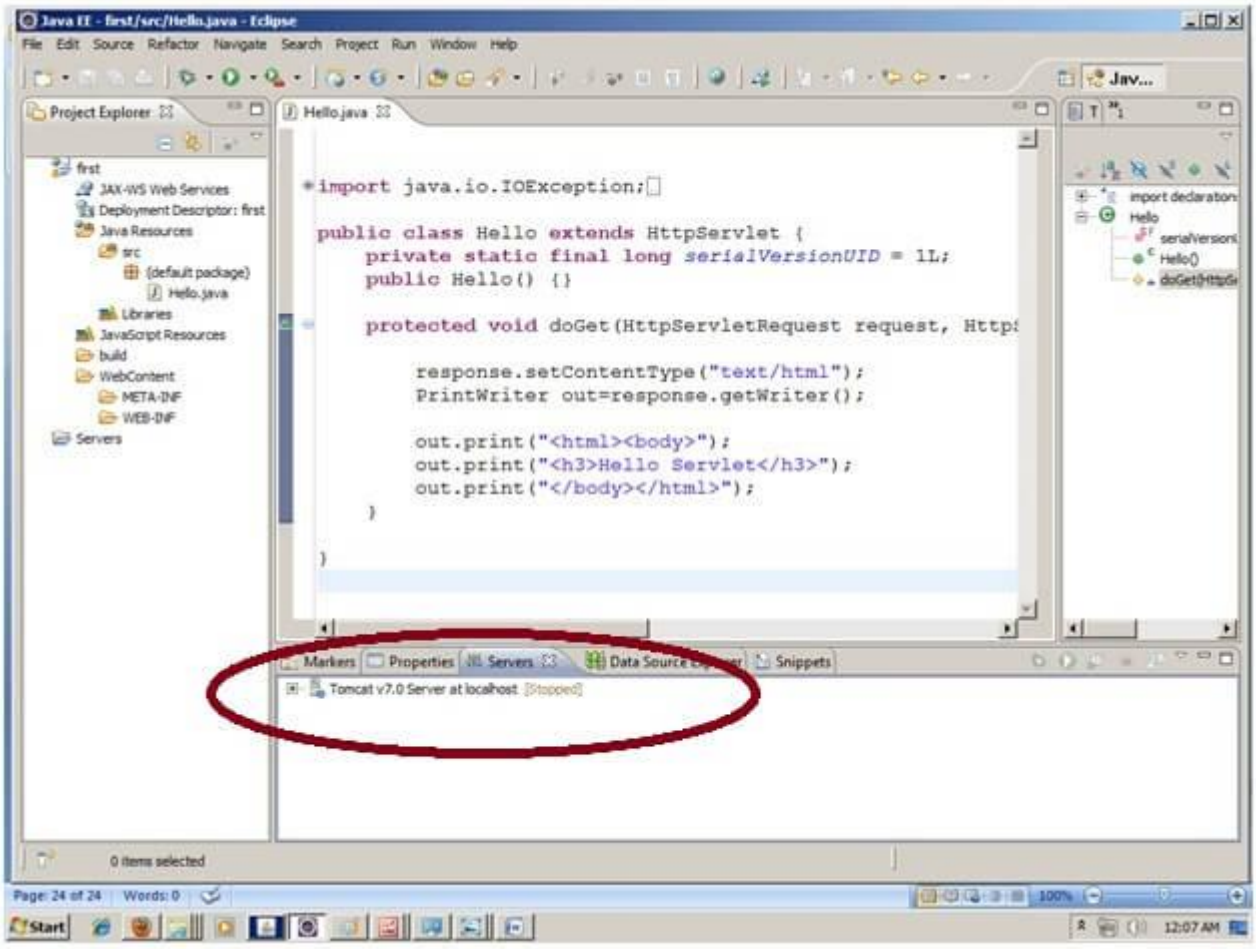

New Server
Define a New Server
Choose the type of server to create
Download additional server adapters
type filter text
Apache
IBM
JBoss
ObjectWeb
Oracle
Server's host name:
localhost
Server name:
< Back
Next >
Finish
Cancel
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
0 items selected
Page: 21 of 21
Words: 0
Start
12:04 AM


New Server
Define a New Server
Choose the type of server to create
Download additional server adapters
Select the server type:
type filter text
Tomcat v4.0 Server
Tomcat v4.1 Server
Tomcat v5.0 Server
Tomcat v6.0 Server
Tomcat v7.0 Server
Server's host name:
localhost
Server name:
Tomcat v7.0 Server at localhost
Next >
Finish
Cancel
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
0 items selected


New Server
Tomcat Server
Specify the installation directory
Name:
Apache Tomcat v7.0
Tomcat installation directory:
Browse...
JRE:
Workbench default JRE
Installed JREs...
< Back
Next >
Finish
Cancel
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
0 items selected
Page: 21 of 21
Words: 0
100%
Start
12:04 AM


Tomcat Server
Specify the installation directory
Name:
Apache Tomcat v7.0
Tomcat installation directory:
Browse...
JRE:
Browse For Folder
Select Tomcat installation directory.
Desktop
Libraries
Local Disk (C:)
apache-tomcat-7.0.23
Folder:
Make New Folder
OK
Cancel
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
0 items selected
Start
12:06 AM


New Server
Tomcat Server
Specify the installation directory
Name:
Apache Tomcat v7.0
Tomcat installation directory:
C:\apache-tomcat-7.0.23\apache-tomcat-7.0.23
Browse...
Download and Install...
JRE:
Workbench default JRE
Installed JREs...
< Back
Next >
Finish
Cancel
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
0 items selected
Page: 24 of 24
Words: 0
100%
Start
12:06 AM



ServletRequest Interface

An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc.

Methods of ServletRequest interface

There are many methods defined in the ServletRequest interface. Some of them are as follows:

Method	Description
public String getParameter(String name)	is used to obtain the value of a parameter by name.
public String[] getParameterValues(String name)	returns an array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box.
java.util.Enumeration getParameterNames()	returns an enumeration of all of the request parameter names.
public int getContentLength()	Returns the size of the request entity data, or -1 if not known.
public String getCharacterEncoding()	Returns the character set encoding for the input of this request.
public String getContentType()	Returns the Internet Media Type of the request entity data, or null if not known.
public ServletInputStream getInputStream() throws IOException	Returns an input stream for reading binary data in the request body.
public abstract String getServerName()	Returns the host name of the server that received the request.
public int getServerPort()	Returns the port number on which this request was received.

Example of ServletRequest to display the name of the user

In this example, we are displaying the name of the user in the servlet. For this purpose, we have used the getParameter method that returns the value for the given request parameter name.

index.html

```
<form action="ServletReqDemo" method="get">
Enter your name<input type="text" name="name"><br>
<input type="submit" value="login">
</form>
```

ServletReqDemo.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServ extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)  throws ServletExcepti
on,IOException
{
res.setContentType("text/html");
PrintWriter pw=res.getWriter();

String name=req.getParameter("name");//will return value
pw.println("Welcome "+name);

pw.close();
}}
```

RequestDispatcher in Servlet

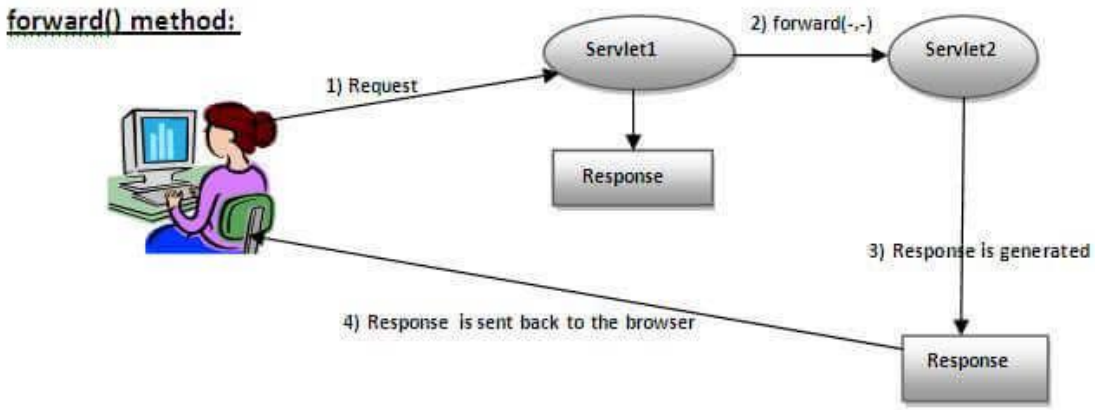
The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

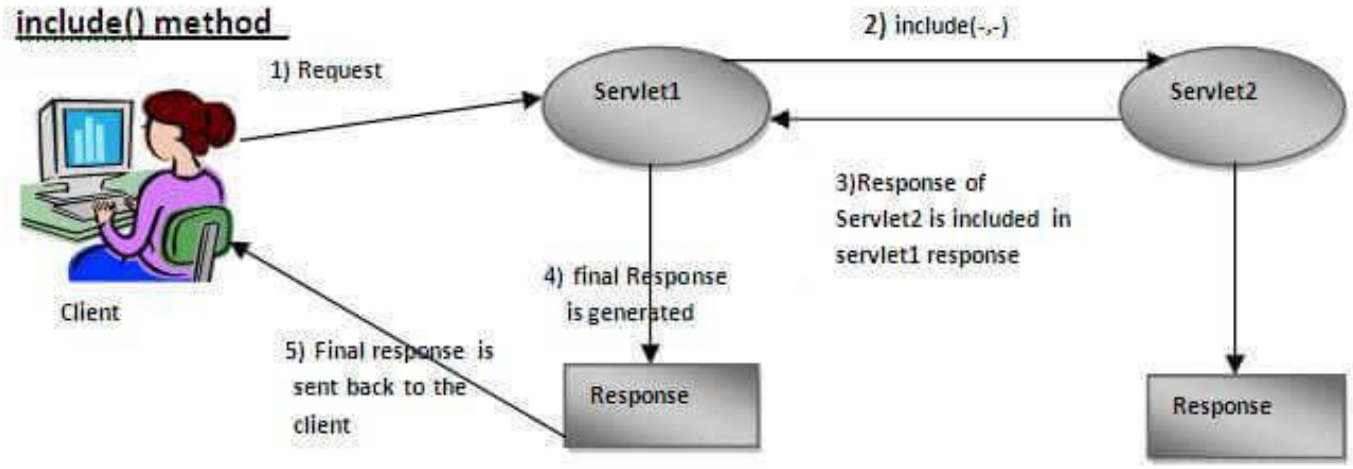
Methods of RequestDispatcher interface

The RequestDispatcher interface provides two methods. They are:

1. **public void forward(ServletRequest request,ServletResponse response)throws ServletException, java.io.IOException:** Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
2. **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:** Includes the content of a resource (servlet, JSP page, or HTML file) in the response.



As you see in the above figure, response of second servlet is sent to the client. Response of the first servlet is not displayed to the user.



As you can see in the above figure, response of second servlet is included in the response of the first servlet that is being sent to the client.

How to get the object of RequestDispatcher

The getRequestDispatcher() method of ServletRequest interface returns the object of RequestDispatcher.

Syntax of getRequestDispatcher method : public RequestDispatcher getRequestDispatcher(String resource);

Example of using getRequestDispatcher method:

RequestDispatcher rd=request.getRequestDispatcher("servlet2");

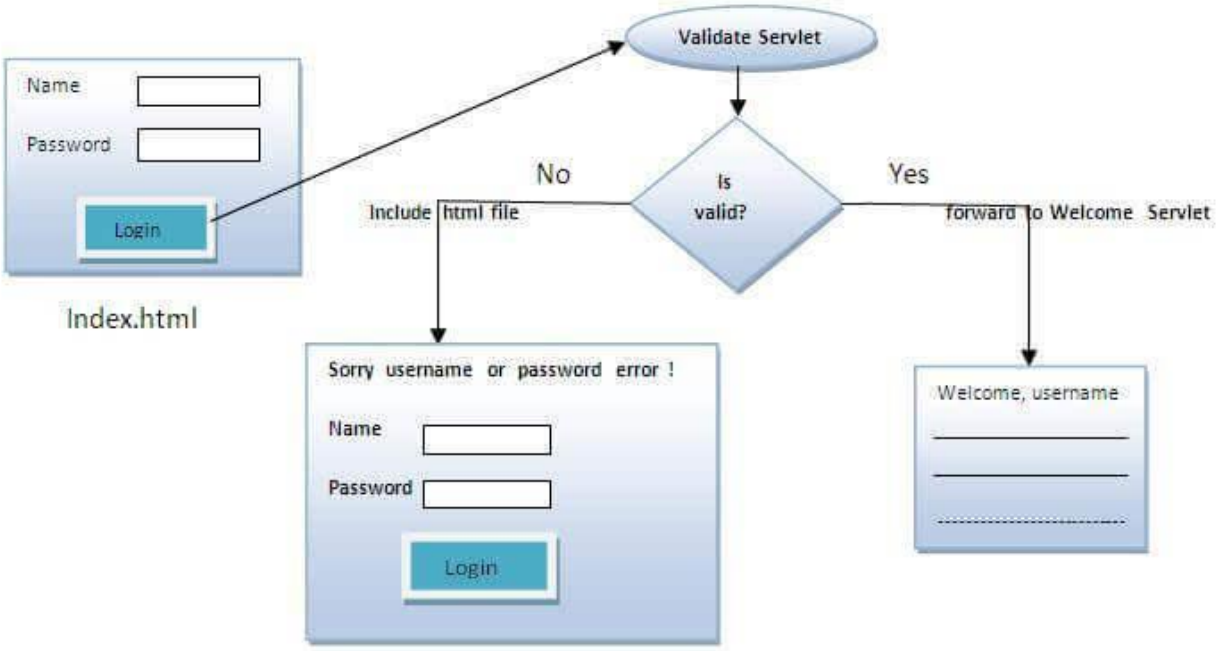
//servlet2 is the url-pattern of the second servlet

rd.forward(request, response);//method may be include or forward

Example of RequestDispatcher interface

In this example, we are validating the password entered by the user. If password is servlet, it will forward the request to the WelcomeServlet, otherwise will show an error message: sorry username or password error!. In this program, we are cheking for hardcoded information. But you can check it to the database also that we will see in the development chapter. In this example, we have created following files:

- **index.html file:** for getting input from the user.
- **Login.java file:** a servlet class for processing the response. If password is servet, it will forward the request to the welcome servlet.
- **WelcomeServlet.java file:** a servlet class for displaying the welcome message.
- **web.xml file:** a deployment descriptor file that contains the information about the servlet.



index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
Password:<input type="password" name="userPass"/><br/>
<input type="submit" value="login"/>
</form>
```

Login.java

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


public class Login extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

  response.setContentType("text/html");
  PrintWriter out = response.getWriter();

  String n=request.getParameter("userName");
  String p=request.getParameter("userPass");

  if(p.equals("servlet"){
    RequestDispatcher rd=request.getRequestDispatcher("servlet2");
    rd.forward(request, response);
  }
  else{
    out.print("Sorry UserName or Password Error!");
    RequestDispatcher rd=request.getRequestDispatcher("/index.html");
    rd.include(request, response);
```

```
        }
      }

    }
```

WelcomeServlet.java

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

    public class WelcomeServlet extends HttpServlet {

      public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String n=request.getParameter("userName");
      out.print("Welcome "+n);
      }

    }
```

web.xml

```
    <web-app>
     <servlet>
       <servlet-name>Login</servlet-name>
       <servlet-class>Login</servlet-class>
      </servlet>
      <servlet>
       <servlet-name>WelcomeServlet</servlet-name>
       <servlet-class>WelcomeServlet</servlet-class>
      </servlet>


      <servlet-mapping>
       <servlet-name>Login</servlet-name>
       <url-pattern>/servlet1</url-pattern>
      </servlet-mapping>
      <servlet-mapping>
       <servlet-name>WelcomeServlet</servlet-name>
       <url-pattern>/servlet2</url-pattern>
      </servlet-mapping>

      <welcome-file-list>
       <welcome-file>index.html</welcome-file>
      </welcome-file-list>
    </web-app>
```

SendRedirect in servlet

The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file. It accepts relative as well as absolute URL. It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.

forward() method	sendRedirect() method

The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
It can work within the server only.	It can be used within and outside the server.
Example: request.getRequestDispacher("servlet2").forward(request,response);	Example: response.sendRedirect("servlet2");

Syntax of sendRedirect() method : public void **sendRedirect(String URL)**throws **IOException;**

Example of sendRedirect() method: **response.sendRedirect("http://www.gmail.com");**

ServletConfig Interface

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

Advantage of ServletConfig

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

Methods of ServletConfig interface

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **public ServletContext getServletContext():**Returns an object of ServletContext.

How to get the object of ServletConfig

1. **getServletConfig() method** of Servlet interface returns the object of ServletConfig.

*Syntax of getServletConfig() method: public **ServletConfig getServletConfig();***

Example of getServletConfig() method: **ServletConfig config=getServletConfig();**

//Now we can call the methods of ServletConfig interface

Syntax to provide the initialization parameter for a servlet

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

```
<web-app>
  <servlet>
    ......

    <init-param>
      <param-name>parametername</param-name>
      <param-value>parametervalue</param-value>
    </init-param>
    ......
  </servlet>
</web-app>
```

Example of ServletConfig to get initialization parameter

In this example, we are getting the one initialization parameter from the web.xml file and printing this information in the servlet.

DemoServlet.java

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

public class DemoServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {

   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
```

```
        ServletConfig config=getServletConfig();
        String driver=config.getInitParameter("driver");
        out.print("Driver is: "+driver);

        out.close();
        }

    }
```

web.xml

```
    <web-app>

    <servlet>
    <servlet-name>DemoServlet</servlet-name>
    <servlet-class>DemoServlet</servlet-class>

    <init-param>
    <param-name>driver</param-name>
    <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
    </init-param>

    </servlet>

    <servlet-mapping>
    <servlet-name>DemoServlet</servlet-name>
    <url-pattern>/servlet1</url-pattern>
    </servlet-mapping>

    </web-app>
```

ServletContext Interface

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

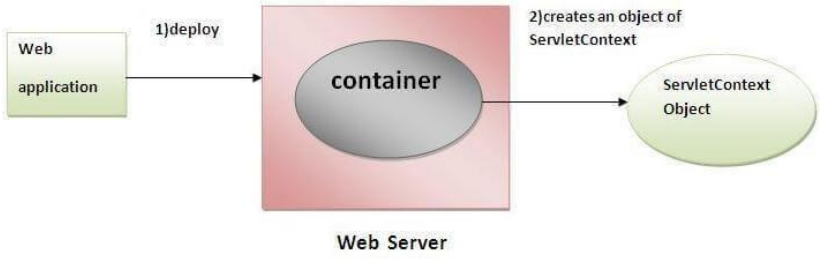
Advantage of ServletContext

Easy to maintain if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

Usage of ServletContext Interface

There can be a lot of usage of ServletContext object. Some of them are as follows:

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.



Commonly used methods of ServletContext interface

There is given some commonly used methods of ServletContext interface.

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.

2. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters.
3. **public void setAttribute(String name,Object object):**sets the given object in the application scope.
4. **public Object getAttribute(String name):**Returns the attribute for the specified name.
5. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.
6. **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context.

How to get the object of ServletContext interface

1. **getServletContext() method** of ServletConfig interface returns the object of ServletContext.
2. **getServletContext() method** of GenericServlet class returns the object of ServletContext.

Syntax of getServletContext() method: *public* ***ServletContext getServletContext()***

Example of getServletContext() method:

```
//We can get the ServletContext object from ServletConfig object
ServletContext application=getServletConfig().getServletContext();

//Another convenient way to get the ServletContext object
ServletContext application=getServletContext();
```

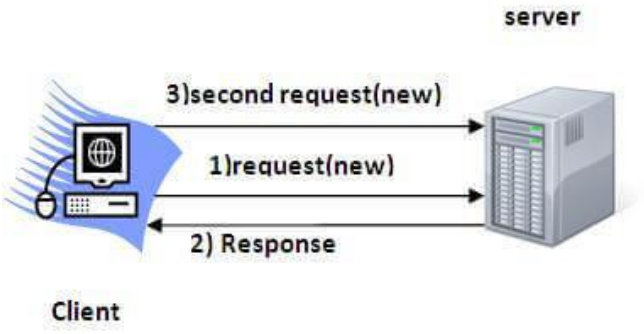
Session Tracking in Servlets

Session simply means a particular interval of time.

Session Tracking is a way to maintain state (data) of an user. It is also known as **session management** in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



Why use Session Tracking?

It is used to recognize the particular user.

Session Tracking Techniques

There are four techniques used in Session tracking:

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

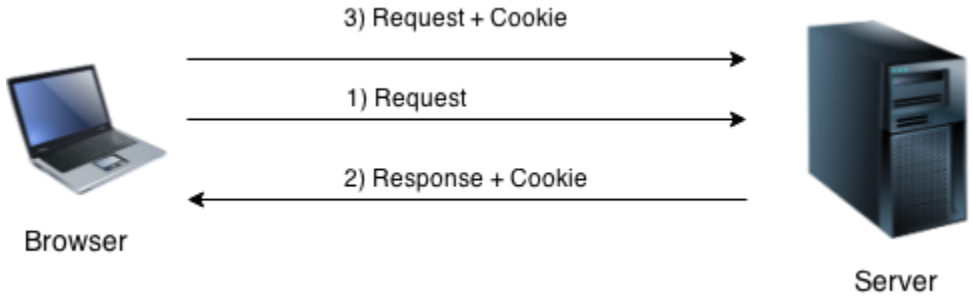
Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

It is **valid for single session** only. It is removed each time when user closes the browser.

Persistent cookie

It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Cookie class

javax.servlet.http.Cookie Class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.

public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.

How to create Cookie?

```
Cookie ck=new Cookie("user","raju");//creating cookie object

response.addCookie(ck);//adding cookie in the response
```

How to delete Cookie?

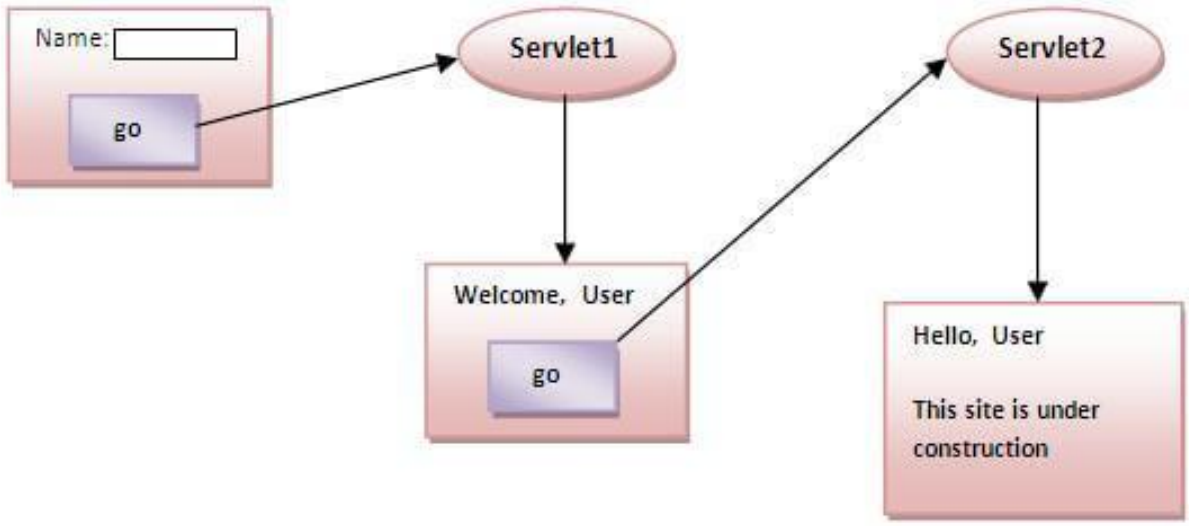
```
Cookie ck=new Cookie("user","");//deleting value of cookie

ck.setMaxAge(0);//changing the maximum age to 0 seconds

response.addCookie(ck);//adding cookie in the response
```

How to get Cookies?

```
Cookie ck[]=request.getCookies();

for(int i=0;i<ck.length;i++){

 out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie

}
```

Simple example of Servlet Cookies

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

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


public class FirstServlet extends HttpServlet {

 public void doPost(HttpServletRequest request, HttpServletResponse response)
{
  try{

  response.setContentType("text/html");
  PrintWriter out = response.getWriter();

  String n=request.getParameter("userName");
  out.print("Welcome "+n);

  Cookie ck=new Cookie("uname",n);//creating cookie object
  response.addCookie(ck);//adding cookie in the response

  //creating submit button
  out.print("<form action='servlet2'>");
  out.print("<input type='submit' value='go'>");
  out.print("</form>");

  out.close();
      }
```

```
catch(Exception e){System.out.println(e);}
 } }
```

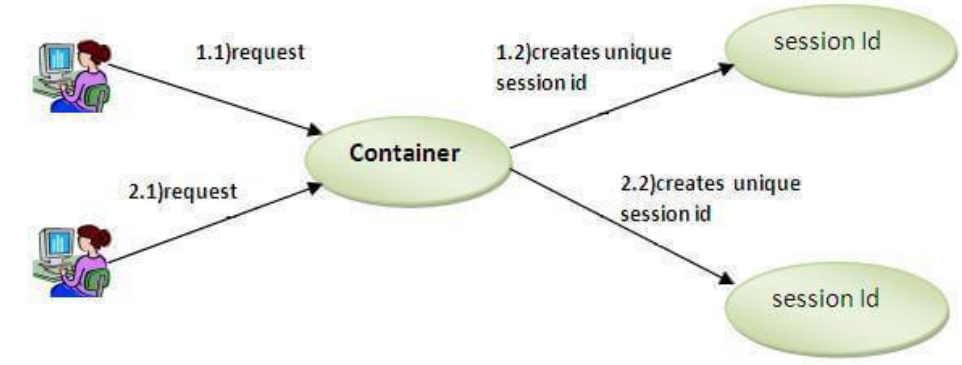
SecondServlet.java

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

public class SecondServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response){
  try{

  response.setContentType("text/html");
  PrintWriter out = response.getWriter();

  Cookie ck[]=request.getCookies();
  out.print("Hello "+ck[0].getValue());

  out.close();

    }catch(Exception e){System.out.println(e);}
  }
}
```

HttpSession interface

In such case, container creates a session id for each user. The container uses this id to identify the particular user. An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



How to get the HttpSession object ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

Example of using HttpSession

In this example, we are setting the attribute in the session scope in one servlet and getting that value from the session scope in another servlet. To set the attribute in the session scope, we have used the setAttribute() method of HttpSession interface and to get the attribute, we have used the getAttribute method.

index.html

```
<form action="servlet1">

Name:<input type="text" name="userName"/><br/>

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

</form>
```

FirstServlet.java

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

public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
    try{
      response.setContentType("text/html");
    PrintWriter out = response.getWriter();
```

```
String n=request.getParameter("userName");
out.print("Welcome "+n);

HttpSession session=request.getSession();
session.setAttribute("uname",n);

out.print("<a href='servlet2'>visit</a>");

out.close();
```

```
        }catch(Exception e){System.out.println(e);}
    }
}
```

SecondServlet.java

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

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
    try{

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    HttpSession session=request.getSession(false);
     String n=(String)session.getAttribute("uname");
     out.print("Hello "+n);

    out.close();

        }catch(Exception e){System.out.println(e);}
  }


}
```

JDBC (Java Database Connectivity):

JDBC is a Java API to connect and execute the query with the database.

It is a part of JavaSE (Java Standard Edition).

JDBC API uses JDBC drivers to connect with the database.

The java.sql package contains classes and interfaces for JDBC API(Application programming interface).

API is a document that contains a description of all the features of a product or software.

It represents classes and interfaces that software programs can follow to communicate with each other.

An API can be created for applications, libraries, operating systems, etc.

We can use JDBC API to handle database using Java program and can perform the following activities:

Step 1: Connect to the database

Step 2: Execute queries and update statements to the database

Step 3: Retrieve the result received from the database.

JDBC Driver is a software component that enables java application to interact with the database.

There are 4 types of JDBC drivers:

1) JDBC-ODBC Bridge Driver

2) Native Driver

3) Network Protocol Driver

4) Thin Driver

There are 5 steps to connect any java application with the database using JDBC.

These steps are as follows:

1. Register the Driver class

2. Create connection

3. Create statement

4. Execute queries

5.Close connection

Step 1: Register the driver class: The forName() is a static method of Class class is used to register the driver class. The forName() is a static method of the java. lang. Class

This method is used to dynamically load the driver class.

Syntax: public static void forName(String className) throws ClassNotFoundException

Here, Java program is loading mysql driver to establish database connection.

Class.forName("com.mysql.jdbc.Driver");

Note: Automatic Loading of Driver class You don't need to write Class.forName() now because it is loaded by default since jdbc4.

The JDBC Drivers (String) will be loaded into the class dynamically at run time and forName method contains static block which creates the Driver class object and register with the DriverManager Service automatically.

It initialize the class "com. mysql. jdbc. Driver" if found in the classpath, this imply that the driver is registered in the JDBC driver manager since the registration process is inside the static initializer of the driver class.

Step 2: Create the connection object:

The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax: public static Connection getConnection(String url,String name,String password) throws SQLException

```
String username="root",password="mysql";
String url="jdbc:mysql://localhost:3306/mrec";
Connection con=DriverManager.getConnection(url,username,password);
```

Driver. Connection URL: The connection URL for the mysql database is jdbc:mysql://localhost:3306/mrec where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and mrec is the database name.

Step 3: Create the Statement object

The createStatement() method of Connection interface is used to create statement.

The object of statement is responsible to execute queries with the database.

Syntax : public Statement createStatement()throws SQLException

Example : Statement stmt=con.createStatement();

Connection interface represents a session between java application and database. All SQL statements are executed and results are returned with in the context of a Connection object. Connection interface is mainly used to create java applications.

Step 4: Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database.

This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax : public ResultSet executeQuery(String sql)throws SQLException

Example: ResultSet rs=stmt.executeQuery("select * from emp");

```
    while(rs.next()){ System.out.println(rs.getInt(1)+" "+rs.getString(2)); }
```

Step 5: Close the connection object

By closing connection object statement and ResultSet will be closed automatically.
The close() method of Connection interface is used to close the connection.

Syntax : public void close()throws SQLException
Example: con.close();

Note: Since Java 7, JDBC has ability to use try-with-resources statement to automatically close resources of type Connection, ResultSet, and Statement.
It avoids explicit connection closing step.

DriverManager class : It acts as an interface between user and drivers.
It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.

This class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Connection interface :A Connection is the session between java application and database.
The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData.

The Connection interface provide many methods for transaction management like commit(), rollback() etc.
By default, connection commits the changes after executing queries.

Commonly used methods of Connection interface:
public Statement createStatement(): creates a statement object that can be used to execute SQL queries.
public void setAutoCommit(boolean status): is used to set the commit status.By default it is true.
public void commit(): saves the changes made since the previous commit/rollback permanent.
public void rollback(): Drops all changes made since the previous commit/rollback.
public void close(): closes the connection and Releases a JDBC resources immediately.

Statement interface : It interface provides methods to execute queries with the database.
It is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Commonly used methods of Statement interface:
public ResultSet executeQuery(String sql):is used to execute SELECT query. It returns the object of ResultSet.
public int executeUpdate(String sql):is used to execute specified query,
it may be create, drop, insert, update, delete etc.

ResultSet interface : The object of ResultSet maintains a cursor pointing to a row of a table.
Initially, cursor points to before the first row.
By default, ResultSet object can be moved forward only and it is not updatable.

But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatement method.
Example: Statement stmt =
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_UPDATABLE);

Commonly used methods of ResultSet interface:
public boolean next(): is used to move the cursor to the one row next from the current position.
public boolean previous():is used to move the cursor to the one row previous from the current position.
public boolean first(): is used to move the cursor to the first row in result set object.
public boolean last(): is used to move the cursor to the last row in result set object.
public String getString(int columnIndex): is used to return the data of specified column index of the current row as String.
public String getString(String columnName):is used to return the data of specified column name of the current row as String.

Example: rs.absolute(3); //getting the record of 3rd row
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));

PreparedStatement interface: The PreparedStatement interface is a subinterface of Statement.
It is used to execute parameterized query.

Example: String sql="insert into emp values(?,?,?)";
As you can see, we are passing parameter (?) for the values.
Its value will be set by calling the setter methods of PreparedStatement.

Why use PreparedStatement: (Improves performance) The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

How to get the instance of PreparedStatement?
The prepareStatement() method of Connection interface is used to return the object of PreparedStatement.
Syntax: public PreparedStatement prepareStatement(String query)throws SQLException{}

Commonly used methods of PreparedStatement interface:
public void setInt(int paramIndex, int value) sets the integer value to the given parameter index.
public void setString(int paramIndex, String value) sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value) sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value) sets the double value to the given parameter index.
public int executeUpdate() executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery() executes the select query. It returns an instance of ResultSet.

Example of PreparedStatement interface that inserts the record:

```
PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
stmt.setInt(1,101);                 //1 specifies the first parameter in the query
stmt.setString(2,"Ratan"); //2 specifies the second parameter in the query
int i=stmt.executeUpdate();  System.out.println(i+" records inserted");
```

Example of PreparedStatement interface that updates the record

```
PreparedStatement stmt=con.prepareStatement("update emp set name=? where id=?");
stmt.setString(1,"Sonoo");//1 specifies the first parameter in the query i.e. name
stmt.setInt(2,101);  //2 specifies the second parameter in the query
int i=stmt.executeUpdate(); System.out.println(i+" records updated");
```

Example of PreparedStatement interface that deletes the record

```
PreparedStatement stmt=con.prepareStatement("delete from emp where id=?");
stmt.setInt(1,101);
int i=stmt.executeUpdate(); System.out.println(i+" records deleted");
```

Example of PreparedStatement interface that retrieve the records of a table

```
PreparedStatement stmt=con.prepareStatement("select * from emp");
ResultSet rs=stmt.executeQuery();
while(rs.next()){  System.out.println(rs.getInt(1)+" "+rs.getString(2)); }
```

Batch Processing in JDBC : Instead of executing a single query, we can execute a batch (group) of queries.
The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing.
It makes the performance fast.

Methods of Statement interface: The required methods for batch processing are given below:

```
void addBatch(String query): It adds query into batch
int[] executeBatch():        It executes the batch of queries
```

Example of batch processing in jdbc :It follows six steps: Load the driver class, Create Connection, Create Statement, Add query in the batch, Execute Batch, Close Connection.

Example:

```
Class.forName("com.mysql.jdbc.Driver");  //Load the driver class
Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/ajay","root","mysql");
```

```
Statement stmt=con.createStatement(); // Create Statement
  stmt.addBatch("insert into user420 values(190,'abhi',40000)");  //add query in the batch
stmt.addBatch("insert into user420 values(191,'umesh',50000)");  //add query in the batch
stmt.executeBatch();//executing the batch
con.close();  // close the connection
```

1. Write a JDBC Program to create Table in the Mysql Database

```
import java.sql.*;
public class CreateTable {
    public static void main(String[] args) throws Exception {

            String username="root";
            String password="mysql";
            String url="jdbc:mysql://localhost:3306/ajay";

            Class.forName("com.mysql.jdbc.Driver");
            Connection con=DriverManager.getConnection(url,username,password);
            System.out.println("Connection is created");

            String sql="create table student5"+"(sid varchar(20),"+ "sname varchar(20),"+"saddr
varchar(20))";
            Statement st=con.createStatement();
            st.executeUpdate(sql);

            System.out.println("Student Table is created...");
            st.close();
            con.close();
    }
}
```

2. Write a JDBC Program to Insert the rows in the Table

```
import java.sql.*;
```

```
public class InsertData {
	public static void main(String[] args) throws Exception {
		String username="root";
		String password="mysql";
		String url="jdbc:mysql://localhost:3306/ajay";

		Class.forName("com.mysql.jdbc.Driver");
		Connection con=DriverManager.getConnection(url,username,password);
		System.out.println("Connection is created");

		Statement st=con.createStatement();
		String sql="insert into student5 values('1001','Ajay','Hyd')";
		st.executeUpdate(sql);
		sql="insert into student5 values('1002','Vijay','Armoor')";
		st.executeUpdate(sql);
		sql="insert into student5 values('1003','Nithin','Bangalore')";
		st.executeUpdate(sql);

		System.out.println("Student three records are inserted...");
		st.close();
		con.close();
	}
}
```

3. **Write a JDBC Program to update the rows in the Table**

```
import java.sql.*;

public class UpdateData {
	public static void main(String[] args) throws Exception {
		String username="root";
```

```
        String password="mysql";
        String url="jdbc:mysql://localhost:3306/ajay";

        Class.forName("com.mysql.jdbc.Driver");
        Connection con=DriverManager.getConnection(url,username,password);
        System.out.println("Connection is created");

        Statement st=con.createStatement();
        String sql="update student set saddr='Hyderabad' where sid=1001";
        st.executeUpdate(sql);

        System.out.println("Student one record is updated...");
        st.close();
        con.close();
    }

}
```

4. **Write a JDBC Program to Delete the rows in the Table**

```
import java.sql.*;
public class DeleteData {
        public static void main(String[] args) throws Exception {
                String username="root";
                String password="mysql";
                String url="jdbc:mysql://localhost:3306/ajay";

                Class.forName("com.mysql.jdbc.Driver");
                Connection con=DriverManager.getConnection(url,username,password);
                System.out.println("Connection is created");

                Statement st=con.createStatement();
```

```
                String sql="delete from student where sid=1003";
                st.executeUpdate(sql);

                System.out.println("Student one record is deleted...");
                st.close();
                con.close();
        }

    }
```

5. **Write a JDBC Program to display the rows in the Table**

```
import java.sql.*;
public class SelectData {
    public static void main(String[] args) throws Exception {

        String username="root";
        String password="mysql";
        String url="jdbc:mysql://localhost:3306/ajay";

        Class.forName("com.mysql.jdbc.Driver");
        Connection con=DriverManager.getConnection(url,username,password);
        System.out.println("Connection is created");

        Statement st=con.createStatement();
        ResultSet rs=st.executeQuery("select *from student");
        System.out.println("All the Employee Details\n");

        while(rs.next()) {
            System.out.println("Student ID: "+rs.getString(1));
            System.out.println("Student Name: "+rs.getString(2));
            System.out.println("Student Address"+rs.getString(3));
```

```
                }

                st.close();
                con.close();
        }
}
```