



## Java Platform, Enterprise Edition

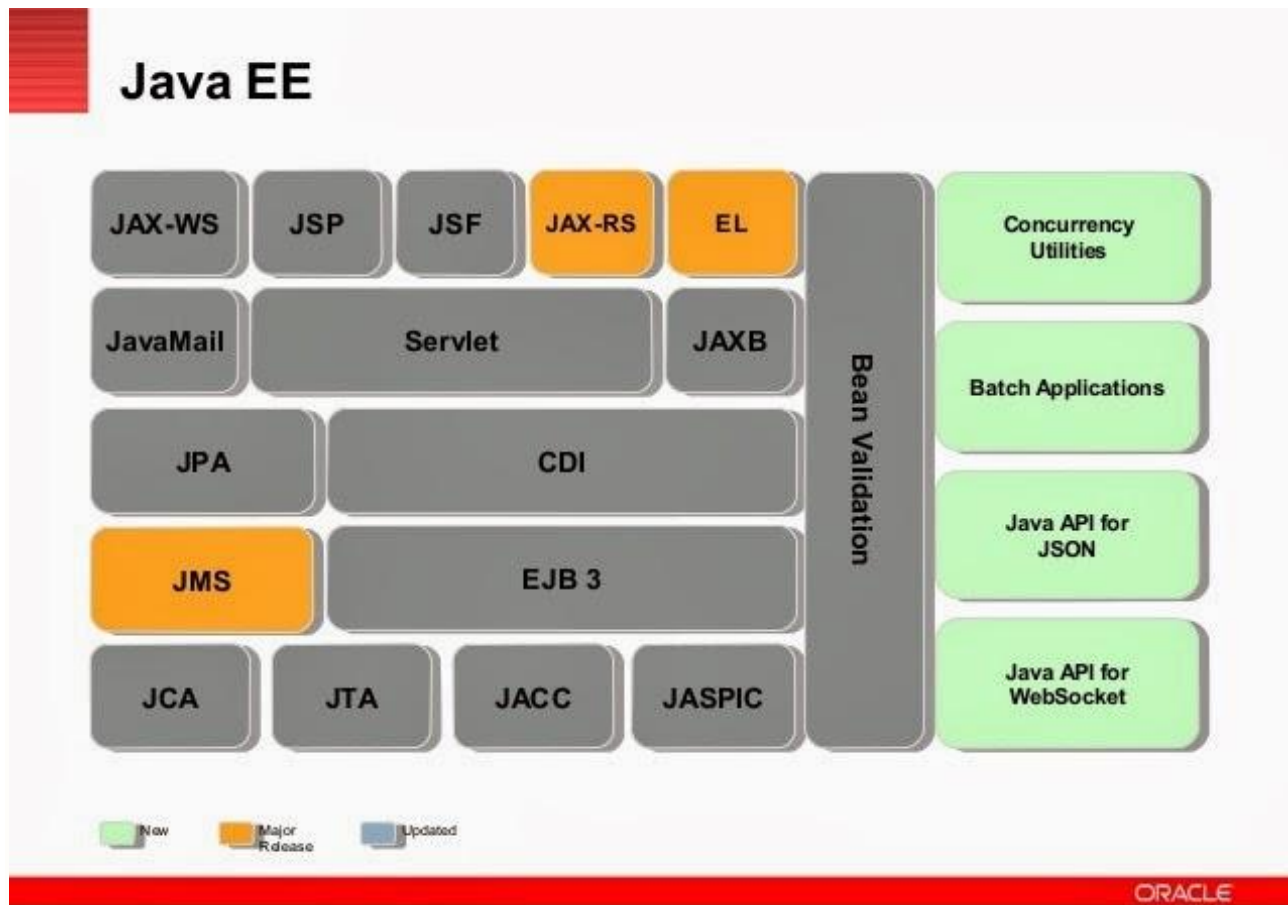
---

Java EE extends the Java Platform, Standard Edition (Java SE), providing an API for object-relational mapping, distributed and multi-tier architectures, and web services.

**Java Platform, Enterprise Edition** or **Java EE** is a widely used enterprise computing [platform](#) developed under the [Java Community Process](#). The platform provides an [API](#) and runtime environment for developing and running [enterprise software](#), including [network](#) and [web services](#), and other large-scale, multi-tiered, scalable, reliable, and secure network applications. Java EE extends the [Java Platform, Standard Edition](#) (Java SE),<sup>[1]</sup> providing an API for [object-relational mapping](#), [distributed](#) and [multi-tier architectures](#), and [web services](#).

Software for Java EE is primarily developed in the [Java](#) programming language. The platform emphasizes [convention over configuration](#) and [annotations](#) for configuration. Optionally [XML](#) can be used to override annotations or to deviate from the platform defaults.

Java EE (Java Enterprise Edition) is an Enterprise java computing platform of Oracle which provides different API like JPA (Java Persistence API), EJB (Enterprise Java Beans), CDI (Context and Dependency injection), Servlet, JSP, JSF and Runtime environment for development and running of Enterprise application software, secure reliable network applications, java web applications development, java web services, mobile applications development and much more. Java EE focused on software developer productivity and numerous enhancements in different features.



## Top 10 Features in Java EE 7

- WebSocket client/server endpoints
- Batch Applications
- JSON Processing
- Concurrency Utilities
- Simplified JMS API
- `@Transactional` and `@TransactionScoped`
- JAX-RS Client API

- Default Resources
- More annotated POJOs
- Faces Flow

## Standards and specifications

---

Java EE is defined by its [specification](#). As with other [Java Community Process](#) specifications, providers must meet certain conformance requirements in order to declare their products as *Java EE compliant*.

Java EE includes several [API](#) specifications, such as [RMI](#), [e-mail](#), [JMS](#), [web services](#), [XML](#), etc., and defines how to coordinate them. Java EE also features some specifications unique to Java EE for components. These include [Enterprise JavaBeans](#), [connectors](#), [servlets](#), [JavaServer Pages](#) and several [web service](#) technologies. This allows developers to create [enterprise applications](#) that are [portable](#) and [scalable](#), and that integrate with legacy technologies. A Java EE [application server](#) can handle transactions, security, scalability, [concurrency](#) and management of the components it is deploying, in order to enable developers to concentrate more on the business logic of the components rather than on infrastructure and integration tasks.

## General APIs

---

The Java EE APIs includes several technologies that extend the functionality of the base [Java SE APIs](#).

- [Java EE 7 Platform Packages](#)
- [Java EE 6 Platform Packages](#)
- [Java EE 5 Platform Packages](#)

```
javax.servlet.*
```

The [servlet](#) specification defines a set of APIs to service mainly [HTTP](#) requests. It includes the [JavaServer Pages](#) (JSP) specification.

```
javax.websocket.*
```

The Java API for WebSocket specification defines a set of APIs to service [WebSocket](#) connections.

```
javax.faces.*
```

This package defines the root of the [JavaServer Faces](#) (JSF) API. JSF is a technology for constructing user interfaces out of components.

```
javax.faces.component.*
```

This package defines the component part of the JavaServer Faces API. Since JSF is primarily component oriented, this is one of the core packages. The package overview contains a UML diagram of the component hierarchy.

```
javax.el.*
```

This package defines the classes and interfaces for Java EE's [Expression Language](#). The Expression Language (*EL*) is a simple language originally designed to satisfy the specific needs of web application developers. It is used specifically in JSF to bind components to (backing) beans and in CDI to name beans, but can be used throughout the entire platform.

```
javax.enterprise.inject.*
```

These packages define the injection annotations for the [Contexts and Dependency Injection \(CDI\)](#) APIs.

```
javax.enterprise.context.*
```

These packages define the context annotations and interfaces for the [Contexts and Dependency Injection \(CDI\)](#) API.

```
javax.ejb.*
```

The [Enterprise JavaBean \(EJB\)](#) specification defines a set of lightweight APIs that an object container (the EJB container) will support in order to provide [transactions](#) (using [JTA](#)), [remote procedure calls](#) (using [RMI](#) or [RMI-IIOP](#)), [concurrency control](#), [dependency injection](#) and [access control](#) for business objects. This package contains the Enterprise JavaBeans classes and interfaces that define the contracts between the enterprise bean and its clients and between the enterprise bean and the ejb container.

```
javax.validation.*
```

This package contains the annotations and interfaces for the declarative validation support offered by the [Bean Validation](#) API. Bean Validation provides a unified way to provide constraints on beans (e.g. JPA model classes) that can be enforced cross-layer. In Java EE, [JPA](#) honors bean validation constraints in the persistence layer, while [JSF](#) does so in the view layer.

```
javax.persistence.*
```

This package contains the contracts between a persistence provider and the managed classes and the clients of the [Java Persistence API \(JPA\)](#).

```
javax.transaction.*
```

This package provides the [Java Transaction API \(JTA\)](#) that contains the interfaces and annotations to interact with the transaction support offered by Java EE. Even though this API abstracts from the

really low-level details, the interfaces are also considered somewhat low-level and the average application developer in Java EE is either assumed to be relying on transparent handling of transactions by the higher level EJB abstractions, or using the annotations provided by this API in combination with CDI managed beans.

```
javax.security.auth.message.*
```

This package provides the core of the Java Authentication SPI (*JASPI*) that contains the interfaces and classes to build authentication modules for secure Java EE applications. Authentication modules are responsible for the interaction dialog with a user (e.g. redirecting to a Form or to an [OpenID](#) provider), verifying the user's input (e.g. by doing an LDAP lookup, database query or contacting the OpenID provider with a token) and retrieving a set of groups/roles that the authenticated user is in or has (e.g. by again doing an LDAP lookup or database query).

```
javax.enterprise.concurrent.*
```

This package provides the interfaces for interacting directly with Java EE's platform default managed thread pool. A higher-level executor service working on this same thread pool can be used optionally. The same interfaces can be used for user-defined managed thread pools, but this relies on vendor specific configuration and is not covered by the Java EE specification.

```
javax.jms.*
```

This package defines the [Java Message Service](#) (*JMS*) API. The JMS API provides a common way for Java programs to create, send, receive and read an enterprise messaging system's messages.

```
javax.batch.api.*
```

This package defines the entry AP for Java EE [Batch Applications](#). The Batch Applications API provides the means to run long running background tasks that possibly involve a large volume of data and which may need to be periodically executed.

```
javax.resource.*
```

This package defines the [Java EE Connector Architecture](#) (*JCA*) API. Java EE Connector Architecture (JCA) is a Java-based technology solution for connecting application servers and enterprise information systems (*EIS*) as part of enterprise application integration (*EAI*) solutions. This is a low-level API aimed at vendors that the average application developer typically does not come in contact with.

## Enterprise JavaBeans

---

**Enterprise JavaBeans (EJB)** is a managed, [server](#) software for modular construction of [enterprise software](#), and one of several [Java APIs](#). EJB is a [server-side software component](#) that [encapsulates](#) the [business logic](#) of an application. The EJB specification is a subset of the [Java EE](#) specification. An EJB [web container](#) provides a [runtime environment](#) for web related software components, including [computer security](#), [Java servlet lifecycle management](#), [transaction processing](#), and other [web services](#).

The EJB specification details how an [application server](#) provides the following responsibilities:

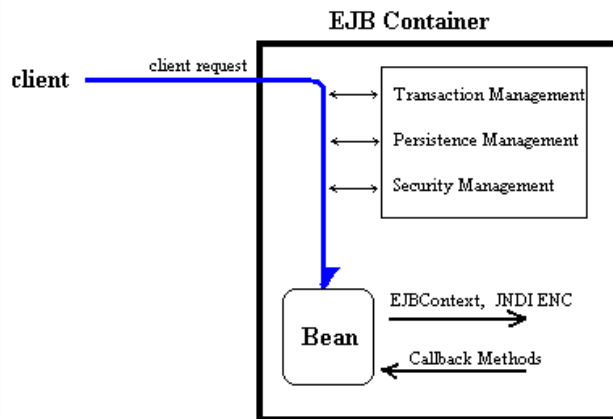
- [Transaction processing](#)
- Integration with the [persistence](#) services offered by the [Java Persistence API \(JPA\)](#)
- [Concurrency control](#)
- [Event-driven programming](#) using [Java Message Service](#) and [Java EE Connector Architecture](#)
- [Asynchronous method invocation](#)
- [Job scheduling](#)
- Naming and [directory services \(JNDI\)](#)
- [Interprocess Communication](#) using [RMI-IIOP](#) and [Web services](#)
- [Security \(JCE and JAAS\)](#)
- [Deployment of software components](#) in an application server

## What is the EJB Container

Enterprise java beans EJB are software components that run in a special environment called an EJB container. The EJB container hosts and manages an enterprise java bean in the same way as Java Web Server hosts a java servlet. All java beans functions perform with in EJB container. The EJB container manages every aspect / features of an enterprise java bean at run time.

Java beans container is a run-time container that are deployed to an Java EE application server. The EJB container automatically creates when the java application server starts and give serves as an interface between a java bean and run-time services such as:

- Life cycle management
- Code generation
- Persistence management
- Security
- Transaction management
- Locking and concurrency control



### **EJB Containers manage enterprise beans at runtime**

The EJB container isolates the Java beans from client applications to directly access. When a java client applications invoke a EJB remote method on an enterprise java bean, the EJB container first intercepts the invocation to ensure persistence, transactions, and security are applied properly to every operation a client performs on the jav bean. The java bean container automatically manages transactions, security and persistence for the java bean due to Java beans developers do not have to write this type of logic into the java bean codes. It provides the facility to Java EJB developers to focus just on encapsulating business rules while the EJB container takes care of everything else.

### **What is Enterprise Java Beans ( EJB ):**

EJB is written in Java programming language and it is server side component that encapsulates the business logic of an java application from customers. Business logic code is managed in EJB that fulfills the purpose of the java applications. For example, in an eCommerce web application, the enterprise java beans might implement the business logic in methods called `checkAvailableItem` and `orderProduct`.

### **Benefits of Enterprise Java Beans EJB**

EJB technology enables rapid and simplified the process of distributed, transactional, secure and portable java desktop applications development and Java ee web applications development because

- EJB container provides System level services to enterprise java beans.
- EJB developer just focus on business logic and on solving business problems.
- Because business logic lies in EJB, so Front end developer can focus on the presentation of client interface.
- The client developer does not have to code the routines that implement business rules or access databases. As a result, clients side has less codes which is particularly important for clients that run on small devices.
- Java Beans are portable components which enable the java application assembler to build new applications from existing java beans.

- EJB is a standard API due to which applications build on EJB can run on any compliant Java EE web application server.

## Types of Enterprise Java Beans:

There are two main types of EJBs

1- Session Beans

2- Message Driven Beans

### ***1- Session Beans***

A session bean encapsulates business logic that can be invoked programmatically by a client over local, remote, or web service client views. A session bean is not persistent, means its data is not saved to a database. EJB Session Beans has three types which are

#### ***Stateful Session Bean***

In stateful session bean, the state of an object consists of the values of its instance variables. In a stateful session bean, the instance variables represent the state of a unique client / bean session.

#### ***Stateless Session Bean***

In EJB stateless session bean conversational state is not maintained with client. When a client invokes the methods of a stateless bean, the bean's instance variables may contain a state specific to that client but only for the duration of the invocation.

#### ***Single tone Session Bean***

A singleton session bean is instantiated once per application and exists for the whole lifecycle of the java application. A single enterprise bean instance is shared across all the applications clients and it is concurrently accessed by clients

### ***2- Message Driven Bean (MDB)***

Message Driven Beans ( MDBs ) also known as Message Beans. Message Driven Beans are business objects whose execution is triggered by messages instead of by method calls.

## When to Use Enterprise Java Beans EJB

EJB can be used in any java application which are simple or complicated. Java applications can be developed with out EJB. Now answer to question " when to use EJB" is that EJB should be used if your application has any of the following requirements.

- For Scalable Java Applications: The application must be scalable. To accommodate a growing number of users, you may need to distribute an application's components across multiple machines. Not only can the enterprise beans of an application run on different machines, but also their location will remain transparent to the clients.
- For Transactional java applications: In application transactions must ensure data integrity. Enterprise java beans support transactions, the mechanisms that manage the concurrent access of shared objects.
- Java Application with different Client like Desktop, Web, Mobile etc: The java application will have a variety of clients. With only a few lines of code, remote EJB clients can easily locate enterprise java beans. These clients can be thin, various, and numerous.



# What is Java-EE Server and different types of Java EE Containers used in Java application Server

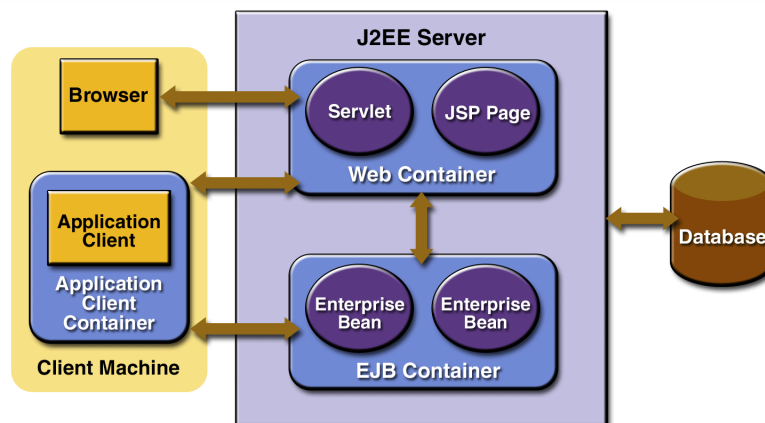
What is Java EE Server

A **Java EE server** is an application server software which implements Java EE platform APIs like Servlet, JSP, JSF, EJB etc and provides different Java EE services like SOAP, Restful etc. Java EE servers are also called application servers because they can provide application data on different clients like Java desktop clients, Web based Client on web browser, Mobile clients, applet etc and runs EJB Components also.

Below is the list of some famous java ee Servers.

- 1- Oracle GlassFish Application Server
- 2- Oracle Weblogic Application server
- 3- IBM Websephere Application server
- 4- Apache Tomcate Application Server
- 5- Jboss Application Server

Java EE servers provide hosting to several application components and used different containers to provide services to these components.



## Java EE Containers

Java EE containers act as an interface between different components like EJB, JSP, JSF, Java mail etc to provide desired functionality. The platform defined the functionality of the containers which are different for different containers. These server containers allows different components to work together to provide required functionality of Java EE applications.

There are mostly three different types of containers used in Java EE Servers which are Web Container, EJB Container and Application Client Container

### 1- The Web Container

The web container provides interface in between web server and web components which can be a servlet, a JSF page and a JSP page etc. The main responsibility of Web container is to

- 1- manages the web component's lifecycle,
- 2- dispatches requests to application components,
- 3- and provides interfaces to context data.

### 2- The Application Client Container

The Java EE Application Client Container runs on client machine and serve as gateway in between application server components and client application. A Java EE application client is a Java SE Application which use Java EE Server components to display Java Applications information and to provide different functions.

### 3- The EJB Container

EJB container runs on the Java EE server. EJB Container provide interface in between Java EE Server and enterprise java beans which provides different business logic in a Java EE Application. The main responsibility of EJB container is to manages the execution of an enterprise beans.

# Servlet Tutorial

Servlet is J2EE server driven technology to create web applications in java.

The `javax.servlet` and `javax.servlet.http` packages provide interfaces and classes for writing our own servlets. Before introduction of Servlet API, CGI technology was used to create dynamic web applications. CGI technology has many drawbacks such as creating separate process for each request, platform dependent code (C, C++), high memory usage and slow performance.

Website:

[Web pages](#), which are the [building blocks](#) of websites, are [documents](#), typically written in [plain text](#) interspersed with formatting instructions of Hypertext Markup Language ([HTML](#), [XHTML](#)). They may incorporate elements from other websites with suitable [markup anchors](#). Webpages are accessed and transported with the [Hypertext Transfer Protocol](#) (HTTP), which may optionally employ encryption ([HTTP Secure](#), HTTPS) to provide security and privacy for the user of the webpage content. The user's application, often a [web browser](#), renders the page content according to its HTML markup instructions onto a [display terminal](#). The pages of a website can usually be accessed from a simple [Uniform Resource Locator](#) (URL) called the [web address](#).

Static Website:

A static website is one that has [web pages](#) stored on the server in the format that is sent to a client web browser. It is primarily coded in [Hypertext Markup Language](#) (HTML); [Cascading Style Sheets](#) (CSS) are used to control appearance beyond basic HTML. Images are commonly used to effect the desired appearance and as part of the main content. Audio or video might also be considered "static" content if it plays automatically or is generally non-interactive.

- Static web sites can be edited using [Text editors](#), such as [Notepad](#) or [TextEdit](#), where content and HTML markup are manipulated directly within the editor program

Dynamic Website:

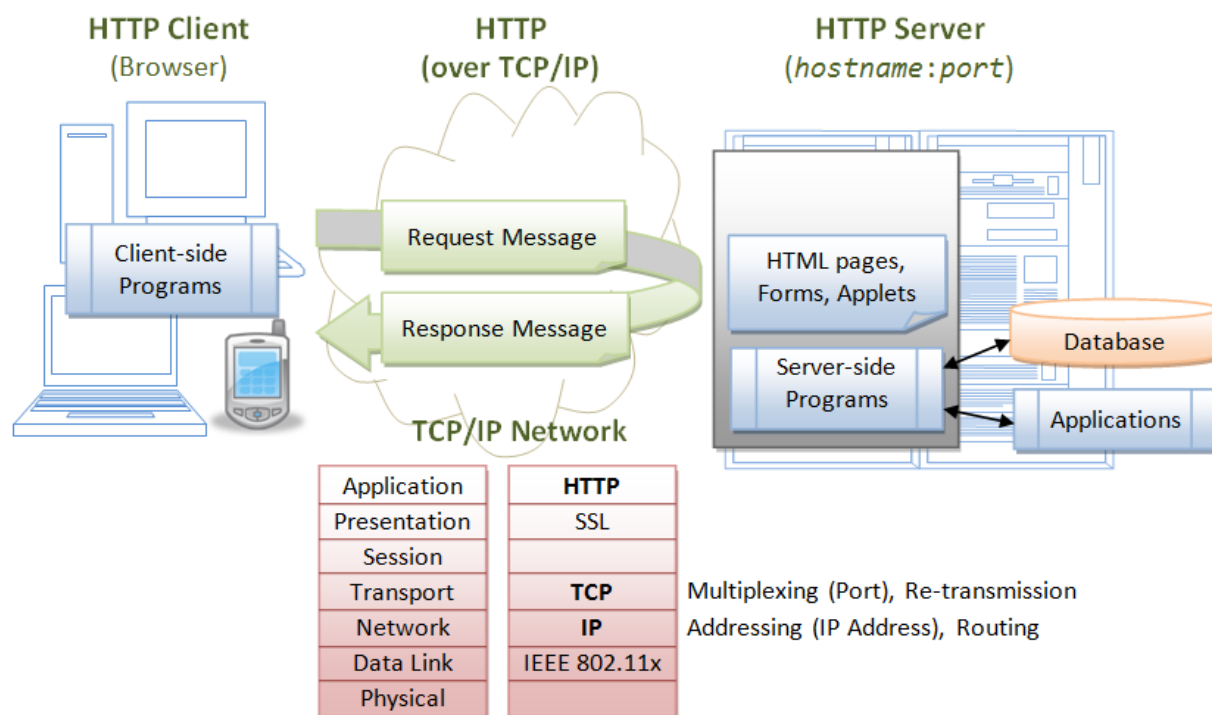
A dynamic website is one that changes or customizes itself frequently and automatically.

Server-side dynamic pages are generated "on the fly" by computer code that produces the HTML (CSS are responsible for appearance and thus, are static files). There are a wide range of software systems, such as [CGI](#), [Java Servlets](#) and [Java Server Pages](#) (JSP), [Active Server Pages](#) and [ColdFusion](#) (CFML) that are available to generate [dynamic web systems and dynamic sites](#). Various [web application frameworks](#) and [web template systems](#) are available for general-use [programming languages](#) like [JAVA](#), [PHP](#), [Perl](#), [Python](#), and [Ruby](#), to make it faster and easier to

create complex dynamic web sites. [Dynamic HTML](#) uses [JavaScript](#) code to instruct the web browser how to interactively modify the page contents.

In the early days, web servers deliver *static* contents that are indifferent to users' requests. Java servlets are *server-side programs* (running inside a web server) that handle clients' requests and return a *customized or dynamic response* for each request. The dynamic response could be based on user's input (e.g., search, online shopping, online transaction) with data retrieved from databases or other applications, or time-sensitive data (such as news and stock prices).

Java servlets typically run on the HTTP protocol. HTTP is an *asymmetrical request-response protocol*. The client sends a *request message* to the server, and the server returns a *response message* as illustrated.



## Server-Side Technologies

There are many (competing) server-side technologies available: Java-based (servlet, JSP, JSF, Struts, Spring, Hibernate), ASP, PHP, CGI Script, and many others.

Java servlet is the *foundation* of the Java server-side technology, JSP (JavaServer Pages), JSF (JavaServer Faces), Struts, Spring, Hibernate, and others, are extensions of the servlet technology.

## Apache Tomcat Server

Servlets are server-side programs run inside a *Java-capable* HTTP server. Apache Tomcat Server (@ <http://tomcat.apache.org>) is the official Reference Implementation (RI) for Java servlet and JSP, provided free by open-source foundation Apache (@ <http://www.apache.org>).

# Servlet Terminology

1. [Basics of Servlet](#)
2. [HTTP](#)
3. [Http Request Methods](#)
4. [Difference between Get and Post](#)
5. [Anatomy of Get Request](#)
6. [Anatomy of Post Request](#)
7. [Content Type](#)

There are some key points that must be known by the servlet programmer like server, container, get request, post request etc. Let's first discuss these points before starting the servlet technology.

The basic **terminology used in servlet** are given below:

1. HTTP
2. HTTP Request Types
3. Difference between Get and Post method
4. Container
5. Server and Difference between web server and application server
6. Content Type
7. Introduction of XML
8. Deployment

## HTTP

---

A HTTP Servlet runs under the HTTP protocol. It is important to understanding the HTTP protocol in order to understand server-side programs (servlet, JSP, ASP, PHP, etc) running over the HTTP. HTTP is a request-response protocol. The client sends a request message to the server. The server, in turn, returns a response message. The messages consists of two parts: header (information about the message) and body (contents). Header provides information about the messages. The data in header is organized in name-value pairs.

1. Http is the protocol that allows web servers and browsers to exchange data over the web.
2. It is a request response protocol.
3. Http uses reliable TCP connections by default on TCP port 80.
4. It is stateless means each request is considered as the new request. In other words, server doesn't recognize the user by default.

## Http Request Methods

Every request has a header that tells the status of the client. There are many request methods. Get and Post requests are mostly used.

The http request methods are:

- GET
- POST
- HEAD
- PUT
- DELETE
- OPTIONS
- TRACE

HTTP Request	Description
<b>GET</b>	Asks to get the resource at the requested URL.
<b>POST</b>	Asks the server to accept the body info attached. It is like GET request with extra info.
<b>HEAD</b>	Asks for only the header part of whatever a GET would return. Just like GET but without the body.
<b>TRACE</b>	Asks for the loopback of the request message, for testing or troubleshooting.
<b>PUT</b>	Says to put the enclosed info (the body) at the requested URL.
<b>DELETE</b>	Says to delete the resource at the requested URL.
<b>OPTIONS</b>	Asks for a list of the HTTP methods to which the thing at the request URL can respond.

## What is the difference between Get and Post?

There are many differences between the Get and Post request. Let's see these differences:

GET	POST
1) In case of Get request, only <b>limited amount of data</b> can be sent because data is sent in header.	In case of post request, <b>large amount of data</b> can be sent because data is sent in body.
2) Get request is <b>not secured</b> because data is exposed in URL bar.	Post request is <b>secured</b> because data is not exposed in URL bar.
3) Get request <b>can be bookmarked</b>	Post request <b>cannot be</b> bookmarked
4) Get request is <b>idempotent</b> . It means second request will be ignored until response of first request is delivered.	Post request is <b>non-idempotent</b>
5) Get request is <b>more efficient</b> and used more than Post	Post request is <b>less efficient</b> and used less than get.

## Anatomy of Get Request

As we know that data is sent in request header in case of get request. It is the default request type. Let's see what information's are sent to the



server.

## Anatomy of Post Request

As we know, in case of post request original data is sent in message body. Let's see how informations are passed to the server in case of post request.





## Container

It provides runtime environment for JavaEE (j2ee) applications.

It performs many operations that are given below:

1. Life Cycle Management
2. Multithreaded support
3. Object Pooling
4. Security etc.

---

## Server

It is a running program or software that provides services.

There are two types of servers:

1. Web Server
2. Application Server

---

### Web Server

Web server contains only web or servlet container. It can be used for servlet, jsp, struts, jsf etc. It can't be used for EJB.

Example of Web Servers are: **Apache Tomcat** and **Resin**.

---

### Application Server

Application server contains Web and EJB containers. It can be used for servlet, jsp, struts, jsf, ejb etc.

Example of Application Servers are:

1. **JBoss** Open-source server from JBoss community.
2. **Glassfish** provided by Sun Microsystem. Now acquired by Oracle.
3. **Weblogic** provided by Oracle. It more secured.
4. **Websphere** provided by IBM.

Nowadays almost all servers are capable as both web and application server.

## Content Type

Content Type is also known as MIME (Multipurpose internet Mail Extension) Type. It is a **HTTP header** that provides the description about what are you sending to the browser.

There are many content types:

- text/html
- text/plain
- application/msword
- application/vnd.ms-excel
- application/jar
- application/pdf
- application/octet-stream
- application/x-zip
- images/jpeg
- video/quicktime etc.

## Servlet Interface

**Servlet interface** provides common behaviour to all the servlets. Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). 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
<b>public void init(ServletConfig config)</b>	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.

<b>public void service(ServletRequest request,ServletResponse response)</b>	provides response for the incoming request. It is invoked at each request by the web container.
<b>public void destroy()</b>	is invoked only once and indicates that servlet is being destroyed.
<b>public ServletConfig getServletConfig()</b>	returns the object of ServletConfig.
<b>public String getServletInfo()</b>	returns information about servlet such as writer, copyright, version etc.

## GenericServlet:

**GenericServlet** class

implements **Servlet**, **ServletConfig** and **Serializable** interfaces. 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.

## 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

There are many methods in HttpServlet class. They are as follows:

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.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT



## 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.

Let's see what are the interfaces of `javax.servlet` package.

### 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

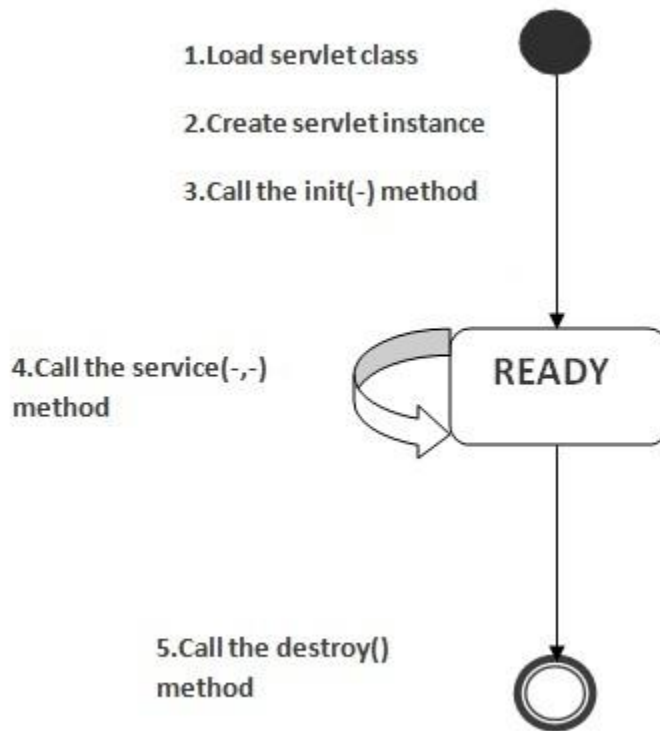
There are many classes in javax.servlet.http package. They are as follows:

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

## 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.

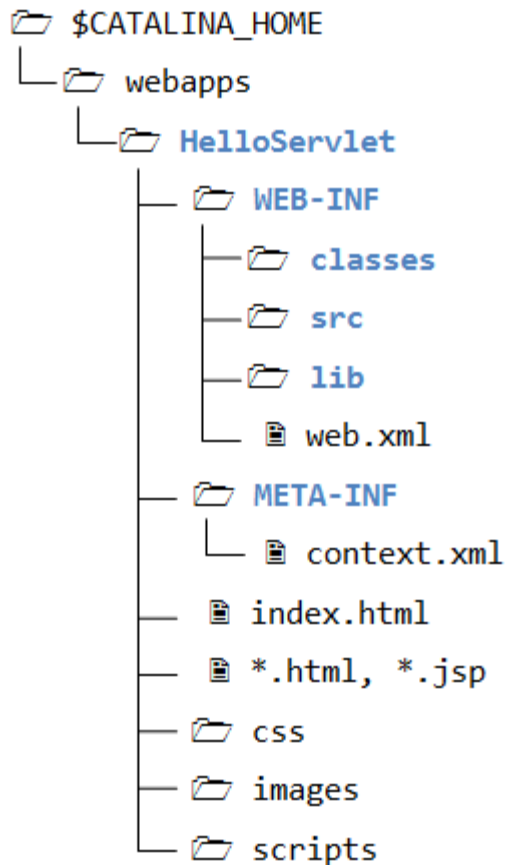
The servlet example can be created by three ways:

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.

### *Create a new Webapp "helloservlet"(Using JSP and servlet)*

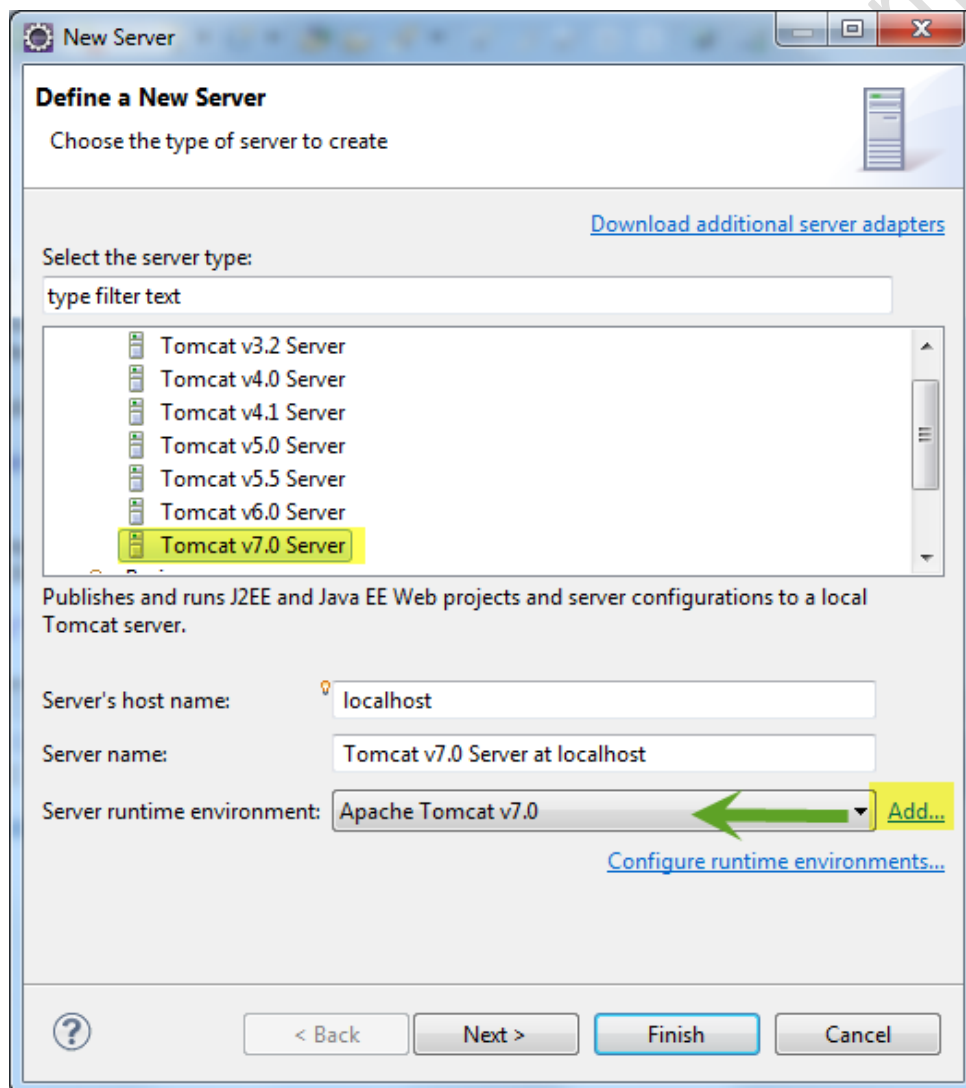
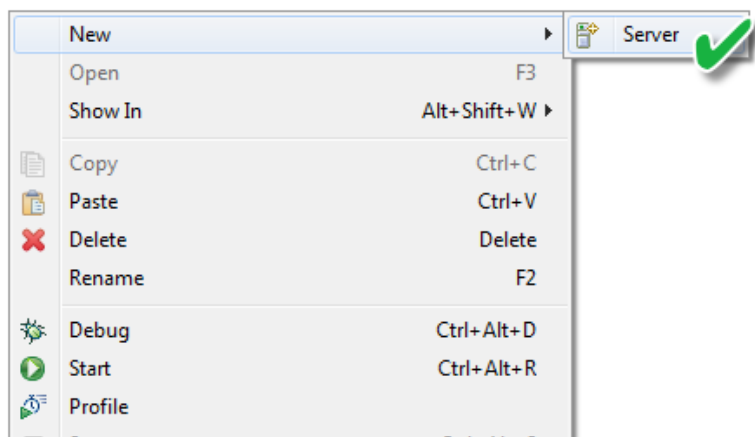
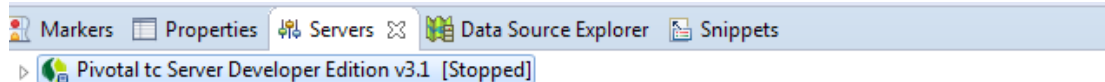
We shall begin by defining a new webapp (web application) called "helloservlet" in Tomcat. A webapp, known as a *web context* in Tomcat, comprises a set of resources, such as HTML files, CSS, JavaScripts, images, programs and libraries.

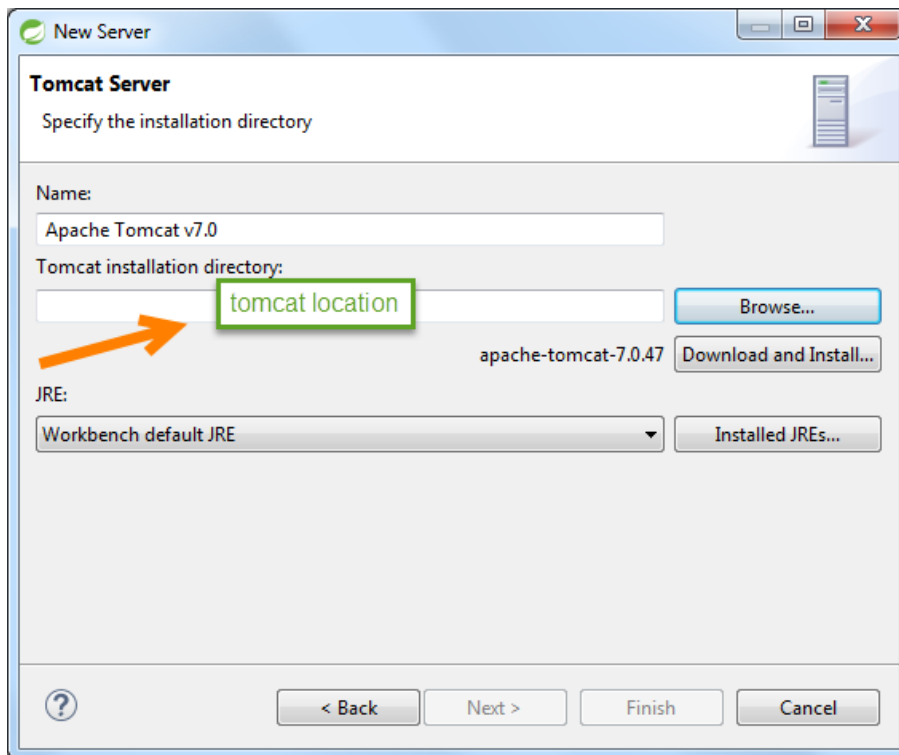


A Java webapp has a *standardized directory structure* for storing various types of resources. The Sun Microsystems defines a unique standard to be followed by all the server vendors. As you can see that the servlet class file must be in the `classes` folder. The `web.xml` file must be under the `WEB-INF` folder.

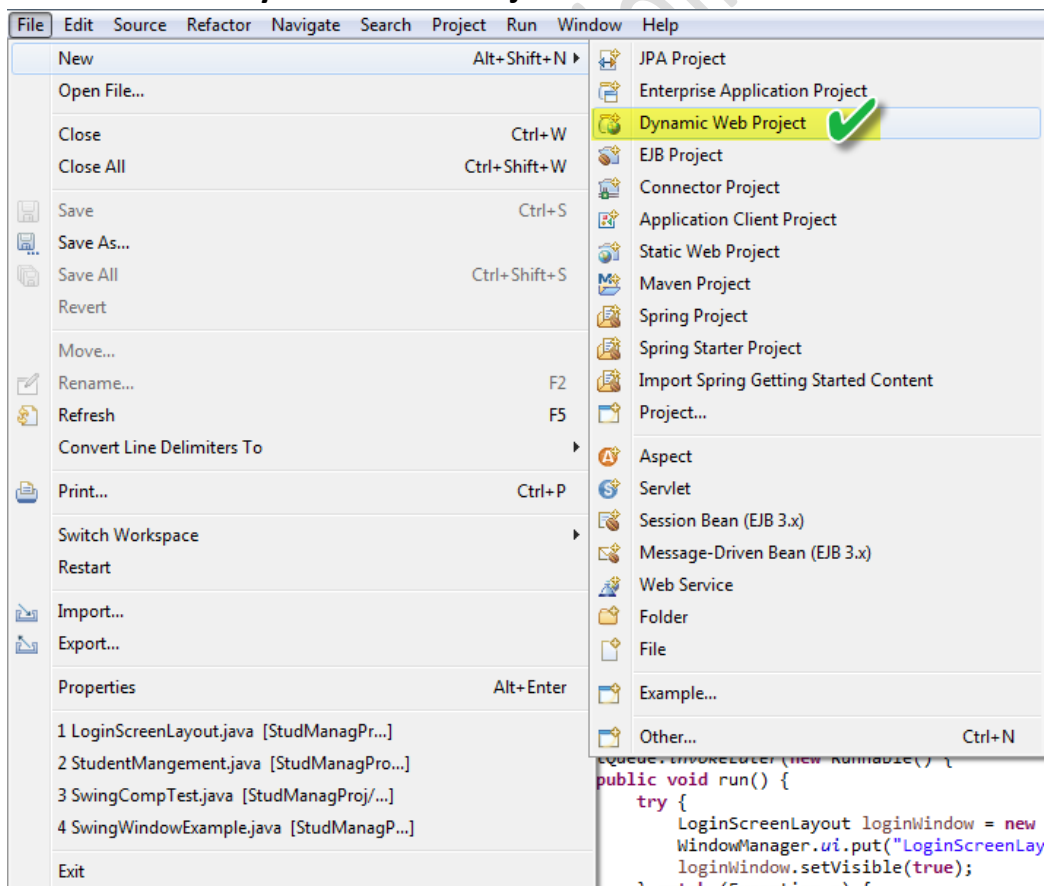
### Task 1: Add Tomcat Server in Eclipse:







## Task 2: Create Dynamic Web Project



**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: **helloervlet** **1**

Project location  
☒ Use default location  
Location: C:\YRO-DRIVE\TOOLS\_TEACH\bw-ws\helloervlet Browse...

Target runtime  
**Apache Tomcat v7.0** **2** New Runtime...

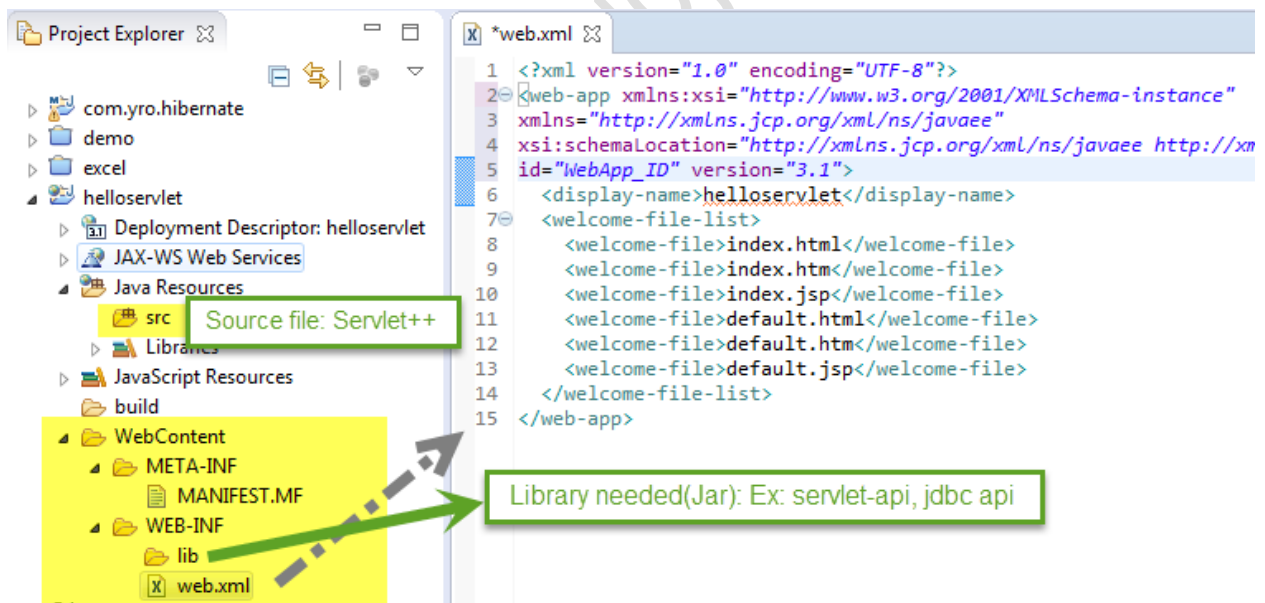
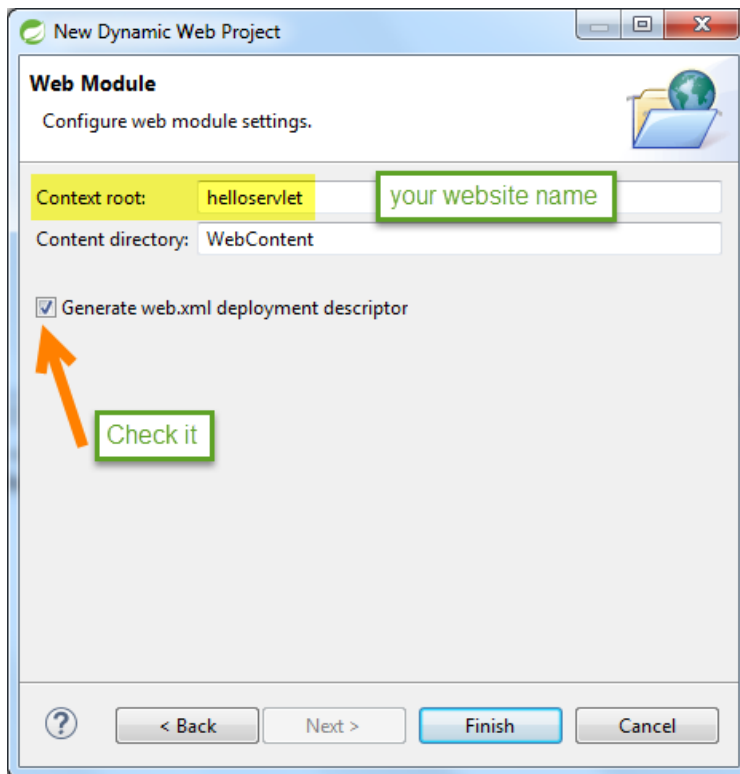
Dynamic web module version  
**2.5** **3**

Configuration  
<custom> Modify...  
Hint: Get started quickly by selecting one of the pre-defined project configurations.

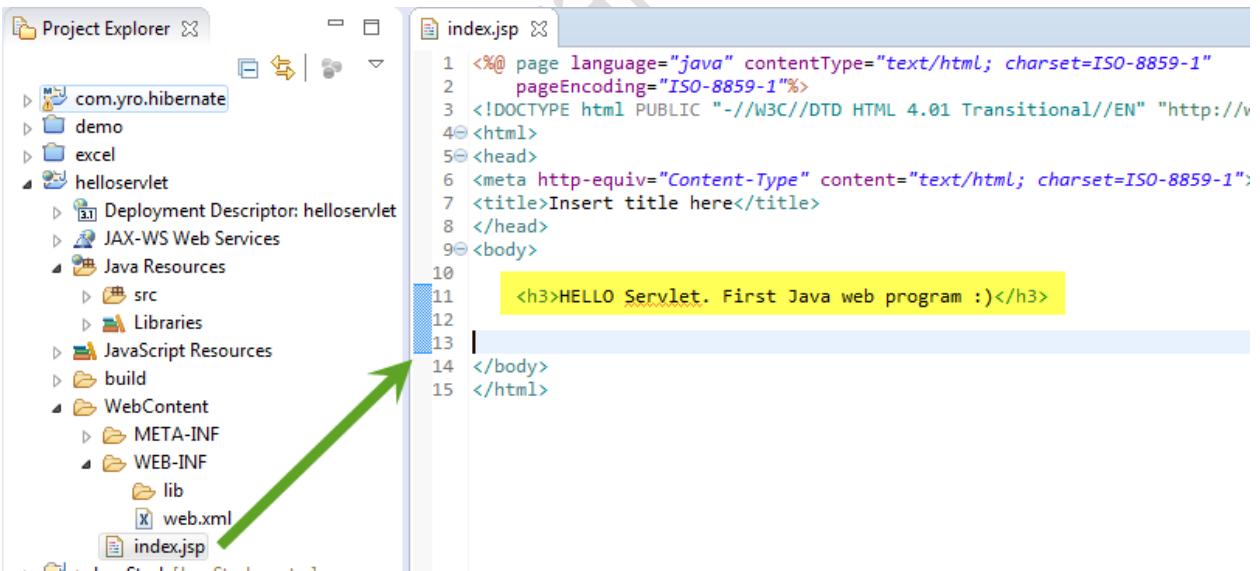
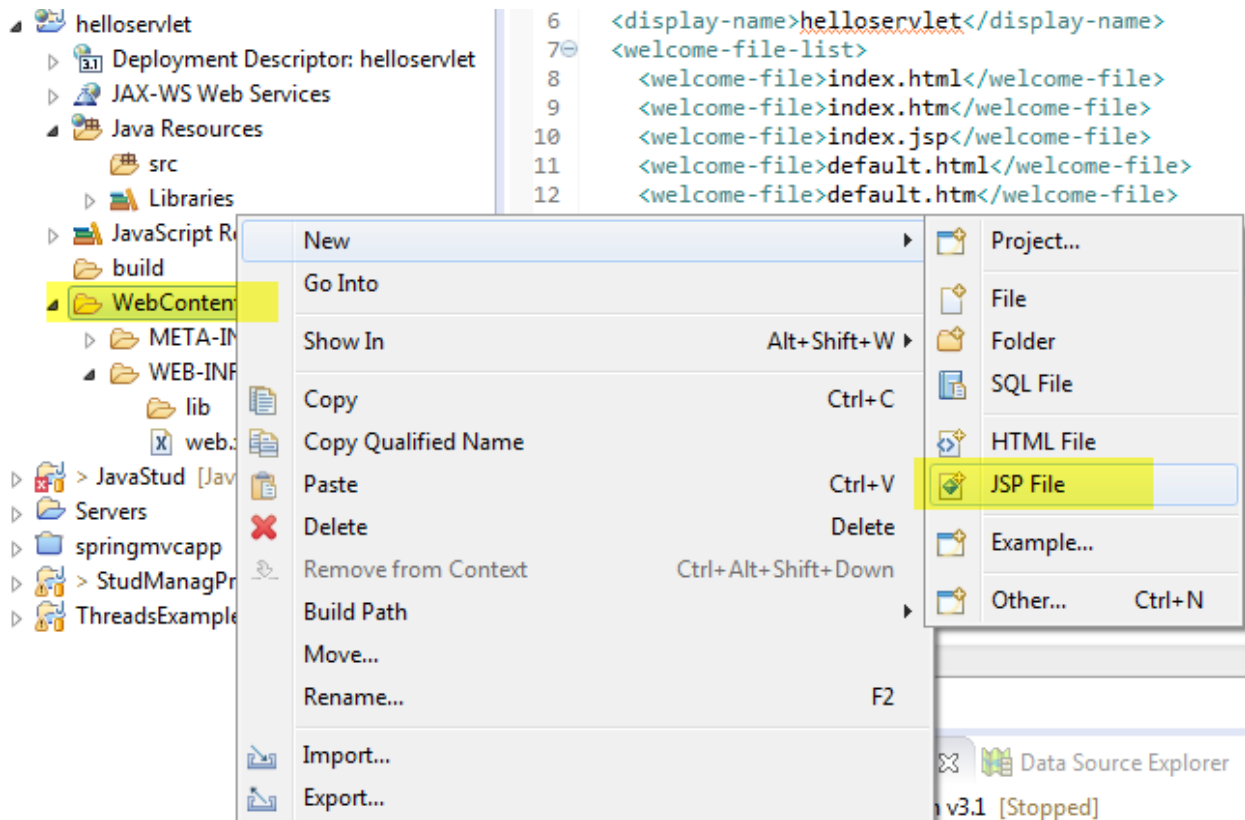
EAR membership  
☐ Add project to an EAR  
EAR project name: helloervletEAR New Project...

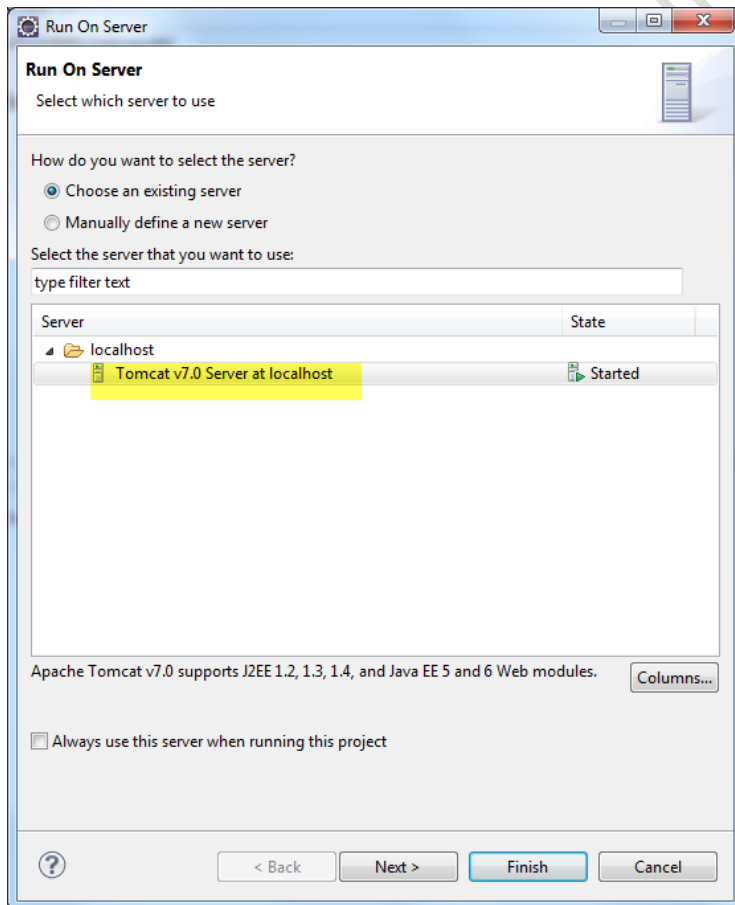
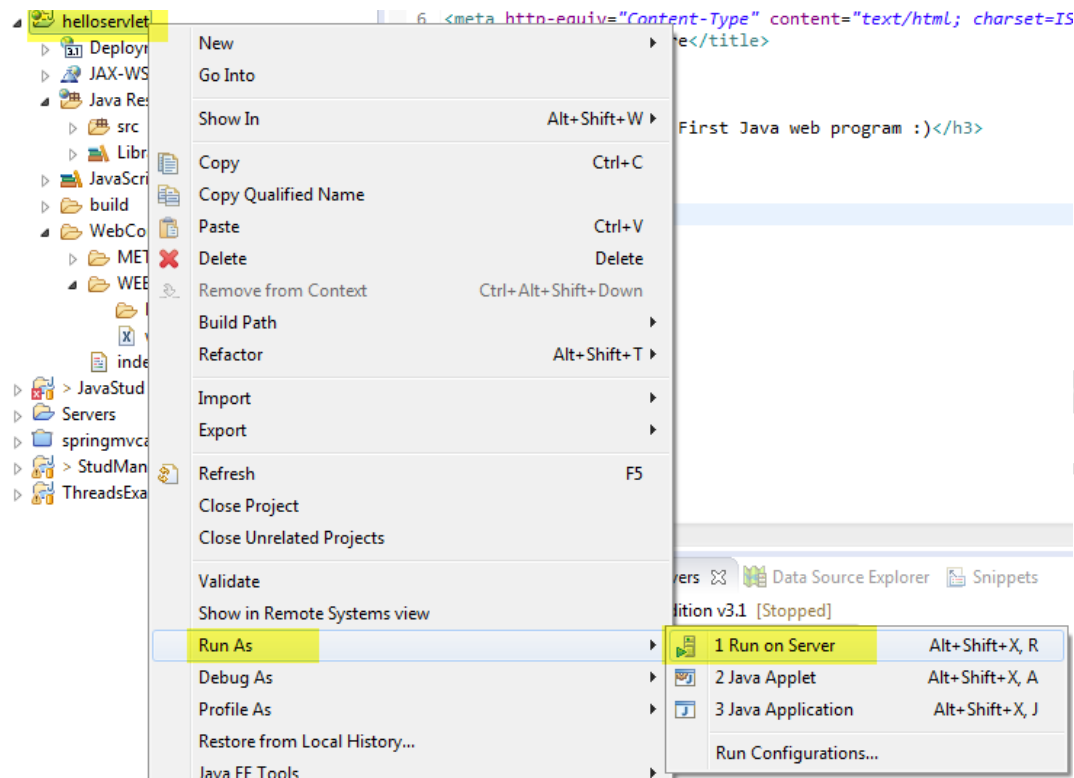
Working sets  
☐ Add project to working sets  
Working sets: Select...

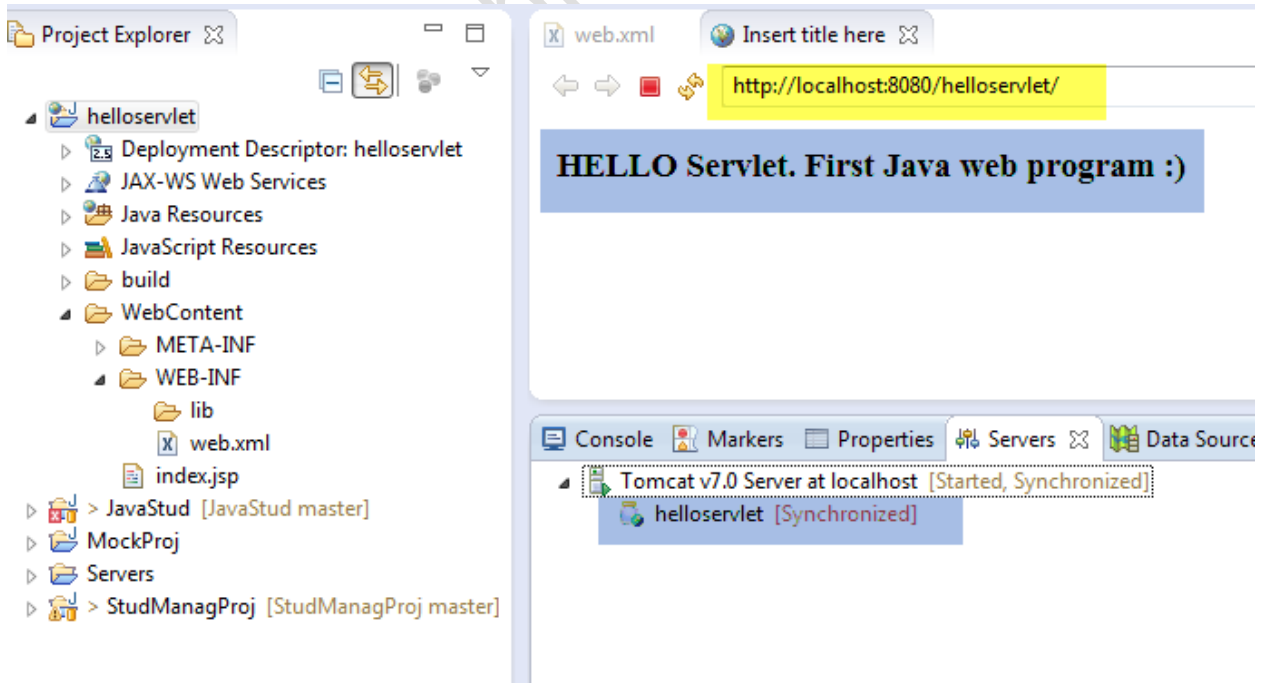
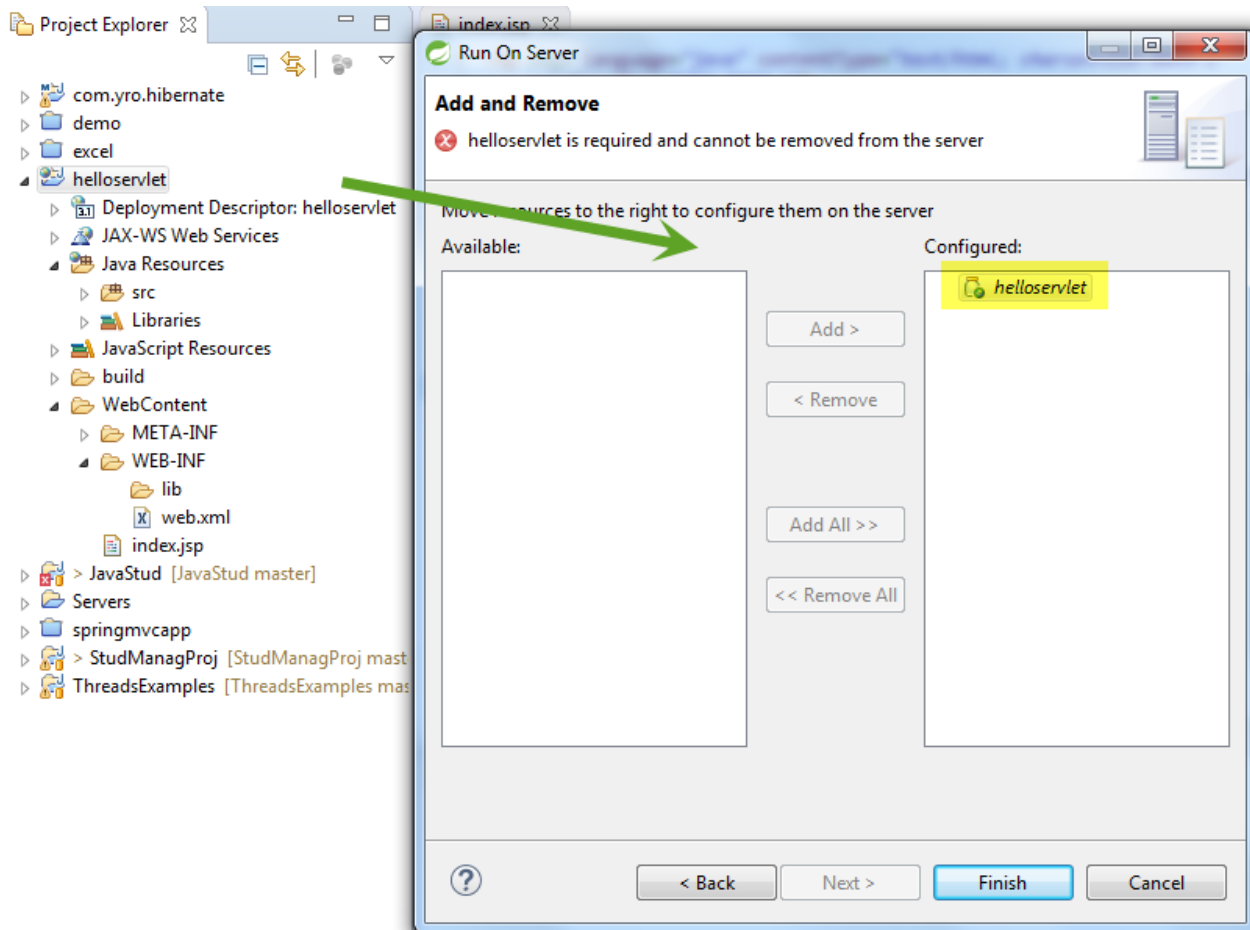
? < Back Next > Finish Cancel



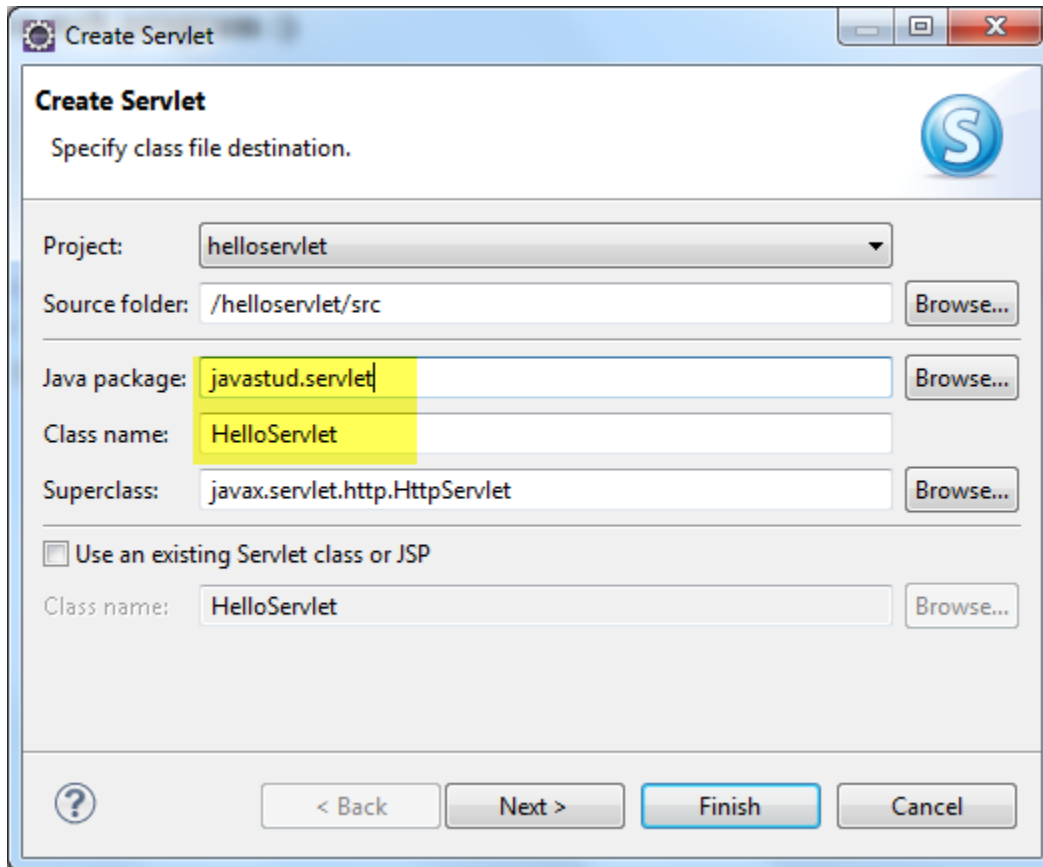
Let's first make web app run only by using JSP.







## Now using servlet: Create HelloServlet



**Create Servlet**  
Specify class file destination.

Project:

Source folder:

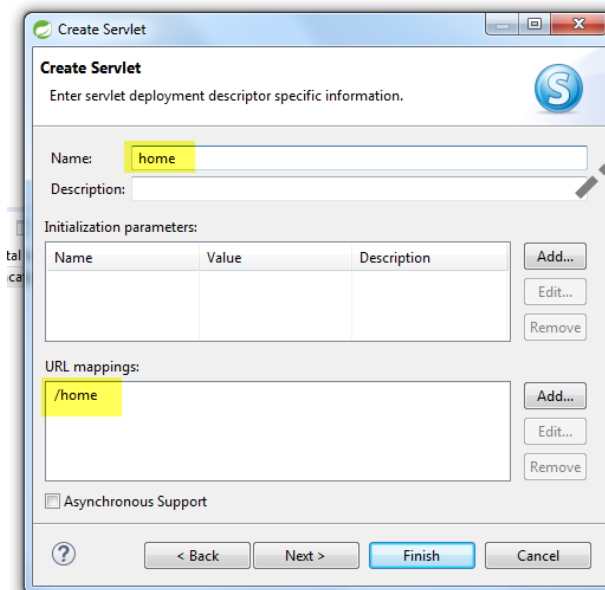
Java package:

Class name:

Superclass:

☐ Use an existing Servlet class or JSP

Class name:



**Create Servlet**  
Enter servlet deployment descriptor specific information.

Name:

Description:

Initialization parameters:

Name	Value	Description

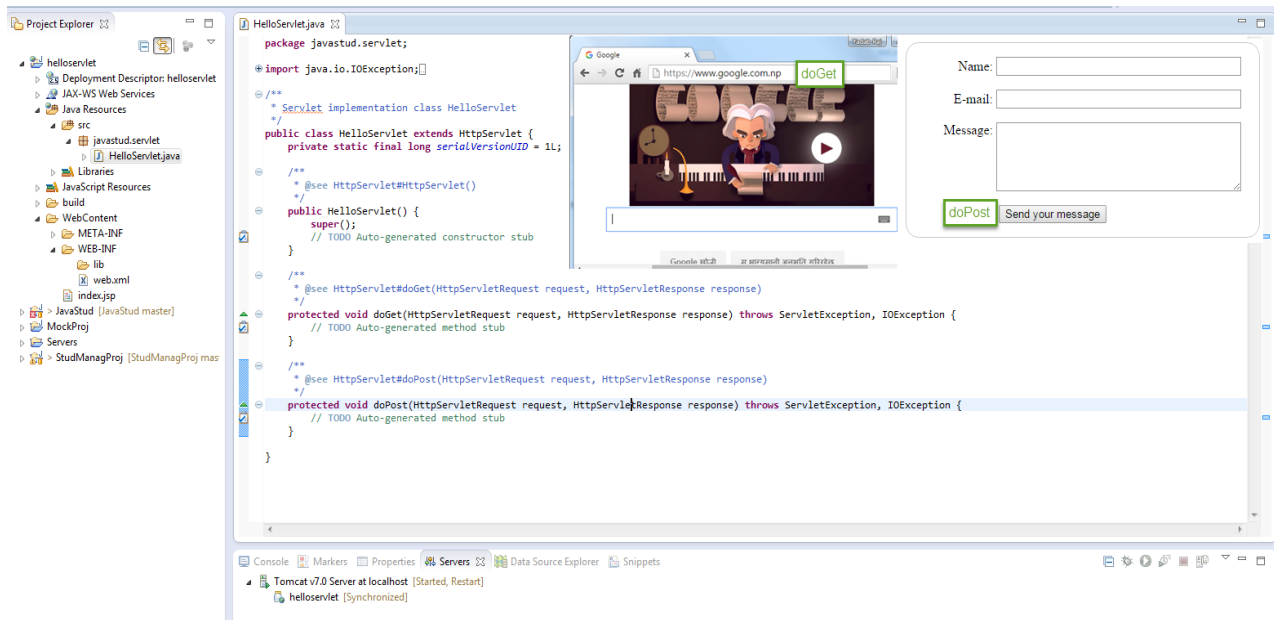
URL mappings:

/home
-------

☐ Asynchronous Support







```

/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    resp.setContentType("text/html");// setting the content type

    PrintWriter pw = resp.getWriter();// get the stream to write the data

    // writing html in the stream
    pw.println("<html><body>");
    pw.println("<h3>Welcome to Hello servlet</h3>");
    pw.println("</body></html>");

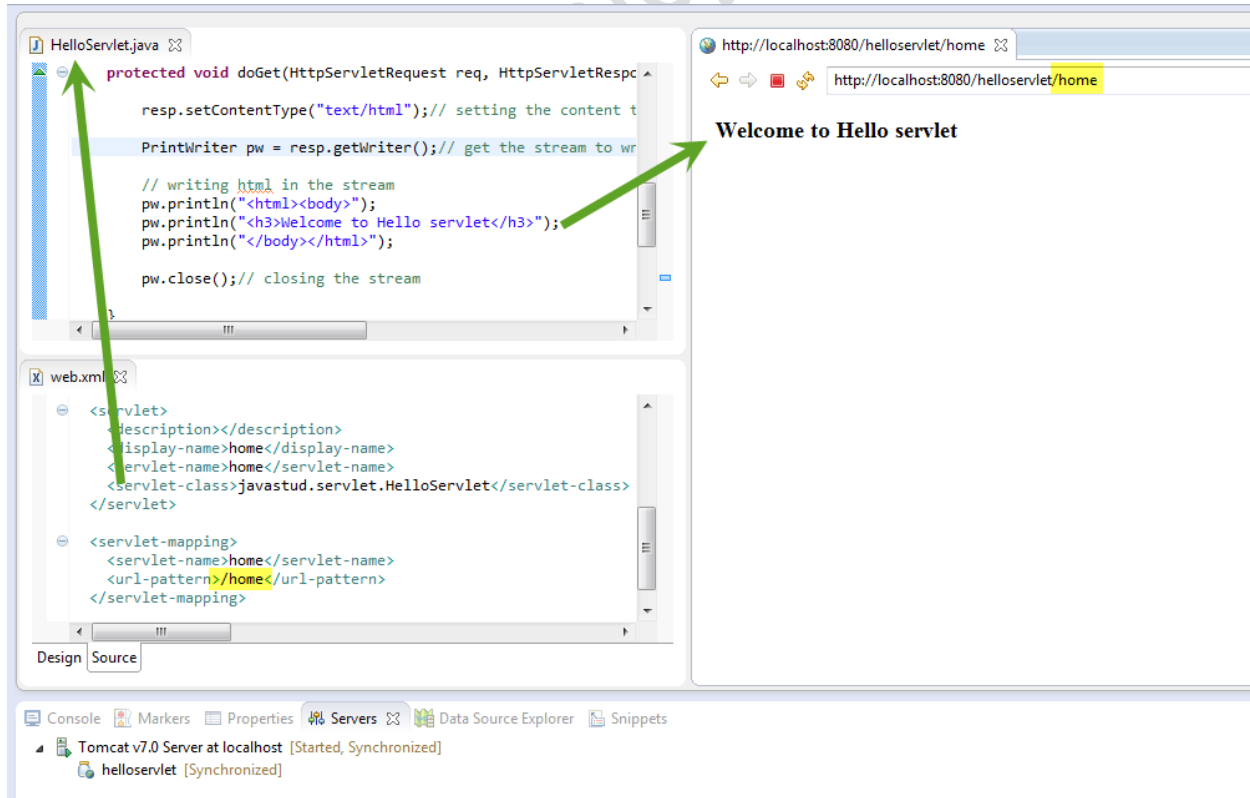
    pw.close();// closing the stream
}

```

## Configure the Application Deployment Descriptor - "web.xml"



Again run on server and restart server.



# 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.

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

TODO: Make show screenshot

## Passing More Parameters

Till Select Combo.

Request Dispatcher:

## RequestDispatcher in Servlet

1. [RequestDispatcher Interface](#)
2. [Methods of RequestDispatcher interface](#)
  1. [forward method](#)
  2. [include method](#)
3. [How to get the object of RequestDispatcher](#)
4. [Example of RequestDispatcher interface](#)

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.

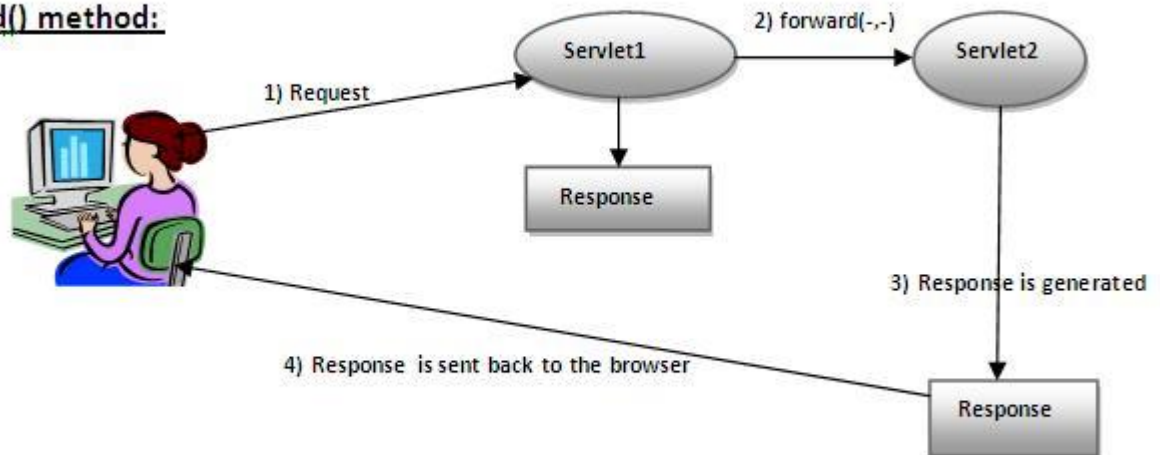
## Methods of RequestDispatcher interface

The RequestDispatcher interface provides two methods. They are:

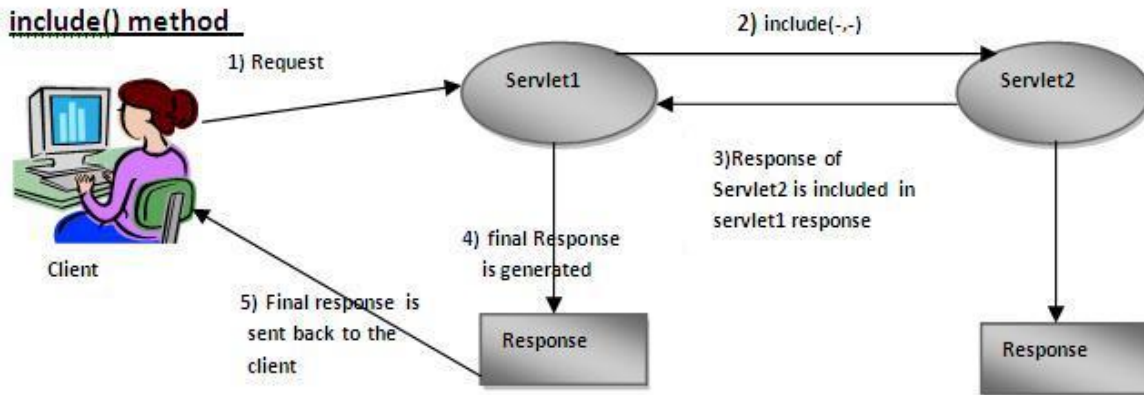
1. **public void forward(ServletRequest request, ServletResponse response) throws ServletException, java.io.IOException:** Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

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.

forward() method:



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:

### *Syntax of `getRequestDispatcher` method*

1. **public** `RequestDispatcher` `getRequestDispatcher(String resource);`

### *Example of using `getRequestDispatcher` method*

1. `RequestDispatcher rd=request.getRequestDispatcher("servlet2");`
2. `//servlet2 is the url-pattern of the second servlet`
- 3.
4. `rd.forward(request, response);` `//method may be include or forward`

## SendRedirect in servlet

It always sends a new request.

1. `response.sendRedirect("http://www.javatpoint.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.

### 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.

1. **public** ServletConfig getServletConfig();

## 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.

### Example of getServletContext() method

1. *//We can get the ServletContext object from ServletConfig object*
2. `ServletContext application=getServletConfig().getServletContext();`
- 3.
4. *//Another convenient way to get the ServletContext object*
5. `ServletContext application=getServletContext();`

# Attribute in Servlet

An **attribute in servlet** is an object that can be set, get or removed from one of the following scopes:

1. request scope
2. session scope
3. application scope

The servlet programmer can pass information's from one servlet to another using attributes. It is just like passing object from one class to another so that we can reuse the same object again and again.

## Attribute specific methods of ServletRequest, HttpSession and ServletContext interface

There are following 4 attribute specific methods. They are as follows:

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

```
//Set Attribute: Once set it will be then available every time.
ServletContext context = getServletContext();
context.setAttribute("Company", "Java Envagilist");

//For GetServletContext Attribute. Which is set from ServletContextSetAttribute
ServletContext context = getServletContext();
out.println("<br/> Company Name: " + context.getAttribute("company"));
```



# Session Tracking in Servlets

1. [Session Tracking](#)
2. [Session Tracking Techniques](#)

**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?

**To recognize the user** 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**

TODO:

# Servlet Login and Logout Example using Cookies

## Servlet Login and Logout Example using Cookies

A **cookie** is a kind of information that is stored at client side.

In the previous page, we learned a lot about cookie e.g. how to create cookie, how to delete cookie, how to get cookie etc.

Here, we are going to create a login and logout example using servlet cookies.

In this example, we are creating 3 links: login, logout and profile. User can't go to profile page until he/she is logged in. If user is logged out, he need to login again to visit profile.

In this application, we have created following files.

1. index.html
2. link.html
3. login.html
4. LoginServlet.java
5. LogoutServlet.java
6. ProfileServlet.java
7. web.xml

*File: index.html*

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset="ISO-8859-1">
5. <title>Servlet Login Example</title>
6. </head>
7. <body>
8.
9. <h1>Welcome to Login App by Cookie</h1>
10. <a href="login.html">Login</a> |
11. <a href="LogoutServlet">Logout</a> |
12. <a href="ProfileServlet">Profile</a>
13.
14. </body>
```

## 15. </html>

File: link.html

```
1. <a href="login.html">Login</a> |
2. <a href="LogoutServlet">Logout</a> |
3. <a href="ProfileServlet">Profile</a>
4. <hr>
```

File: login.html

```
1. <form action="LoginServlet" method="post">
2. Name:<input type="text" name="name"><br>
3. Password:<input type="password" name="password"><br>
4. <input type="submit" value="login">
5. </form>
```

File: LoginServlet.java

```
1. package com.javatpoint;
2.
3. import java.io.IOException;
4. import java.io.PrintWriter;
5. import javax.servlet.ServletException;
6. import javax.servlet.http.Cookie;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
10. public class LoginServlet extends HttpServlet {
11.     protected void doPost(HttpServletRequest request, HttpServletResponse response)
12.         throws ServletException, IOException {
13.         response.setContentType("text/html");
14.         PrintWriter out=response.getWriter();
15.
16.         request.getRequestDispatcher("link.html").include(request, response);
17.
18.         String name=request.getParameter("name");
19.         String password=request.getParameter("password");
20.
21.         if(password.equals("admin123")){
22.             out.print("You are successfully logged in!");
23.             out.print("<br>Welcome, "+name);
24.
25.             Cookie ck=new Cookie("name",name);
26.             response.addCookie(ck);
27.         }else{
28.             out.print("sorry, username or password error!");
29.             request.getRequestDispatcher("login.html").include(request, response);
30.         }
31.
32.         out.close();
33.     }
34. }
```

35. }

---

*File: LogoutServlet.java*

```
1. package com.javatpoint;
2.
3. import java.io.IOException;
4. import java.io.PrintWriter;
5. import javax.servlet.ServletException;
6. import javax.servlet.http.Cookie;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
10. public class LogoutServlet extends HttpServlet {
11.     protected void doGet(HttpServletRequest request, HttpServletResponse response)
12.         throws ServletException, IOException {
13.         response.setContentType("text/html");
14.         PrintWriter out=response.getWriter();
15.
16.
17.         request.getRequestDispatcher("link.html").include(request, response);
18.
19.         Cookie ck=new Cookie("name","");
20.         ck.setMaxAge(0);
21.         response.addCookie(ck);
22.
23.         out.print("you are successfully logged out!");
24.     }
25. }
```

---

*File: ProfileServlet.java*

```
1. package com.javatpoint;
2.
3. import java.io.IOException;
4. import java.io.PrintWriter;
5. import javax.servlet.ServletException;
6. import javax.servlet.http.Cookie;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
10. public class ProfileServlet extends HttpServlet {
11.     protected void doGet(HttpServletRequest request, HttpServletResponse response)
12.         throws ServletException, IOException {
13.         response.setContentType("text/html");
14.         PrintWriter out=response.getWriter();
15.
16.         request.getRequestDispatcher("link.html").include(request, response);
17.
18.         Cookie ck[]=request.getCookies();
19.         if(ck!=null){
20.             String name=ck[0].getValue();
21.             if(!name.equals(""))||name!=null){
```

```

22.         out.print("<b>Welcome to Profile</b>");
23.         out.print("<br>Welcome, "+name);
24.     }
25. }else{
26.     out.print("Please login first");
27.     request.getRequestDispatcher("login.html").include(request, response);
28. }
29. out.close();
30. }
31. }

```

=====COOKIE=====

1. Cookie ck=**new** Cookie("name",name);
2. response.addCookie(ck);

Delete:

1. Cookie ck=**new** Cookie("name","");
2. ck.setMaxAge(0);
3. response.addCookie(ck);

1. Cookie ck[]=request.getCookies();
2. **if**(ck!=**null**){
3. String name=ck[0].getValue();
- }

## URL Rewriting

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??

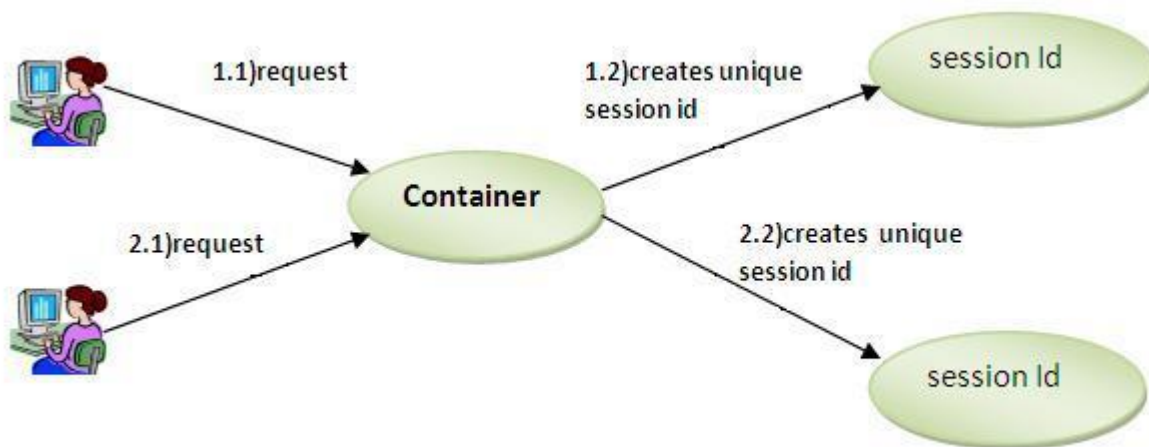
```
out.print("<a href='servlet2?uname="+n+">visit</a>");
```

## HttpSession interface

1. [HttpSession interface](#)
2. [How to get the HttpSession object](#)
3. [Commonly used methods of HttpSession interface](#)
4. [Example of using HttpSession](#)

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.

## Servlets - Writing Filters

Servlet Filters are Java classes that can be used in Servlet Programming for the following purposes:

- To intercept requests from a client before they access a resource at back end.
- To manipulate responses from server before they are sent back to the client.

There are are various types of filters suggested by the specifications:

- Authentication Filters.
- Data compression Filters.
- Encryption Filters.
- Filters that trigger resource access events.
- Image Conversion Filters.
- Logging and Auditing Filters.
- MIME-TYPE Chain Filters.
- Tokenizing Filters .

- XSL/T Filters That Transform XML Content.

Filters are deployed in the deployment descriptor file **web.xml** and then map to either servlet names or URL patterns in your application's deployment descriptor.

When the web container starts up your web application, it creates an instance of each filter that you have declared in the deployment descriptor. The filters execute in the order that they are declared in the deployment descriptor.

## Servlet Filter Methods:

A filter is simply a Java class that implements the `javax.servlet.Filter` interface. The `javax.servlet.Filter` interface defines three methods:

S.N.	Method & Description
1	<p><b>public void doFilter (ServletRequest, ServletResponse, FilterChain)</b></p> <p>This method is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.</p>
2	<p><b>public void init(FilterConfig filterConfig)</b></p> <p>This method is called by the web container to indicate to a filter that it is being placed into service.</p>
3	<p><b>public void destroy()</b></p> <p>This method is called by the web container to indicate to a filter that it is being taken out of service.</p>



## Using Multiple Filters:

Your web application may define several different filters with a specific purpose. Consider, you define two filters *AuthenFilter* and *LogFilter*. Rest of the process would remain as explained above except you need to create a different mapping as mentioned below:

```
<filter>
  <filter-name>LogFilter</filter-name>
  <filter-class>LogFilter</filter-class>
  <init-param>
    <param-name>test-param</param-name>
    <param-value>Initialization Paramter</param-value>
  </init-param>
</filter>

<filter>
  <filter-name>AuthenFilter</filter-name>
  <filter-class>AuthenFilter</filter-class>
  <init-param>
    <param-name>test-param</param-name>
    <param-value>Initialization Paramter</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>LogFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
  <filter-name>AuthenFilter</filter-name>
  <url-pattern>/*</url-pattern>
```

```
</filter-mapping>
```

YROJHA yadabrajojha@gmail.com