SECTION A

Java EE

The **Java EE** stands for **Java Enterprise Edition**, which was earlier known as J2EE and is currently known as Jakarta EE. It is a set of specifications wrapping around Java SE (Standard Edition). The Java EE provides a platform for developers with enterprise features such as distributed computing and web services. Java EE applications are usually run on reference run times such as **microservers** or **application servers**. Examples of some contexts where Java EE is used are e-commerce, accounting, banking information systems.

Specifications of Java EE

Java EE has several specifications which are useful in making web pages, reading and writing from database in a transactional way, managing distributed queues. The Java EE contains several APIs which have the functionalities of base Java SE APIs such as Enterprise JavaBeans, connectors, Servlets, Java Server Pages and several web service technologies.

1. Web Specifications of Java EE :

a. Servlet- This specification defines how you can manage HTTP requests either in a synchronous or asynchronous way. It is low level, and other specifications depend on it

b. WebSocket- WebSocket is a computer communication protocol, and this API provides a set of APIs to facilitate WebSocket connections.

c. Java Server Faces- It is a service which helps in building GUI out of components.

d. Unified Expression Language- It is a simple language which was designed to facilitate web application developers.

2. Web Service Specifications of Java EE

- Java API for RESTful Web Services- It helps in providing services having Representational State Transfer schema.
- Java API for JSON Processing- It is a set of specifications to manage the information provided in JSON format.
- Java API for JSON Binding- It is a set of specifications provide for binding or parsing a JSON file into Java classes.

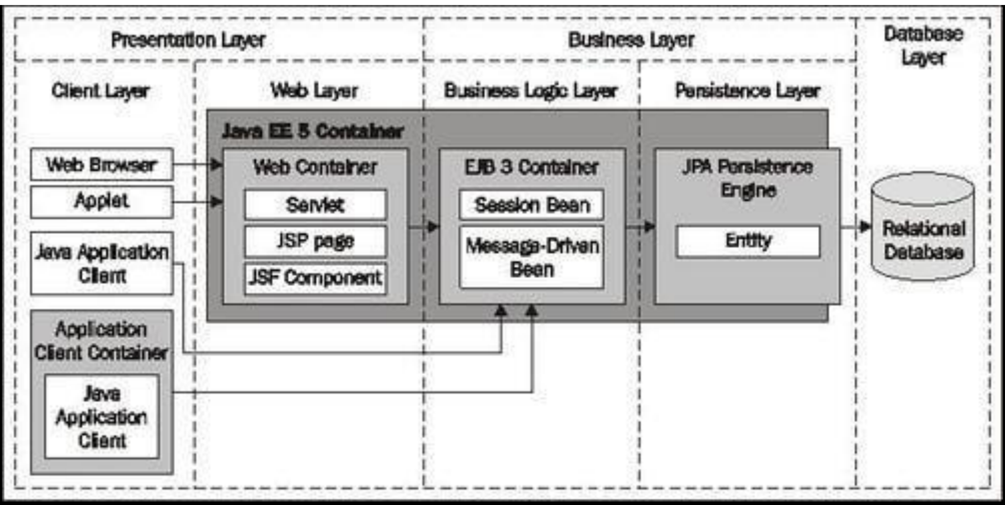
- Java Architecture for XML Binding- It allows binding of xml into Java objects.
- Java API for XML Web Services- SOAP is an xml based protocol to access web services over http. This API allows you to create SOAP web services

JAVA EE ARCHITECTURE

Java Platform Enterprise Edition (Java EE) technology provides services to enterprise applications using a multi-layer architecture. Java EE applications are web-enabled and Java based, which means they may be written once and deployed on any container supporting the Java EE standard. An application server is the environment in which the container resides. However, in practice we don't need to distinguish between an application server and a container, so we will use the terms interchangeably. The Java EE specification is supported by commercial vendors such as Sun, IBM, Oracle, BEA Systems as well as open-source ventures such as JBoss.

Java EE presentation layer technologies include servlets, JSP pages, and JSF components. These are developed for a business application then subsequently deployed and run in a web container. A client would interact with the web container either from a browser or an applet. In either case the http or https internet protocol would be used for communication.

Enterprise JavaBeans version 3 (EJB 3) is the technology Java EE version 5 (Java EE 5) provides for the business layer. In Java EE 5 we subdivide the business layer into one layer which is concerned with business processing and a second layer which deals with persistence. In EJB 3 the business processing artifacts are **session** and **message-driven beans**. These are developed for a business application and deployed and run in an EJB container. The persistence layer artifact is an **entity**; this is persisted to the database layer using a **persistence provider** or **persistence engine**. The persistence engine implements another specification, the **Java Persistence API (JPA)**. Both EJB 3 and the JPA are specifications for which a number of organizations provide implementations. Both specifications can be downloaded from http://www.jcp.org/en/jsr/detail?id=220. The figure below summarizes Java EE 5 architecture:



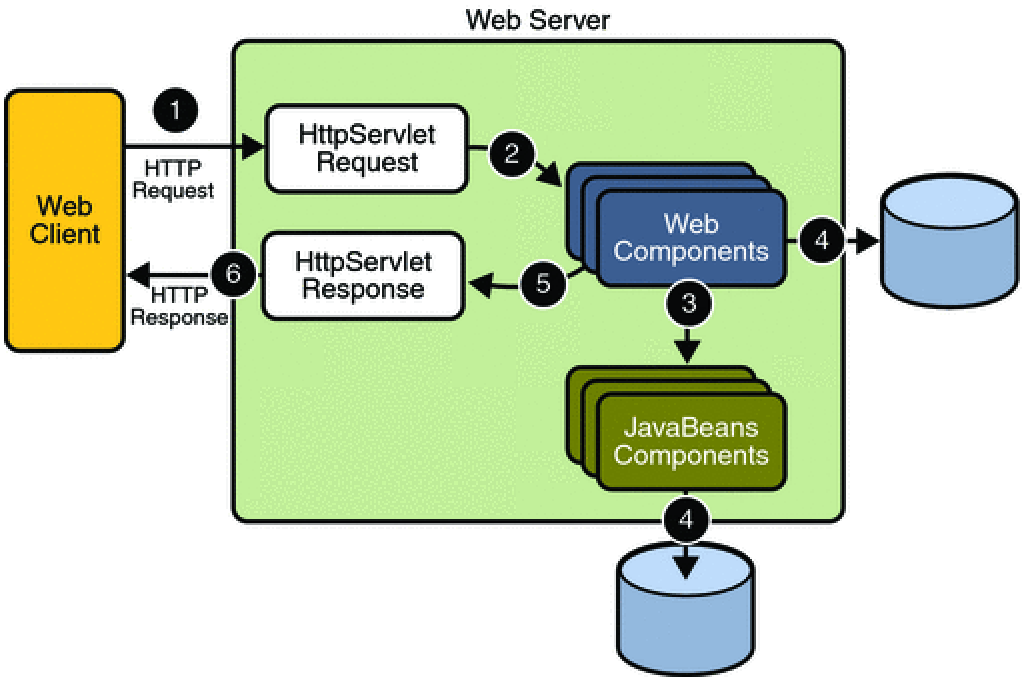
Note that our 3-layer model has become 5-layers. The distinction between client/web and business logic/persistence layers is not always made. Consequently we refer to Java EE architecture simply as n-layer or multi-layer. A Java EE container offers many other services such as web services, the Java Messaging Service (JMS), and resource adapters.

Note from the diagram that we can access an EJB directly from a Java SE application, such as Swing, without going through a web container. The Java application can be stand-alone, or can be run from an Application Client Container (ACC). An ACC enables a client executing in its own Java Virtual Machine (JVM) outside the EJB container to access a limited number of Java EE services.

Web Applications

In the Java 2 platform, **web components** provide the dynamic extension capabilities for a web server. Web components are either Java servlets, JSP pages, or web service endpoints. The interaction between a web client and a web application is illustrated in Figure 3-1. The client sends an HTTP request to the web server. A web server that implements Java Servlet and JavaServer Pages technology converts the request into an `HTTPServletRequest` object. This object is delivered to a web component, which can interact with JavaBeans components or a database to generate dynamic content. The web component can then generate an `HTTPServletResponse` or it can pass the request to another web component. Eventually a web component generates a `HTTPServletResponse` object. The web server converts this object to an HTTP response and returns it to the client.

Figure 3-1 Java Web Application Request Handling



Introduction to JDBC (Java Database Connectivity)

JDBC or Java Database Connectivity is a Java API to connect and execute the query with the database. It is a specification from Sun microsystems that provides a standard abstraction(API or Protocol) for java applications to communicate with various databases. It provides the language with java database connectivity standards. It is used to write programs required to access databases. JDBC, along with the database driver, can access databases and spreadsheets. The enterprise data stored in a relational database(RDB) can be accessed with the help of JDBC APIs.

Definition of JDBC(Java Database Connectivity)

JDBC is an API(Application programming interface) used in java programming to interact with databases. *The classes and interfaces of JDBC* allow the *application to send* requests *made by users to the specified database.*

Purpose of JDBC

Enterprise applications created using the JAVA EE technology need to interact with databases to store application-specific information. So, interacting with a database requires efficient database connectivity, which can be achieved by using the ODBC(Open database connectivity) driver. This driver is used with JDBC to interact or communicate with various kinds of databases such as Oracle, MS Access, Mysql, and SQL server database

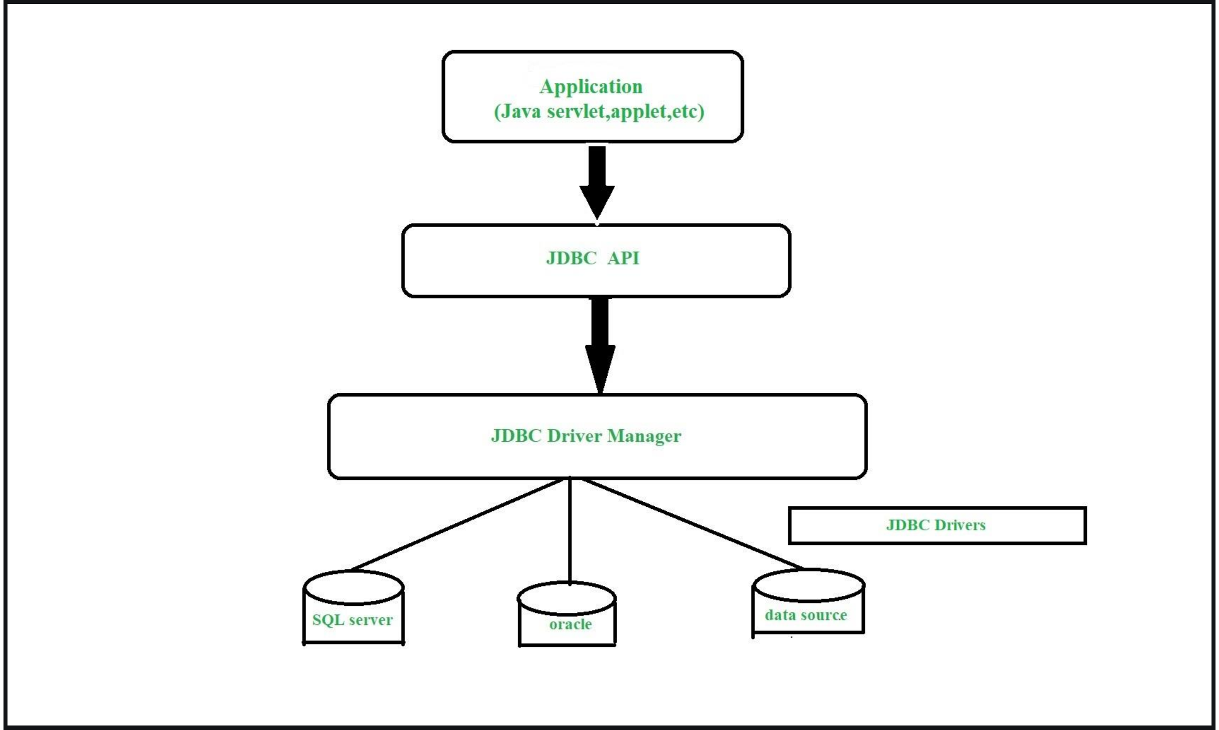
Components of JDBC

There are generally four main components of JDBC through which it can interact with a database. They are as mentioned below:

1. JDBC API: It provides various methods and interfaces for easy communication with the database. It provides two packages as follows, which contain the java SE and Java EE platforms to exhibit WORA(write once run everywhere) capabilities.

2. It also provides a standard to connect a database to a client application.
3. JDBC Driver manager**:** It loads a database-specific driver in an application to establish a connection with a database. It is used to make a database-specific call to the database to process the user request.
4. JDBC Test suite: It is used to test the operation(such as insertion, deletion, updation) being performed by JDBC Drivers.
5. JDBC-ODBC Bridge Drivers: It connects database drivers to the database. This bridge translates the JDBC method call to the ODBC function call. It makes use of the **sun.jdbc.odbc** package which includes a native library to access ODBC characteristics.

Architecture of JDBC



Architecture of JDBC

Description:

1. **Application:** It is a java applet or a servlet that communicates with a data source.

2. **The JDBC API:** The JDBC API allows Java programs to execute SQL statements and retrieve results. Some of the important classes and interfaces defined in JDBC API are as follows:
3. **DriverManager:** It plays an important role in the JDBC architecture. It uses some database-specific drivers to effectively connect enterprise applications to databases.
4. **JDBC drivers:** To communicate with a data source through JDBC, you need a JDBC driver that intelligently communicates with the respective data source.

JDBC Drivers

JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

1. Type-1 driver or JDBC-ODBC bridge driver
2. Type-2 driver or Native-API driver
3. Type-3 driver or Network Protocol driver
4. Type-4 driver or Thin driver

ypes of JDBC Architecture(2-tier and 3-tier)

The JDBC architecture consists of two-tier and three-tier processing models to access a database. They are as described below:

1. **Two-tier model:** A java application communicates directly to the data source. The JDBC driver enables the communication between the application and the data source. When a user sends a query to the data source, the answers for those queries are sent back to the user in the form of results.
   The data source can be located on a different machine on a network to which a user is connected. This is known as a **client/server configuration**, where the user's machine acts as a client, and the machine has the data source running acts as the server.

2. **Three-tier model:** In this, the user's queries are sent to middle-tier services, from which the commands are again sent to the data source. The results are sent back to the middle tier, and from there to the user.
   This type of model is found very useful by management information system directors.

Interfaces of JDBC API

A list of popular *interfaces* of JDBC API is given below:

- Driver interface
- Connection interface

- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

The Java Database Connectivity (JDBC) API provides universal data access from the Java programming language. Using the JDBC API, you can access virtually any data source, from relational databases to spreadsheets and flat files. JDBC technology also provides a common base on which tools and alternate interfaces can be built.

The JDBC API is comprised of two packages:

- `java.sql`
- `javax.sql`

You automatically get both packages when you download the Java Platform Standard Edition (Java SE) 8.

To use the JDBC API with a particular database management system, you need a JDBC technology-based driver to mediate between JDBC technology and the database. Depending on various factors, a driver might be written purely in the Java programming language or in a mixture of the Java programming language and Java Native Interface (JNI) native methods. To obtain a JDBC driver for a particular database management system, see JDBC Data Access API.

Working of JDBC

Java application that needs to communicate with the database has to be programmed using JDBC API. JDBC Driver supporting data sources such as Oracle and SQL server has to be added in java application for JDBC support which can be done dynamically at run time. This JDBC driver intelligently communicates the respective data source.

Creating a simple JDBC application

```
package com.vinayak.jdbc;

import java.sql.*;

public class JDBCDemo {

    public static void main(String args[])
        throws SQLException, ClassNotFoundException
    {
        String driverClassName
            = "sun.jdbc.odbc.JdbcOdbcDriver";
        String url = "jdbc:odbc:XE";
        String username = "scott";
        String password = "tiger";
        String query
            = "insert into students values(109, 'bhatt')";
        // Load driver class
        Class.forName(driverClassName);

        // Obtain a connection
        Connection con = DriverManager.getConnection(
            url, username, password);

        // Obtain a statement
        Statement st = con.createStatement();

        // Execute the query
        int count = st.executeUpdate(query);
        System.out.println(
            "number of rows affected by this query= "
            + count);

        // Closing the connection as per the
        // requirement with connection is completed
        con.close();
    }
} // class
```

SECTION B

Introduction to Java Servlets

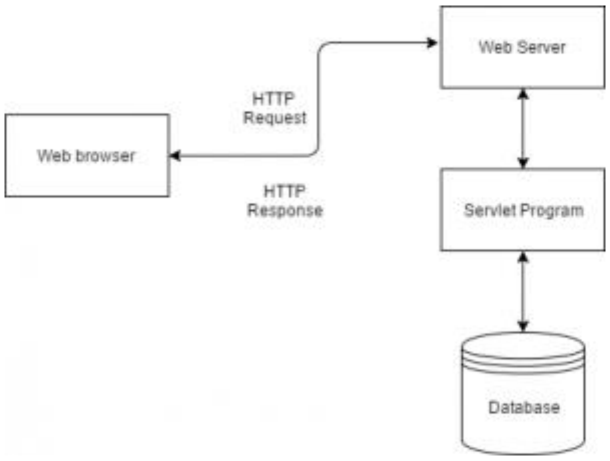
Today we all are aware of the need of creating dynamic web pages i.e the ones which have the capability to change the site contents according to the time or are able to generate the contents according to the request received by the client. If you like coding in Java, then you will be happy to know that using Java there also exists a way to generate dynamic web pages and that way is Java Servlet. But before we move forward with our topic let’s first understand the need for server-side extensions.

Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.

Properties of Servlets are as follows:

- Servlets work on the server-side.
- Servlets are capable of handling complex requests obtained from the webserver.

Servlet Architecture is can be depicted from the image itself as provided below as follows:



Execution of Servlets basically involves six basic steps:

1. The clients send the request to the webserver.
2. The web server receives the request.
3. The web server passes the request to the corresponding servlet.
4. The servlet processes the request and generates the response in the form of output.
5. The servlet sends the response back to the webserver.
6. The web server sends the response back to the client and the client browser displays it on the screen.

Features of Servlet

Now that we have understood what is a servlet and for what purpose it is being used. Let's proceed further and discuss its main features.

1. Portable:
As I mentioned above that Servlet uses Java as a programming language, Since java is platform independent, the same holds true for servlets. For example, you can create a servlet on Windows operating system that users GlassFish as web server and later run it on any other operating system like Unix, Linux with Apache tomcat web server, this feature makes servlet portable and this is the main advantage servlet has over CGI.

2. Efficient and scalable:
Once a servlet is deployed and loaded on a web server, it can instantly start fulfilling request of clients. The web server invokes servlet using a lightweight thread so multiple client requests can be fulling by servlet at the same time using the multithreading feature of Java. Compared to CGI where the server has to initiate a new process for every client request, the servlet is truly efficient and scalable.

3. Robust:
By inheriting the top features of Java (such as Garbage collection, Exception handling, Java Security Manager etc.) the servlet is less prone to memory management issues and memory leaks. This makes development of web application in servlets secure and less error prone.

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:

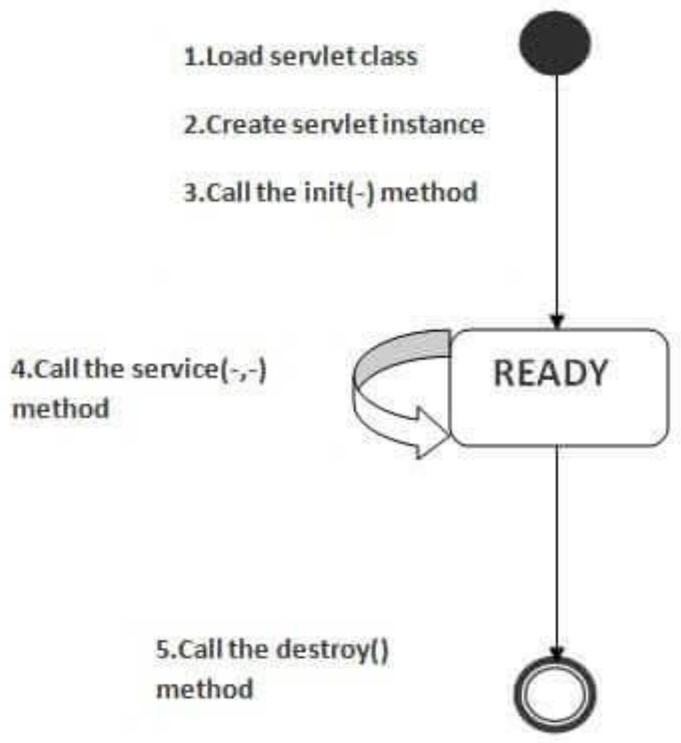
1. HttpServlet
2. Cookie

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.


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
4.Call the service(-,-) method
READY
5.Call the destroy() method

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.

javax.servlet.ServletConfig is an interface as a part of servlet API. For every Servlet class in our application, the web container will create one ServletConfig object and the web container will pass this object as an argument to the **public void init(ServletConfig config)** method of our Servlet class object. Some of the important points on ServletConfig are:

- ServletConfig is an object containing some initial parameters or configuration information created by Servlet Container and passed to the servlet during initialization.
- ServletConfig is for a particular servlet, which means one should store servlet-specific information in web.xml and retrieve them using this object.

Example

Suppose, one is building a job portal and desires to share different email ids (which may get changed over time) with to recruiter and job applicant. So, he decides to write two servlets one for handling the recruiter's request and another one for the job applicant.

Where to store email-ids?
Put email-id as a name-value pair for different servlet inside web.xml which can further be retrieved using getServletConfig().getInitParameter("name") in the servlet.

Methods in the ServletConfig Interface

There are 4 Methods in the ServletConfig interface

1. **public abstract java.lang.String getServletName()**
   In <servlet-name> we have given a logical name for our TestServlet class. So if we use the method getServletName() it will return the logical name of Servlet and in this case, it will return "Test"**.**
2. **public abstract javax.servlet.ServletContext getServletContext()**
   This method will simply return ServletContext Object. Web container creates one ServletContext object for every web application.
3. **public abstract java.lang.String getInitParameter(java.lang.String)**

We can store init parameters as a part of web.xml

4. **public abstract java.util.Enumeration<java.lang.String> getInitParameterNames()**
   This method will return Enumeration having names of all init parameter names.

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

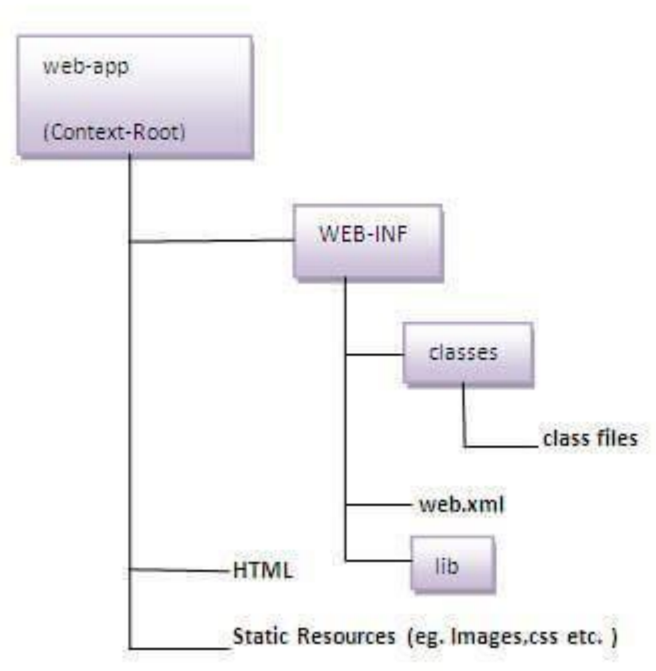
The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

Here, we are going to use **apache tomcat server** in this example. The 6 steps are as follows:

1. Create a directory structure

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. Create 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 h doGet(), doPost, doHead() etc.

In this example we are going to create a servlet that extends the HttpServlet class. In this example, HttpServlet class and providing the implementation of the doGet() method. Notice that get request i

DemoServlet.java

```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 a deployment descriptor

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

<servlet-mapping>
<servlet-name>sonoojaiswal</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).

6. Access the servlet

Open broser and write http://hostname:portno/contextroot/urlpatternofservlet. For example:

Session In Java

The time interval in which two systems(i.e. the client and the server) communicate with each other can be termed as a session. In simpler terms, a session is a state consisting of several requests and response between the client and the server.

It is a known fact that HTTP and Web Servers are both stateless. Hence, the only way to maintain the state of the user is by making use of technologies that implement session tracking.
Session tracking in servlets can be implemented by a number of methods, cookies being one of them. However, they have multiple disadvantages:

- Only textual information can be kept by them.
- If cookies are disabled by a user, the web application is unable to make use of them.
- Not more than 4kb of data can be contained by a single cookie.

- Another way to implement session tracking is by creating sessions with unique session ids for every user in a java servlet.

Moving on with this article on Session in Java

Advantages:

- All kinds of objects, such as database and text can be stored into a session.
- Sessions are secure.

Moving on with Disadvantages

Disadvantages:

- Since the session object is stored on a server, there is performance overhead.
- Serialization and de-serialization also lead to overhead.

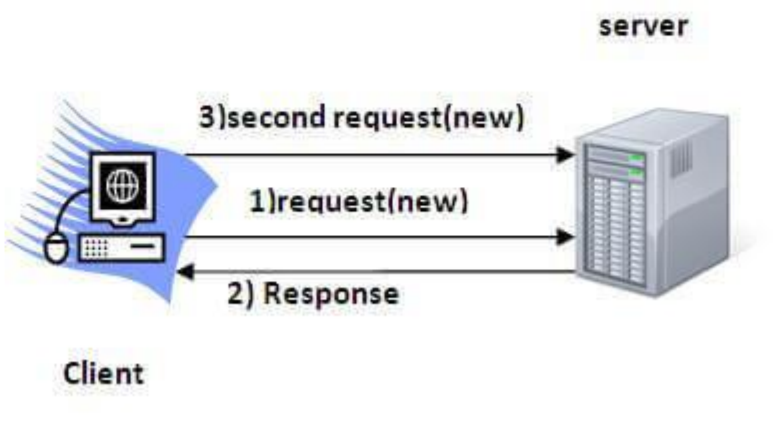
Session Tracking in Servlets

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?

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**

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.

2. **Hidden Form Field**

In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.

In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Let's see the code to store value in hidden field.

```
<input type="hidden" name="uname" value="Vimal Jaiswal">
```

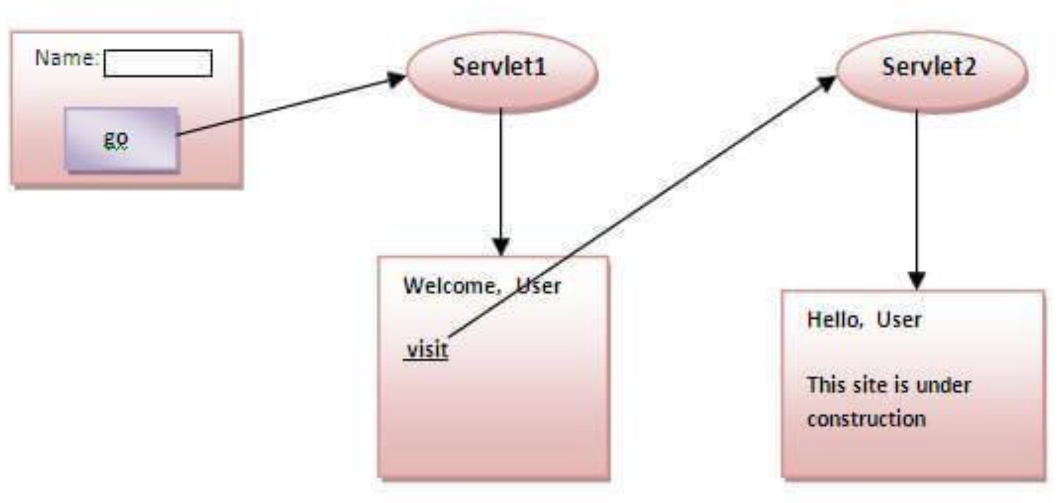
Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.

3. **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&??

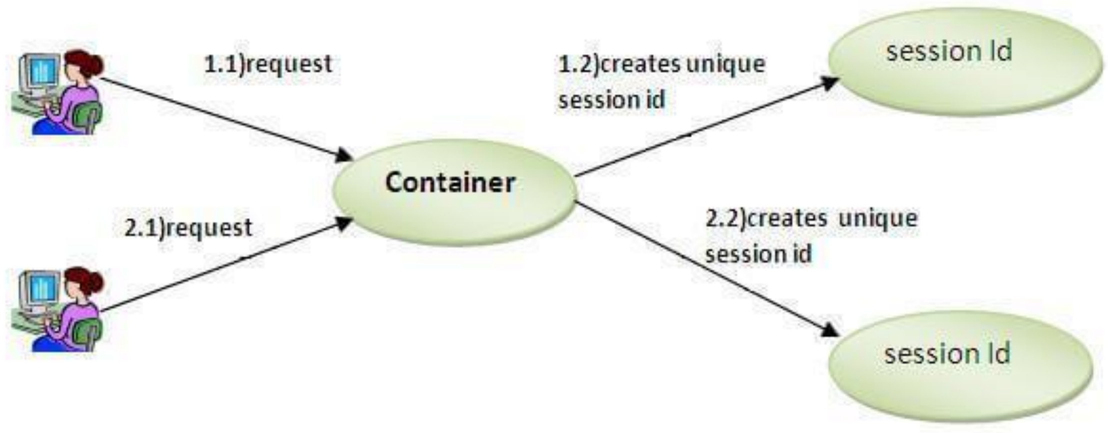
A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.



4. **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.



Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The
provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener

FocusEvent	FocusListener

Event and Listener in Servlet

Events are basically occurrence of something. Changing the state of an object is known as an event.

We can perform some important tasks at the occurrence of these exceptions, such as counting total and current logged-in users, creating tables of the database at time of deploying the project, creating database connection object etc.

There are many Event classes and Listener interfaces in the javax.servlet and javax.servlet.http packages.

Event classes

The event classes are as follows:

1. ServletRequestEvent
2. ServletContextEvent
3. ServletRequestAttributeEvent
4. ServletContextAttributeEvent
5. HttpSessionEvent
6. HttpSessionBindingEvent

Event interfaces

The event interfaces are as follows:

1. ServletRequestListener
2. ServletRequestAttributeListener
3. ServletContextListener
4. ServletContextAttributeListener
5. HttpSessionListener
6. HttpSessionAttributeListener
7. HttpSessionBindingListener
8. HttpSessionActivationListener

SECTION C

Introduction

- It stands for **Java Server Pages**.
- It is a server side technology.
- It is used for creating web application.
- It is used to create dynamic web content.
- In this JSP tags are used to insert JAVA code into HTML pages.
- It is an advanced version of Servlet Technology.
- It is a Web based technology helps us to create dynamic and platform independent web pages.
- In this, Java code can be inserted in HTML/ XML pages or both.
- JSP is first converted into servlet by JSP container before processing the client's request.

JSP pages are more advantageous than Servlet:

- They are easy to maintain.
- No recompilation or redeployment is required.
- JSP has access to entire API of JAVA .
- JSP are extended version of Servlet.

Features of JSP

- **Coding in JSP is easy** :- As it is just adding JAVA code to HTML/XML.
- **Reduction in the length of Code** :- In JSP we use action tags, custom tags etc.
- **Connection to Database is easier** :-It is easier to connect website to database and allows to read or write data easily to the database.
- **Make Interactive websites** :- In this we can create dynamic web pages which helps user to interact in real time environment.
- **Portable, Powerful, flexible and easy to maintain** :- as these are browser and server independent.
- **No Redeployment and No Re-Compilation** :- It is dynamic, secure and platform independent so no need to re-compilation.
- **Extension to Servlet** :- as it has all features of servlets, implicit objects and custom tags

Syntax available in JSP are following

1. **Declaration Tag** :-It is used to declare variables.

   Syntax:-
   ```
   <%!  Dec var  %>
   ```
   Example:-
   ```
   <%! int var=10; %>
   ```

2. **Java Scriplets** :- It allows us to add any number of JAVA code, variables and expressions.

   Syntax:-
   ```
   <% java code %>
   ```

3. **JSP Expression** :- It evaluates and convert the expression to a string.

   Syntax:-
   ```
   <%= expression %>
   ```
   Example:-
   ```
   <% num1 = num1+num2 %>
   ```

4. **JAVA Comments** :- It contains the text that is added for information which has to be ignored.

   Syntax:-
   ```
   <% -- JSP Comments %>
   ```

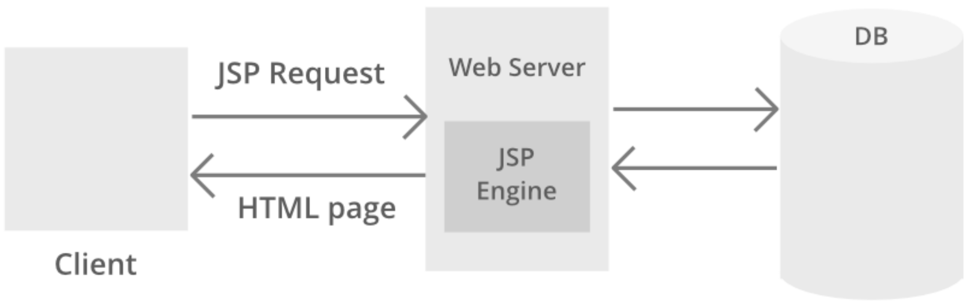
Process of Execution

Steps for Execution of JSP are following:-

- Create html page from where request will be sent to server eg try.html.
- To handle to request of user next is to create .jsp file Eg. new.jsp
- Create project folder structure.
- Create XML file eg my.xml.
- Create WAR file.
- Start Tomcat
- Run Application

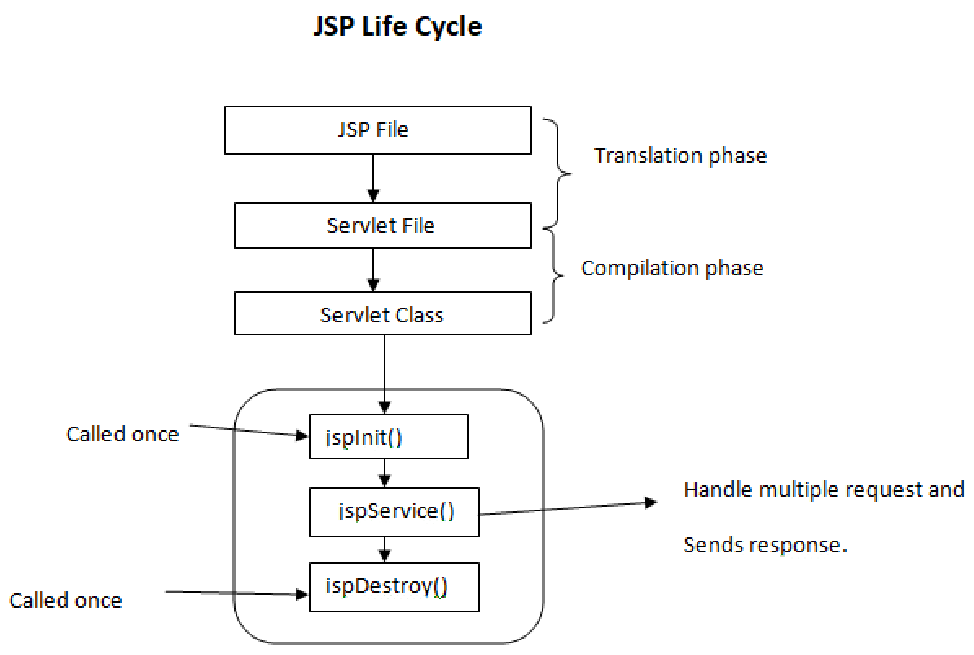
JSP Architecture

JSP architecture gives a high-level view of the working of JSP. JSP architecture is a 3 tier architecture. It has a Client, Web Server, and Database. The client is the web browser or application on the user side. Web Server uses a JSP Engine i.e; a container that processes JSP. For example, Apache Tomcat has a built-in JSP Engine. JSP Engine intercepts the request for JSP and provides the runtime environment for the understanding and processing of JSP files. It reads, parses, build Java Servlet, Compiles and Executes Java code, and returns the HTML page to the client. The webserver has access to the Database. The following diagram shows the architecture of JSP.



Life cycle of JSP

A Java Server Page life cycle is defined as the process that started with its creation which later translated to a servlet and afterward servlet lifecycle comes into play. This is how the process goes on until its destruction.



Following steps are involved in the JSP life cycle:

- Translation of JSP page to Servlet
- Compilation of JSP page(Compilation of JSP into test.java)
- Classloading (test.java to test.class)
- Instantiation(Object of the generated Servlet is created)
- Initialization(jspInit() method is invoked by the container)

- Request processing(_jspService()is invoked by the container)
- JSP Cleanup (jspDestroy() method is invoked by the container)

Translation of JSP page to Servlet :
This is the first step of the JSP life cycle. This translation phase deals with the Syntactic correctness of JSP. Here test.jsp file is translated to test.java.

Compilation of JSP page :
Here the generated java servlet file (test.java) is compiled to a class file (test.class).

Classloading :
Servlet class which has been loaded from the JSP source is now loaded into the container.

Instantiation :
Here an instance of the class is generated. The container manages one or more instances by providing responses to requests.

Initialization :
jspInit() method is called only once during the life cycle immediately after the generation of Servlet instance from JSP.

Request processing :
_jspService() method is used to serve the raised requests by JSP. It takes request and response objects as parameters. This method cannot be overridden.

JSP Cleanup :
In order to remove the JSP from the use by the container or to destroy the method for servlets jspDestroy()method is used. This method is called once, if you need to perform any cleanup task like closing open files, releasing database connections jspDestroy() can be overridden.

Elements of JSP

The Scriptlet

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Following is the syntax of Scriptlet −

```
<% code fragment %>
```

You can write the XML equivalent of the above syntax as follows −

```
<jsp:scriptlet>
   code fragment
</jsp:scriptlet>
```

Any text, HTML tags, or JSP elements you write must be outside the scriptlet. Following is the simple and first example for JSP −

```
<html>
   <head><title>Hello World</title></head>

   <body>
      Hello World!<br/>
      <%
         out.println("Your IP address is " +
request.getRemoteAddr());
      %>
   </body>
</html>
```

JSP Directives

These directives provide directions and instructions to the container, telling it how to handle certain aspects of the JSP processing.

A JSP directive affects the overall structure of the servlet class. It usually has the following form −

```
<%@ directive attribute = "value" %>
```

Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.

The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

There are three types of directive tag −

S.No.	Directive & Description
1	**<%@ page ... %>** Defines page-dependent attributes, such as scripting language, error page, and bufferin requirements.
2	**<%@ include ... %>** Includes a file during the translation phase.
3	**<%@ taglib ... %>** Declares a tag library, containing custom actions, used in the page

JSP Actions

These actions use constructs in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.

There is only one syntax for the Action element, as it conforms to the XML standard −

```
<jsp:action_name attribute = "value" />
```

Action elements are basically predefined functions. The following table lists out the available JSP actions −

S.No.	Syntax & Purpose
1	**jsp:include** Includes a file at the time the page is requested.
2	**jsp:useBean** Finds or instantiates a JavaBean.
3	**jsp:setProperty** Sets the property of a JavaBean.
4	**jsp:getProperty** Inserts the property of a JavaBean into the output.
5	**jsp:forward** Forwards the requester to a new page.
6	**jsp:plugin** Generates browser-specific code that makes an OBJECT or EMBED tag for the Jav plugin.
7	**jsp:element** Defines XML elements dynamically.
8	**jsp:attribute**

	Defines dynamically-defined XML element's attribute.
9	**jsp:body** Defines dynamically-defined XML element's body.
10	**jsp:text** Used to write template text in JSP pages and documents.

JSP Implicit Objects

These Objects are the Java objects that the JSP Container makes available to the developers in each page and the developer can call them directly without being explicitly declared. JSP Implicit Objects are also called **pre-defined variables**.

Following table lists out the nine Implicit Objects that JSP supports −

S.No.	Object & Description
1	**request** This is the **HttpServletRequest** object associated with the request.
2	**response** This is the **HttpServletResponse** object associated with the response to the client.
3	**out** This is the **PrintWriter** object used to send output to the client.
4	**session** This is the **HttpSession** object associated with the request.
5	**application** This is the **ServletContext** object associated with the application context.
6	**config** This is the **ServletConfig** object associated with the page.

7	**pageContext** This encapsulates use of server-specific features like higher performance **JspWriters**.
8	**page** This is simply a synonym for **this**, and is used to call the methods defined by the translate servlet class.
9	**Exception** The **Exception** object allows the exception data to be accessed by designated JSP.

JSP Best Practices

1. Separate HTML from Java
2. Place business logic in JavaBeans
3. Factor general behavior out of custom tag handler classes
4. Favor HTML in Java handler classes over Java in JSPs
5. Use an appropriate inclusion mechanism
6. Use a JSP template mechanism
7. Use stylesheets
8. Use the MVC pattern
9. Use available custom tag libraries
10. Determine the appropriate level of XML compliance
11. Use JSP comments in most cases
12. Follow HTML best practices
13. Utilize the JSP exception mechanism

These tips will help you write JSPs that are reusable and easy to maintain.

JSTL - JSP Standard Tag library

The JavaServer Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates the core functionality common to many JSP applications.

JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating the existing custom tags with the JSTL tags.

Install JSTL Library

To begin working with JSP tages you need to first install the JSTL library. If you are using the Apache Tomcat container, then follow these two steps −

Step 1 − Download the binary distribution from Apache Standard Taglib and unpack the compressed file.

Step 2 − To use the Standard Taglib from its **Jakarta Taglibs distribution**, simply copy the JAR files in the distribution's 'lib' directory to your application's **webapps\ROOT\WEB-INF\lib** directory.

To use any of the libraries, you must include a <taglib> directive at the top of each JSP that uses the library.

Classification of The JSTL Tags

The JSTL tags can be classified, according to their functions, into the following JSTL tag library groups that can be used when creating a JSP page −

- **Core Tags**
- **Formatting tags**
- **SQL tags**
- **XML tags**
- **JSTL Functions**

Core Tags

The core group of tags are the most commonly used JSTL tags. Following is the syntax to include the JSTL Core library in your JSP −

```
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core"
%>
```

Following table lists out the core JSTL Tags −

S.No.	Tag & Description
1	<c:out> Like <%= ... >, but for expressions.
2	<c:set > Sets the result of an expression evaluation in a **'scope'**
3	<c:remove > Removes a **scoped variable** (from a particular scope, if specified).
4	<c:catch> Catches any **Throwable** that occurs in its body and optionally exposes it.
5	<c:if> Simple conditional tag which evalutes its body if the supplied condition is true.

6	<c:choose> Simple conditional tag that establishes a context for mutually exclusive conditiona operations, marked by **<when>** and **<otherwise>**.
7	<c:when> Subtag of **<choose>** that includes its body if its condition evalutes to **'true'**.
8	<c:otherwise > Subtag of **<choose>** that follows the **<when>** tags and runs only if all of the pric conditions evaluated to **'false'**.
9	<c:import> Retrieves an absolute or relative URL and exposes its contents to either the page, a Strin in **'var'**, or a Reader in **'varReader'**.
10	<c:forEach > The basic iteration tag, accepting many different collection types and supporting subsettin and other functionality .
11	<c:forTokens> Iterates over tokens, separated by the supplied delimeters.
12	<c:param> Adds a parameter to a containing **'import'** tag's URL.
13	<c:redirect > Redirects to a new URL.
14	<c:url> Creates a URL with optional query parameters

SECTION D

Enterprise Java Beans: EJB 3.0 Fundamentals, EJB Architecture and Concepts,
Classifications and Configurations of EJBs.
XML:- Introduction and XML Basics, XML Syntax, Declaration, XML Elements and Attributes, XML Parser.

Enterprise Java Beans

Enterprise Java Beans (EJB) is one of the several Java APIs for standard manufacture of enterprise software. EJB is a server-side software element that summarizes business logic of an application. Enterprise Java Beans web repository yields a runtime domain for web related software elements including computer reliability, Java Servlet Lifecycle (JSL) management, transaction procedure and other web services. The EJB enumeration is a subset of the Java EE enumeration.

The EJB enumeration aims to provide a standard way to implement the server-side business software typically found in enterprise applications. Such machine code addresses the same types of problems, and solutions to these problems are often repeatedly re-implemented by programmers. Enterprise Java Beans is assumed to manage such common concerns as endurance, transactional probity and security in a standard way that leaves programmers free to focus on the particular parts of the enterprise software at hand.

To run EJB application we need an application server (EJB Container) such as Jboss, Glassfish, Weblogic, Websphere etc. It performs:

1. Life cycle management

2. Security

3. Transaction management

4. Object pooling

Types of Enterprise Java Beans

There are **three** types of EJB:

1. ***Session Bean:*** Session bean contains business logic that can be invoked by local, remote or webservice client. There are two types of session beans: (i) Stateful session bean and (ii) Stateless session bean.

- **(i) Stateful Session bean :**
  Stateful session bean performs business task with the help of a state. Stateful session bean can be used to access various method calls by storing the information in an instance variable. Some of the applications require information to be stored across separate method calls. In a shopping site, the items chosen by a customer must be stored as data is an example of stateful session bean.

- **(ii) Stateless Session bean :**
  Stateless session bean implement business logic without having a persistent storage mechanism, such as a state or database and can used shared data. Stateless session bean can be used in situations where information is not required to used across call methods.

2. ***Message Driven Bean:*** Like Session Bean, it contains the business logic but it is invoked by passing message.

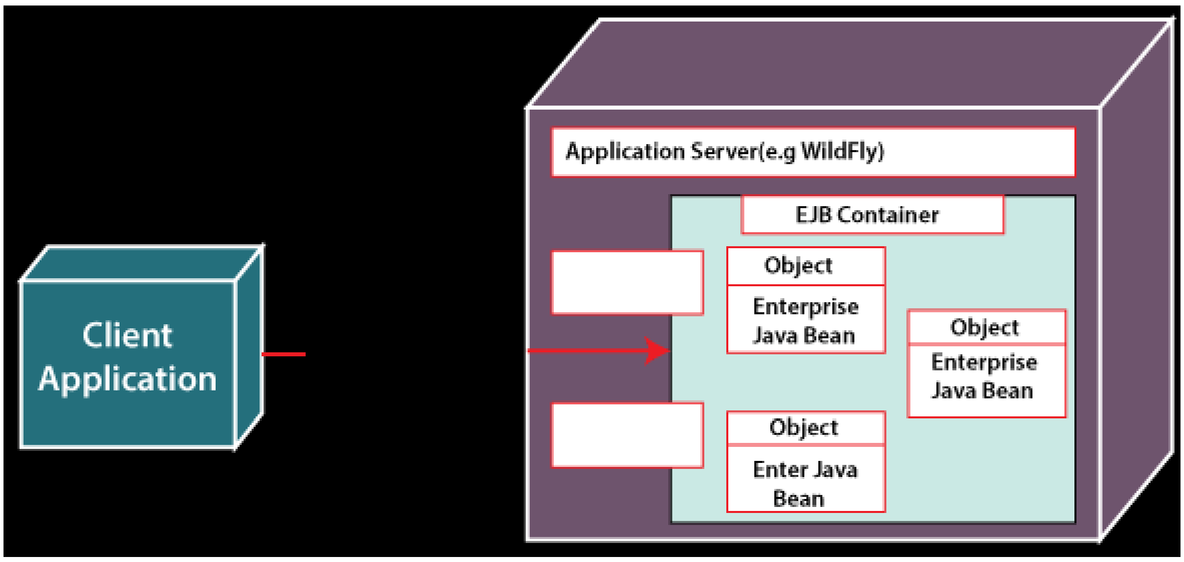
3. ***Entity Bean:*** It summarizes the state that can be remained in the database. It is deprecated. Now, it is replaced with JPA (Java Persistent API). There are two types of entity bean:

- **(i) Bean Managed Persistence :**
  In a bean managed persistence type of entity bean, the programmer has to write the code for database calls. It persists across multiple sessions and multiple clients.

- **(ii) Container Managed Persistence :**
  Container managed persistence are enterprise bean that persists across database. In container managed persistence the container take care of database calls.

EJB Architecture and Concepts,

The EJB architecture has two main layers, i.e., Application Server and EJB Container, based on which the EJB architecture exist. The graphical representation of the EJB architecture is given below.



In the above diagram, the logical representation of how EJBs are invoked and deployed by using RMI(Remote Method Invocation) is defined. The containers of the EJB cannot be self-deployed. In order to deploy the containers, it requires the Application server.

Application Server

In the EJB architecture, the Application server is the outermost layer that holds or contains the Container to be deployed. The application layer plays an important role in executing the application developed using the beans. It provides the necessary environment to execute those applications. Some most popular application servers are Web-logic, Tomcat, JBoss, Web-sphere, Wildfly, and Glass-finish. The main tasks of the application server are:

1. Manage Interfaces
2. Execution of the processes
3. Connecting to the database
4. Manage other resources.

Container

In EJB architecture, the Container is the second outermost layer. It is a very important layer for enterprise beans that are contained in it. For the enterprise bean, the Container provides various supporting services, which are as follows:

- It provides support for transactional services such as registering the objects, assign remote interfaces, purge the instances.
- It provides support for monitoring the object's activities and coordinating distributed components.
- It provides support for security services.
- It provides support for the pooling of resources.
- It provides support for managing the Life-cycle of beans and their concurrency.
- It provides support to concentrate on business logic.

Beans

Java beans of the enterprise are installed in the Container in the same way as a Plain old java object (POJO) is installed and registered to the Container. For developing secured, large scale and robust business applications, beans provide business logic.

XML Basics

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The design goals of XML focus on simplicity, generality, and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, the language is widely used for the representation of arbitrary data structures such as those used in web services.

1. XML stands for extensible Markup Language
2. XML is a markup language like HTML
3. XML is designed to store and transport data
4. XML is designed to be self-descriptive

XML Syntax

XML documents must contain one **root** element that is the **parent** of all other elements:

In this example **<note>** is the root element:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML elements

XML elements can be defined as building blocks of an XML. Elements can behave as containers to hold text, elements, attributes, media objects or all of these.

Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an empty-element tag.

Syntax

Following is the syntax to write an XML element −

```
<element-name attribute1 attribute2>
....content
</element-name>
```

where,

- **element-name** is the name of the element. The *name* its case in the start and end tags must match.
- **attribute1, attribute2** are attributes of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is written as −

```
name = "value"
```

name is followed by an = sign and a string *value* inside double(" ") or single(' ') quotes.

Empty Element

An empty element (element with no content) has following syntax −

```
<name attribute1 attribute2.../>
```

Following is an example of an XML document using various XML element −

```
<?xml version = "1.0"?>
<contact-info>
   <address category = "residence">
      <name>Tanmay Patil</name>
      <company>TutorialsPoint</company>
      <phone>(011) 123-4567</phone>
   </address>
</contact-info>
```

XML Elements Rules

Following rules are required to be followed for XML elements −

- An element *name* can contain any alphanumeric characters. The only punctuation mark allowed in names are the hyphen (-), under-score (_) and period (.).
- Names are case sensitive. For example, Address, address, and ADDRESS are different names.
- Start and end tags of an element must be identical.
- An element, which is a container, can contain text or elements as seen in the above example.

Attributes

The XML attribute is a part of an XML element. The addition of attribute in XML element gives more precise properties of the element i.e, it enhances the properties of the XML element.

Syntax:

```
<element_name attribute1 attribute2 ... > Contents... </element_name>
```

In the above syntax element_name is the name of an element which can be any name. The attribute1, attribute2, … is XML attribute having unique attribute name. Then in the content section, any message can be written and at the end, the element name is ended.

Below some examples are given which illustrate the above syntax:

Example 1:

```
<text category = "message">Hello Geeks</text>
```

In the above example, XML element is text, the **category** is the attribute name and **message** is the attribute value, Attribute name and its value always appear in pair. The attribute name is used without any quotation but attribute value is used in single (‘ ‘) or double quotation (” ”).

Example 2:

```
<text category = "message" purpose ="Greet">Hello Geeks</text>
```

In the above example, two attribute is used with different name. So, in a single element multiple attribute is used having unique attribute name.

But if we use two distinct element then we can use the attribute having the same attribute name. This can be understood with the help of below example:

Example 3:

```
<text category = "message" >Hello Geeks.</text>
<text category = "message">How are you.</text>
```

Attribute Types:

There are three types of attributes described below:

- **String types Attribute:** This type of attribute takes any string literal as a value.
- **Enumerated Type Attribute:** This attribute is further distributed in two types-
  - **Notation Type:** This attribute is used to declares that an element will be referenced to a notation which is declared somewhere else in the XML document.
  - **Enumeration:** This attribute is used to specify a particular list of values which match with attribute values.
- **Tokenized Type Attribute:** This attribute is further distributed in many types:
  - **ID:** This attribute is used to identify the element.
  - **IDREF:** This attribute is used to refer an ID which has already been named for another element.
  - **IDREFS:** This attribute is used to refer all IDs of an element.
  - **ENTITY:** This attribute is used to indicate the attribute which will represent an external entity used in the document.
  - **ENTITIES:** This attribute is used to indicate the attribute which will represent external entities used in the document.

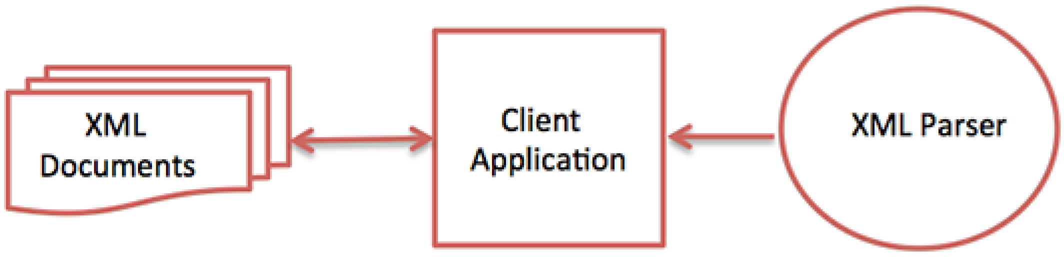
Rules for creating an attribute: There are some rules that should be followed while creating an attribute:

- An attribute should not repeat itself in a single start or empty-element tag.
- An attribute should be declared using the attribute-list declaration in the DTD (Document Type Definition).
- An attribute element is used without any quotation and the attribute value is used in a single (‘ ‘) or double quotation (” “).
- An attribute name and its value should always appear in pair.
- An attribute value should not contain direct or indirect entity references to external entities.

XML parser

XML parser is a software library or a package that provides interface for client applications to work with XML documents. It checks for proper format of the XML document and may also validate the XML documents. Modern day browsers have built-in XML parsers.

Following diagram shows how XML parser interacts with XML document −



The goal of a parser is to transform XML into a readable code.

To ease the process of parsing, some commercial products are available that facilitate the breakdown of XML document and yield more reliable results.

Some commonly used parsers are listed below −

- **MSXML (Microsoft Core XML Services)** − This is a standard set of XML tools from Microsoft that includes a parser.
- **System.Xml.XmlDocument** − This class is part of .NET library, which contains a number of different classes related to working with XML.
- **Java built-in parser** − The Java library has its own parser. The library is designed such that you can replace the built-in parser with an external implementation such as Xerces from Apache or Saxon.

- **Saxon** − Saxon offers tools for parsing, transforming, and querying XML.
- **Xerces** − Xerces is implemented in Java and is developed by the famous open source Apache Software Foundation.