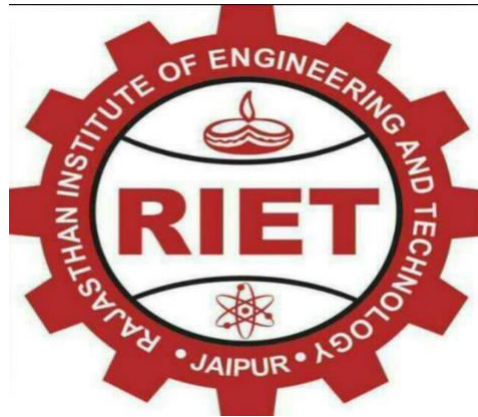


# **RAJASTHAN INSTITUTE OF ENGINEERING AND TECHNOLOGY (2018-2022)**

## **Advanced Java Lab**



## **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Submitted To**

Ms. Shalini Sharma  
Asst. Prof. CSE

**Submitted By**

Nitish K. Mahto  
18ERECS047  
BATCH A2 (V<sup>th</sup> sem)

# INDEX

SR. NO.	EXPERIMENT	PAGE NO.
1	Introduction To Swing, MVC Architecture, Applets, Applications and Pluggable Look and Feel, Basic swing components : Text Fields, Buttons, Toggle Buttons, Checkboxes, and Radio Buttons.	3
2	Java database Programming, java.sql Package, JDBC driver, Network Programming With java.net Package, Client and Server Programs, Content And Protocol Handlers.	7
3	RMI architecture, RMI registry, Writing distributed application with RMI, Naming services, Naming And Directory Services, Overview of JNDI, Object serialization and Internationalization	11
4	J2EE architecture, Enterprise application concepts, n-tier application concepts, J2EE platform, HTTP protocol, web application, Web containers and Application servers	15
5	Server side programming with Java Servlet, HTTP and Servlet, Servlet API, life cycle, configuration and context, Request and Response objects, Session handling and event handling, Introduction to filters with writing simple filter application	18
6	JSP architecture, JSP page life cycle, JSP elements, Expression Language, Tag Extensions, Tag Extension API, Tag handlers, JSP Fragments, Tag Files, JSTL, Core Tag library, overview of XML Tag library, SQL Tag library and Functions Tag library.	26

# Experiment 1

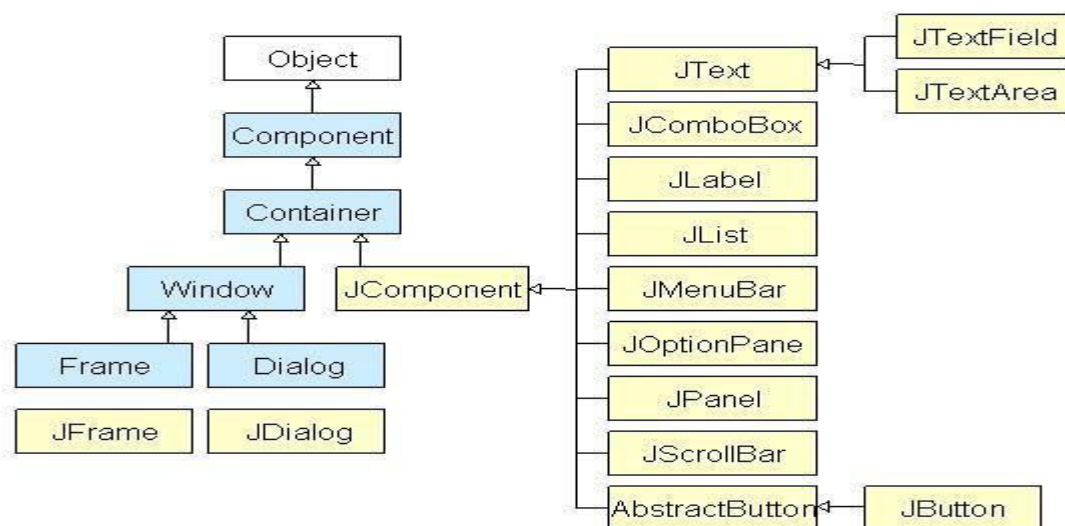
## Objective :-

Introduction To Swing, MVC Architecture, Applets, Applications and Pluggable Look and Feel, Basic swing components : Text Fields, Buttons, Toggle Buttons, Checkboxes, and Radio Buttons

## Swing :

The Swing components are a high-level collection of GUI components, implemented entirely in Java. (Swing was the name of the project that developed these components.) Swing renders components such as buttons and labels in a container such as a frame or a dialog box. The container is a heavyweight component that does rely on the underlying windowing system, but the components themselves do not make use of any user interface components from the underlying windowing system. The container provides the drawing area where various Swing components render themselves. Swing components are called lightweight because they do not have any representation in the underlying windowing system. For example, on a Microsoft Windows system, a Swing button does not use a Windows button. Instead, the button is drawn using graphics primitives such as lines and rectangles. This means that a Swing component should look the same on any system. In fact, as you will see later in this section, Swing allows you to change the look and feel of the user interface (UI) on the fly.

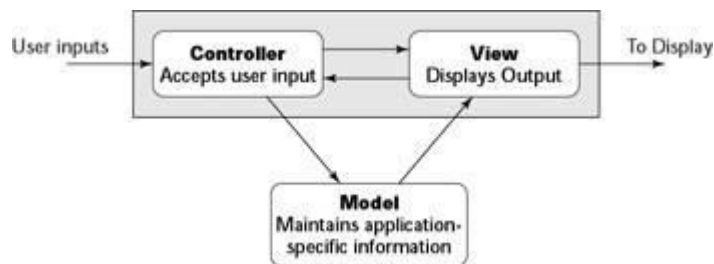
## Basic Swing Components :-



Class	Description
JApplet	Implements a Java applet capable of using Swing components. Any applet that uses Swing classes must be a subclass of JApplet.
JButton	Displays a button with some text and, optionally, an icon
JCheckBox	Displays a check box
JCheckBoxMenuItem	Displays a menu item that can be selected or deselected
JColorChooser	Displays a complex dialog box from which the user can select a color
JComboBox	Displays a combo box that includes a text field and a button to view a drop-down list of items
JComponent	Represents the superclass of most Swing classes
JDesktopPane	Implements a DesktopManager object that can be plugged into a JInternalFrame object
JDialog	Provides a container in which Swing components can be laid out to create custom dialog boxes
JEditorPane	Provides a text component to edit various types of content such as plaintext, HTML, and Rich Text Format (RTF)
JFileChooser	Displays a complex dialog box in which the user can browse through folders and select a file
JFormattedTextField	Provides a single-line text entry and editing area, with the ability to format arbitrary values (extends JTextField)
JFrame	Provides a container in which other Swing components can be laid out. Most standalone GUI applications use a JFrame as the top-level container for laying out other Swing components.
JInternalFrame	Implements a lightweight frame object that can be placed inside a JDesktopPane object
JLabel	Displays a label showing text or an image or both
JLayeredPane	Allows the display of multiple layered panes in a frame so that components can be overlaid
JList	Displays a list of objects (text or icons) from which the user can select one or more items
JMenu	Implements a drop-down menu that can be attached to a menu bar (the menu can show text or images or both)
JMenuBar	Implements a menu bar
JMenuItem	Implements a menu item that appears in a JMenu
JOptionPane	Displays a dialog box that prompts the user for input and then provides that input to the program
JPanel	Provides a lightweight container for arranging other components such as JButton, JLabel, and JComboBox

## The Model-View-Controller (MVC) :-

Architecture and Swing components use a modified form of the Model-View-Controller (MVC) architecture. The classic MVC architecture of Smalltalk-80 breaks an application up into three separate layers: Model-This is the application layer that implements the application's functionality. All application-specific code is in this layer. View-This is the presentation layer that implements whatever is needed to present information from the application layer to the user. In a GUI, the view provides the windows. Controller-This is the virtual terminal layer that handles the user's interactions with the application. This is a graphics library that presents a device-independent interface to the presentation layer.



## Applet :

An applet is a special kind of Java program that runs in a Java enabled browser. This is the first Java program that can run over the network using the browser. Applet is typically embedded inside a web page and runs in the browser.

In other words, we can say that Applets are small Java applications that can be accessed on an Internet server, transported over Internet, and can be automatically installed and run as apart of a web document.

## Lifecycle of Java Applet

Following are the stages in Applet

- 1.Applet is initialized.
- 2.Applet is started

3.Applet is painted.

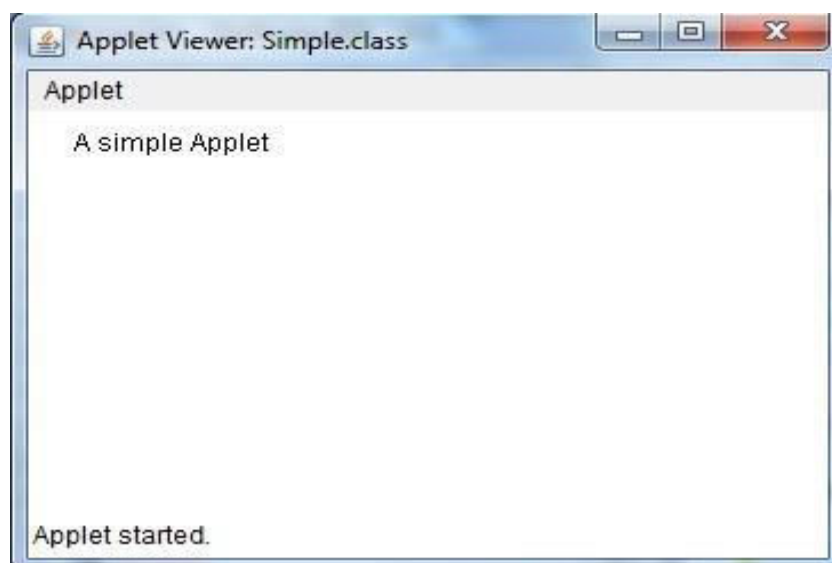
4.Applet is stopped.

5.Applet is destroyed.

## Sample Code Of Applet

```
import java.applet.*;  
  
public class Simple extends Applet{  
    public void paint(Graphics g){  
        g.drawString("A simple Applet", 20, 20);  
    }  
}
```

## Output :



# Experiment 2

## Objective :-

Java database Programming, java.sql Package, JDBC driver, Network Programming With java.net Package, Client and Server Programs, Content And Protocol Handlers.

## Theory :-

### Java network programming :-

Java Networking is a concept of connecting two or more computing devices together so that we can share resources. Java socket programming provides facility to share data between different computing devices. It is best used for sharing resources and for centralized software management .

### Java.net package :-

The java.net package provides many classes to deal with networking applications in Java. Some of classes are given below :

- DatagramPacket
- DatagramSocket
- DatagramSocketImpl
- InterfaceAddress
- JarURLConnection
- MulticastSocket
- InetSocketAddress
- InetAddress

### Protocol and Content handlers :

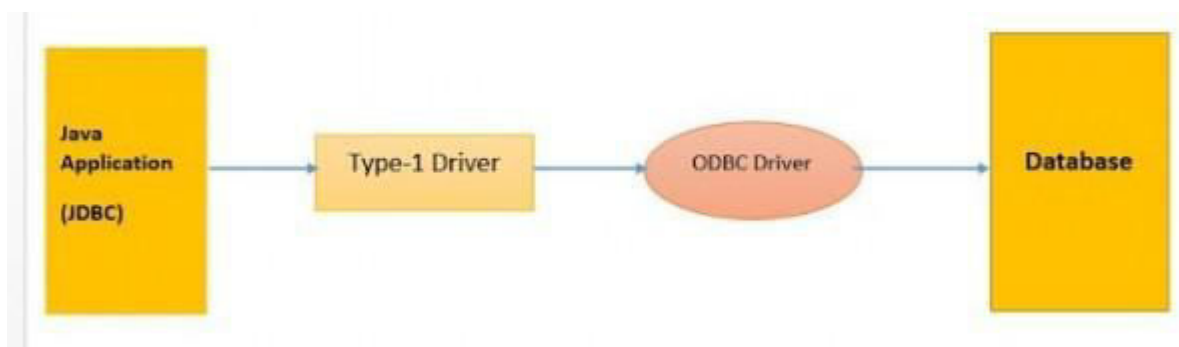
Handlers are classes that extend the capabilities of the standard URL class. A **protocol handler** provides a reference to a java.io.InputStream object (and a java.io.OutputStream object, where appropriate) that retrieves the content of a URL. **Content handlers** take an InputStream for a given MIME type and convert it into a Java object of the appropriate type.

## JDBC Driver :-

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

### • JDBC-ODBC Bridge Driver:

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.

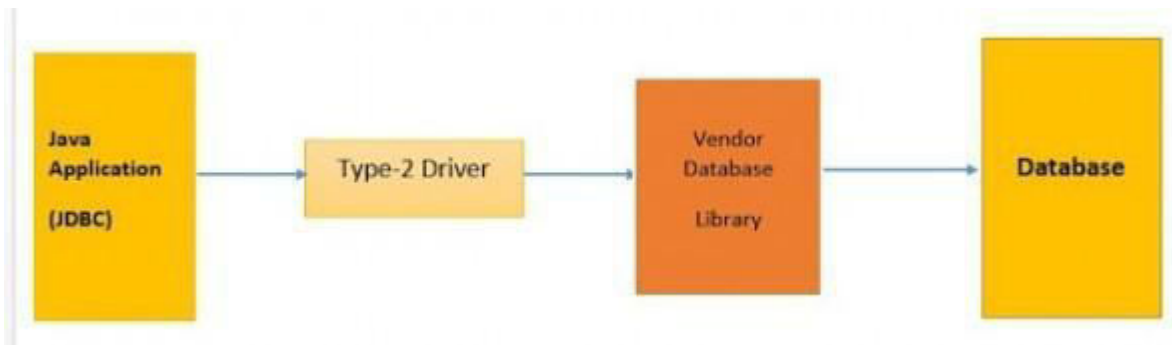


Advantages	Disadvantages
<ul style="list-style-type: none"><li>• Type-1 Driver is database independent driver.</li><li>• it is very easy to use.</li><li>• Not require to install this driver separately(By default in windows)</li></ul>	<ul style="list-style-type: none"><li>• It is the slowest driver.</li><li>• Type-1 driver internally depends upon ODBC driver so ODBC driver concept application only on window machine i.e. platform dependent driver.</li></ul>



### Native Driver :-

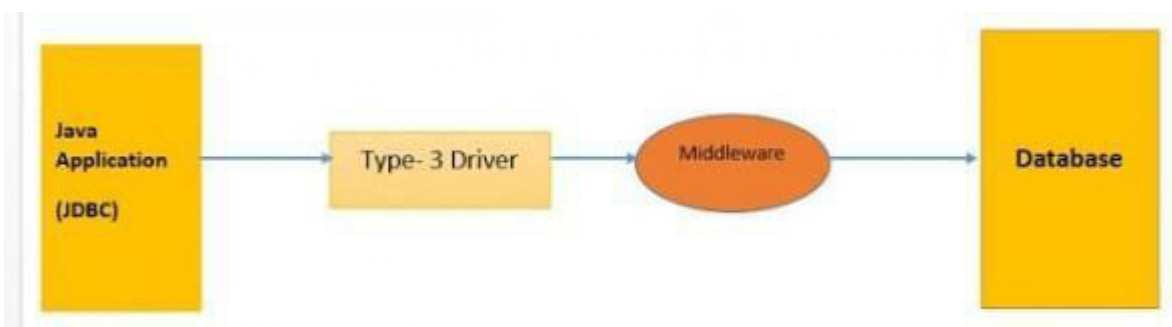
The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.



Advantages	Disadvantages
<ul style="list-style-type: none"><li>• Good performance as compared to the type-1 driver.</li><li>• No ODBC Driver require.</li><li>• Type-2 Drivers are operating system specific and compiled.</li></ul>	<ul style="list-style-type: none"><li>• It is a database dependent driver.</li><li>• It is a platform-dependent driver.</li><li>• Only Oracle provides type-2 Driver</li></ul>

### Network Protocol Driver :-

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.



## Java.sql package :-

The java.sql package contains the entire JDBC API that sends SQL (Structured Query Language) statements to relational databases and retrieves the results of executing those SQL statements. It contains classes and interfaces for JDBC API. A list of popular interfaces of JDBC API are given below:

<ul style="list-style-type: none"><li>• Driver interface</li><li>• Connection interface</li><li>• Statement interface</li><li>• PreparedStatement interface</li><li>• CallableStatement interface</li></ul>	<ul style="list-style-type: none"><li>• ResultSet interface</li><li>• ResultSetMetaData interface</li><li>• DatabaseMetaData interface</li><li>• RowSet interface</li></ul>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------

# Experiment 3

## Objective :-

RMI architecture, RMI registry, Writing distributed application with RMI, Naming services, Naming And Directory Services, Overview of JNDI, Object serialization and Internationalization.

## Theory:-

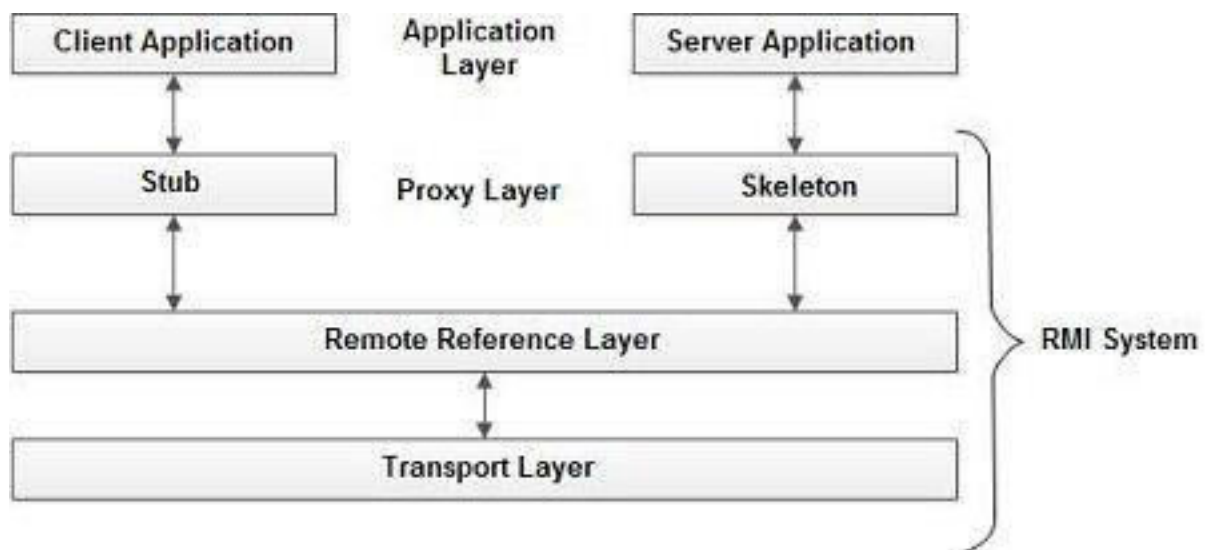
### RMI :

RMI stands for Remote Method Invocation. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM. RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package `java.rmi`.

### Architecture of an RMI Application

In an RMI application, we write two programs, a server program (resides on the server) and a client program (resides on the client).

- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
- The client program requests the remote objects on the server and tries to invoke its methods.



**Architecture of RMI**

- Transport Layer** – This layer connects the client and the server. It manages the existing connection and also sets up new connections.
- Stub**– A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- Skeleton** – This is the object which resides on the server side. Stub communicates with this skeleton to pass request to the remote object.
- RRL(Remote Reference Layer)** – It is the layer which manages the references made by the client to the remote object.

## Steps to implement Interface

1. Defining a remote interface
2. Implementing the remote interface
3. Creating Stub and Skeleton objects from the implementation class using rmic (rmi compiler)
4. Start the rmi registry
5. Create and execute the server application program.
6. Create and execute the client application program.

## Program :-

### Adder.java

```
import java.rmi.*;

public interface Adder extends Remote{
    public int add(int x,int y)throws RemoteException;
}
```

### AdderRemote.java

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject
implements Adder{
    AdderRemote() throws RemoteException{
        super();
    }
    public int add(int x,int y){return x+y;}
}
```

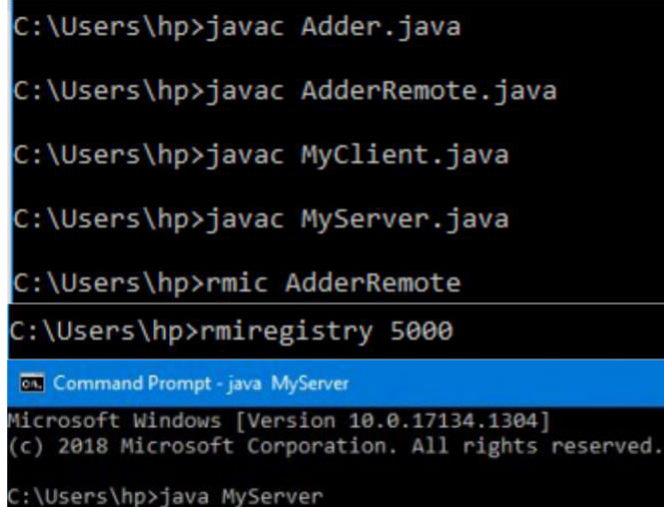
## MyServer.java

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
    public static void main(String args[]){
        try{
            Adder stub=new AdderRemote();
            Naming.rebind("rmi://localhost:5000/abc",stub);
        }catch(Exception e){System.out.println(e);}
    }
}
```

## MyClient.java

```
import java.rmi.*;
public class MyClient{
    public static void main(String args[]){
        try{
            Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/abc");
            System.out.println(stub.add(45,45));
        }catch(Exception e){}
    }
}
```

## Output



```
C:\Users\hp>javac Adder.java
C:\Users\hp>javac AdderRemote.java
C:\Users\hp>javac MyClient.java
C:\Users\hp>javac MyServer.java
C:\Users\hp>rmic AdderRemote
C:\Users\hp>rmiregistry 5000

Command Prompt - java MyServer
Microsoft Windows [Version 10.0.17134.1304]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\hp>java MyServer
```

```
Command Prompt
Microsoft Windows [Version 10.0.17134.1304]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\hp>java MyClient
90

C:\Users\hp>
```

## rmi services :-

need to be hosted in a server process. The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.

Method	Description
public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;	It returns the reference of the remote object.
public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;	It binds the remote object with the given name.
public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;	It destroys the remote object which is bound with the given name.
public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;	It binds the remote object to the new name.
public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;	It returns an array of the names of the remote objects bound in the registry.

# Experiment 4

## Objective :-

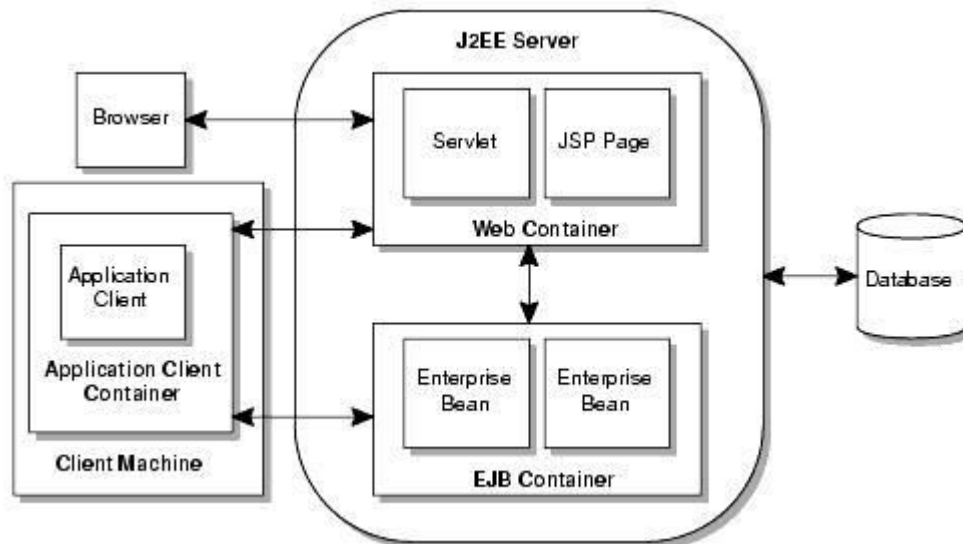
J2EE architecture, Enterprise application concepts, n-tier application concepts, J2EE platform, HTTP protocol, web application, Web containers and Application servers

## Theory:-

### J2EE :-

Java 2 Enterprise Edition is a platform that provides an environment to develop enterprise applications using multitier architecture. This technology is highly intensified provides developer scalable, efficient and faster solutions for information management. This improves the productivity of development and standards for deploying enterprise applications.

### Architecture of J2EE



### J2EE Uses Three Tiers

#### Client Tier:

Client tier consists of user programs that interact with the user for request and response. Clients can be classified as a Web Client and Application Client.

#### 1. Web client :

consists of dynamic web pages of various mark-up languages that are generated by

web components running in web tier or web browser which renders pages received from the server. Web clients are also called as thin clients that usually do not perform things like query database, execute business rules. When using thin client-heavy operations are offloaded to enterprise beans executing in the J2EE server.

Applets: Web pages received from web tier embedded an Applet these run on a web browser. Web components are APIs for creating a web client program. Web components enable the user to design cleaner and more modular applications. They provide a way to separate application programming.

## **2. The application client :**

runs on the client machine and handles the tasks that give richer user interfaces. Typically the GUI is created by Swings or AWT. Application clients can directly access EJBs running in the business tier using an HTTP connection.

## **Middle Tier:**

Middle tier usually contains enterprise beans and web services that distribute business logic for the applications.

### **1. Web Tier /Web Component :**

Web components can be servlets or JSP pages. Servlets can dynamically process the request and generate the responses. Compared to JSP and servlets – servlets are dynamic pages to some extent but JSP pages are static in nature. During application assembly process Client's Static HTML programs and applet, codes are bundled in web tier/ Web Component. Actually these HTML and applets are not considered as elements of web components. Server-side utility classes are also bundled with web component but they are not considered as web components. Web tier might include EJB components for processing user inputs and sends the input to Enterprise bean running in the business tier.

### **2. EJB Tier /EJB Component :**

Enterprise components handle usually business code that is logic to solve particular business domains such as banking or finance are handled by enterprise bean running in the business tier. Enterprise Container receives data from client processes if necessary, sends it to the enterprise information system for storage. Enterprise bean also retrieves data from storage, processes it and sends it back to the client.

## **Enterprise Data Tier:**

Enterprise data is stored in a relational database. This tier contains containers, components and services. This tier consists of database servers, enterprise resource planning systems and other data sources. Resources are typically located on a separate machine than the J2EE Server and accessed by components on the business tier.

Technologies used in EIS Tier:

Java Database Connectivity API (JDBC).

Java Persistence API.

Java Connector Architecture.

Java Transaction API.



## Containers in J2EE Architecture

### 1. Application Client Container:

The container includes a set of classes, libraries, other files that are required to execute client programs in their own JVM. Manages the execution of client components. It also provides services that enable java client program to execute. This container is specific to the EJB container. Compared to other containers in J2EE this container is lightweight.

#### Features:

- Security: responsible for collecting authentication Data such as User Name and Password and sends data over RM/IIOP to the server. The server then processes the data using the JAAS module. Even though authentication techniques are provided by the client container but these are not under the control of the application client.

- Naming: Allows the application clients to use Java Naming and Directory Interface (JNDI).

### 2. Web Container:

Web Container is a component of the web server that interacts with Java servlets. A web container is responsible for managing the servlet lifecycle and mapping URLs. Web container handles a request from Servlets and JSP files and other files that includes server-side code. Web container implements a web component contract of the J2EE architecture. This provides a runtime environment for additional web components security, transaction, deployment, etc.

### 3. EJB Container:

Enterprise Java bean container consists of server components that contain business logic. Provides local and remote access to enterprise beans. EJB container is responsible for creating enterprise bean, binding enterprise bean to the naming services. More than one module can be installed within a single EJB container. It performs transactional actions like – Start Transaction, Commits or Rollback transaction, manages various connection pools for database resources, synchronizing bean instance variables with corresponding data items that are stored in the database.

### 4. Applet Container:

The container where the client's Applet programs will run may be in a web browser or other application programs that support applet programming. Applets are subject to more restrictions due to the sandbox security model this limits access to the client machine. These are normal web pages downloaded from the web servers and executes on the client browser.

# Experiment 5

## Objective :

Server side programming with Java Servlet, HTTP and Servlet, Servlet API, life cycle, configuration and context, Request and Response objects, Session handling and event handling, Introduction to filters with writing simple filter application.

## Theory :

### 1. Server Side Programming with Java Serverlet :-

#### Servlet

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.
- Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language.

#### Common Gateway Interface (CGI)

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

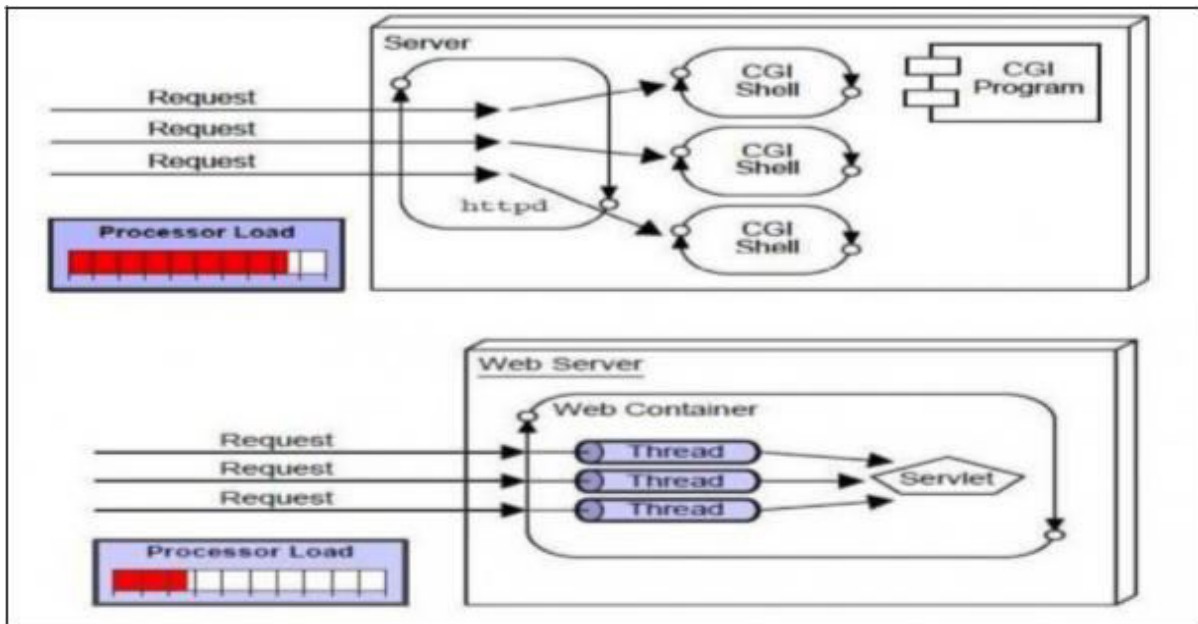
#### Disadvantages of CGI:-

- If the number of clients increases, it takes more time for sending the response.
- For each request, it starts a process, and the web server is limited to start processes.
- It uses platform dependent language e.g. C, C++, perl.

#### Advantages of Servlet:-

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

## Servlet vs CGI



## 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.

## Servlet Interface

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

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

## Methods of Servlet Interface

Method	Description
<code>public void init(ServletConfig config)</code>	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
<code>public void service(ServletRequest request, ServletResponse response)</code>	provides response for the incoming request. It is invoked at each request by the web container.
<code>public void destroy()</code>	is invoked only once and indicates that servlet is being destroyed.
<code>public ServletConfig</code>	returns the object of ServletConfig.

Method	Description
getServletConfig()	
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

## Program Code Implementing Servlet Interface

### First.java

```
import java.io.*;
import javax.servlet.*;
public class First implements Servlet{
    ServletConfig config=null;
    public void init(ServletConfig config){
        this.config=config;
        System.out.println("Servlet is initialized");
    }
    public void service(ServletRequest req,ServletResponse
    res) throws IOException,ServletException{
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        out.print("<html><body>");
        out.print("<b>Hello this is an example of simple
        servlet</b>"); out.print("</body></html>");
    }
    public void destroy(){System.out.println("Servlet is destroyed");}
    public ServletConfig getServletConfig(){return config;}
    public String getServletInfo(){return "copyright 2007-
    1010";} }
```

### web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app >
<servlet>
    <servlet-name>s1</servlet-name>
    <servlet-class>First</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>s1</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
    <display-name>Simple</display-name>
    <welcome-file-list>
```

## web.xml

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

## index.html

```
<a href="hello">Invoke
Servlet</a>
```

## Output

The screenshot displays a web browser window and an IDE console. The browser window shows the URL `http://localhost:8080/Simple/index.html` and the text `Invoke Servlet`. The IDE console shows the output of the Tomcat server, including the startup logs and the message `Servlet is initialized`.

Browser window 1: `http://localhost:8080/Simple/index.html` displays `Invoke Servlet`.

Browser window 2: `http://localhost:8080/Simple/hello` displays `Hello this is an example of simple servlet`.

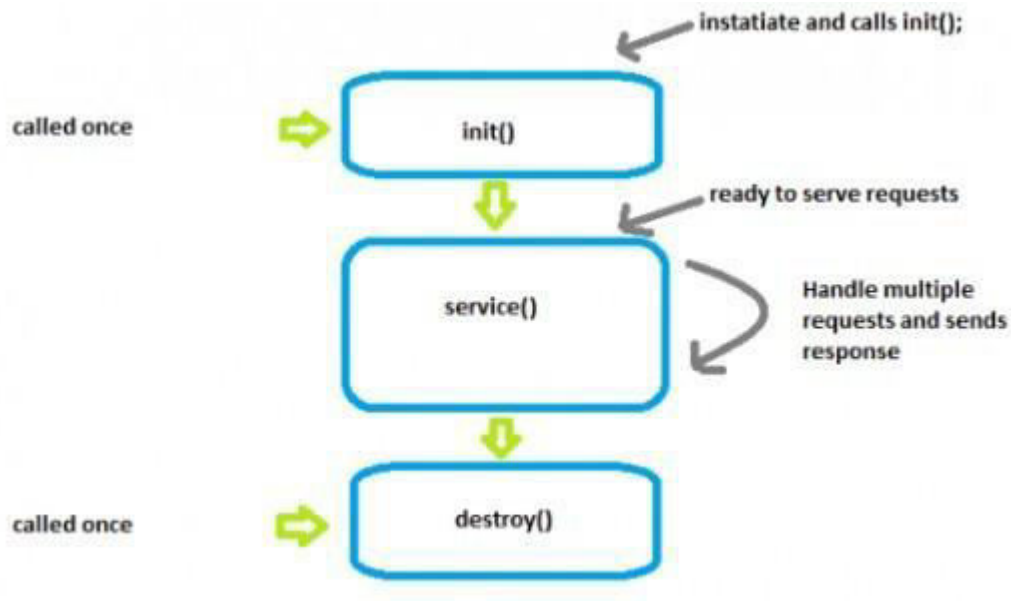
IDE Console:

```
Tomcat v9.0 Server at localhost (5) [Apache Tomcat] C:\Program Files\Java\jdk1.8.0_141\bin\javaw.exe (26-May-2020, 3:18:43 pm)
INFO: Starting service [Catalina]
May 26, 2020 3:18:48 PM org.apache.catalina.core.StandardEngine startInternal
INFO: Starting Servlet engine: [Apache Tomcat/9.0.35]
May 26, 2020 3:18:49 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-nio-8080"]
May 26, 2020 3:18:49 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in [1,287] milliseconds
Servlet is initialized
```

## Life Cycle of a Servlet

A servlet life cycle can be defined as the entire process from its creation till the destruction.

- The servlet is initialized by calling the `init()` method.
- The servlet calls `service()` method to process a client's request.
- The servlet is terminated by calling the `destroy()` method.
- Finally, servlet is garbage collected by the garbage collector of the JVM



## HttpServlet

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

## 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.

## Usage of ServletContext Interface

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

## Methods of ServletContext Interface

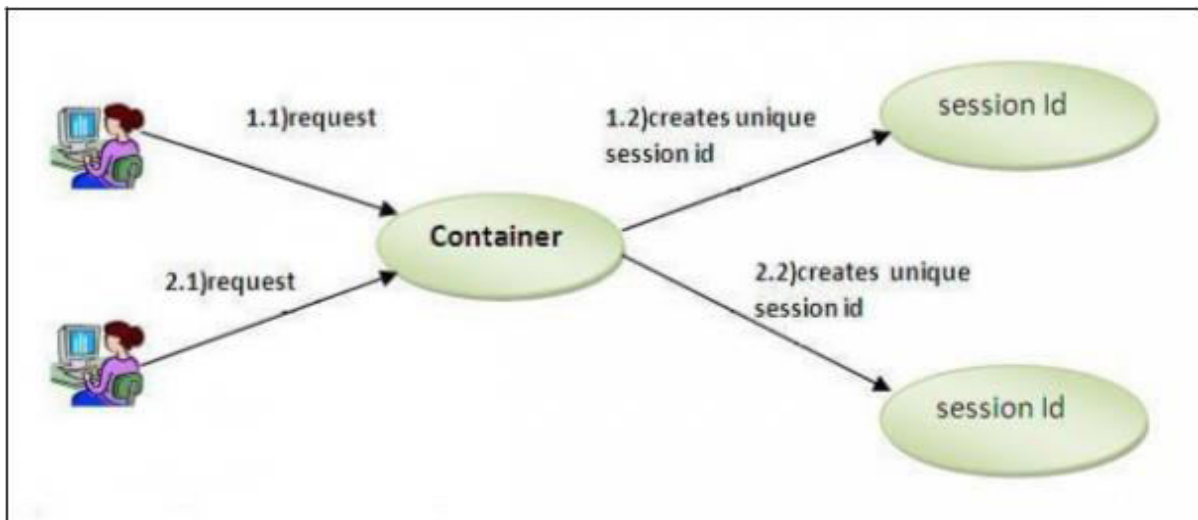
Method	Description
public String getInitParameter(String name)	Returns the parameter value for the specified parameter name
public Enumeration getInitParameterNames()	Returns the names of the context's initialization parameters
public void setAttribute(String name,Object object)	sets the given object in the application scope
public Object getAttribute(String name)	Returns the attribute for the specified name
public Enumeration getInitParameterNames()	Returns the names of the context's initialization parameters as an Enumeration of String objects
public void removeAttribute(String name)	Removes the attribute with the given name from the servlet context

## HttpSession Interface

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:

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



Method	Description
<code>public HttpSession getSession()</code>	Returns the current session associated with this request, or if the request does not have a session, creates one
<code>public HttpSession getSession(boolean create)</code>	Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session
<code>public String getId()</code>	Returns a string containing the unique identifier value
<code>public long getCreationTime()</code>	returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT
<code>public long getLastAccessedTime()</code>	Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
<code>public void invalidate()</code>	Invalidates this session then unbinds any objects bound to it

## Servlet Filter

- A filter is an object that is invoked at the pre-processing and post-processing of a request.
- It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.
- The servlet filter is pluggable, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.



## Usage of Filter

- recording all incoming requests
- logs the IP addresses of the computers from which the requests originate
- conversion
- data compression
- encryption and decryption
- input validation etc.

## Advantages of Filter

- Filter is pluggable.
- One filter don't have dependency onto another resource.
- Less Maintenance

## Filter API

The javax.servlet package contains the three interfaces of Filter API.

- Filter
- FilterChain
- FilterConfig

Method	Description
public void init(FilterConfig config)	init() method is invoked only once. It is used to initialize the filter.
public void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain)	doFilter() method is invoked every time when user request to any resource, to which the filter is mapped.It is used to perform filtering tasks.
public void destroy()	This is invoked only once when filter is taken out of the service.

# EXPERIMENT 6

## Objective :

JSP architecture, JSP page life cycle, JSP elements, Expression Language, Tag Extensions, Tag Extension API, Tag handlers, JSP Fragments, Tag Files, JSTL, Core Tag library, overview of XML Tag library, SQL Tag library and Functions Tag library.

## Theory :

### Introduction to JSP

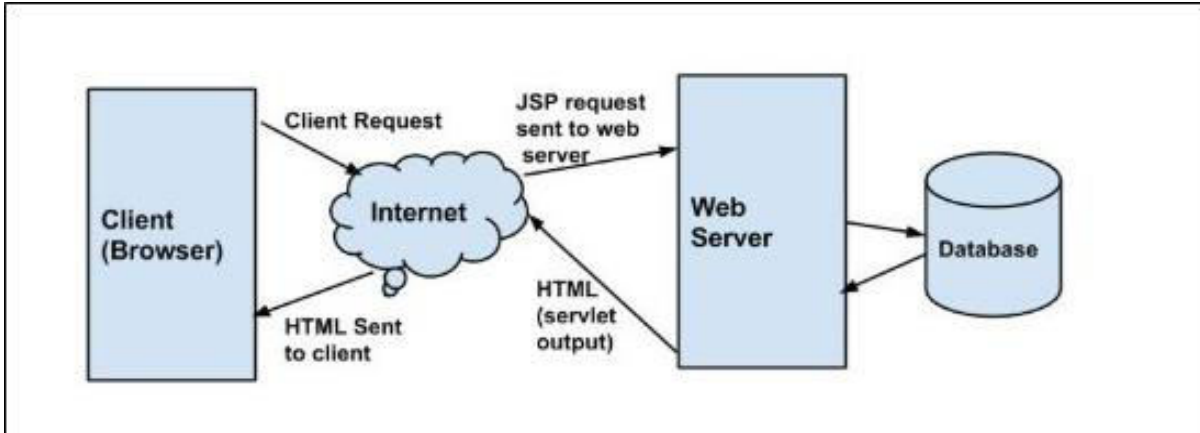
- Java Server Pages or as is normally called JSP is a Java based technology that simplifies the developing of dynamic web sites.
- It is a technology developed by Sun Microsystems, back in 1999.
- JSP pages are HTML pages with embedded code that allows to access data from Java code running on the server.
- JSP contains an extension of .jsp
- JSP provides separation of HTML presentation logic from the application logic.

### Advantages of JSP

- JSP Provides an extensive infrastructure for:
  - Tracking sessions.
  - Managing cookies.
  - Reading and sending HTML headers.
  - Parsing and decoding HTML form data.
- JSP is Efficient: Every request for a JSP is handled by a simple Java thread
- JSP is Scalable: Easy integration with other backend services
- Seperation of roles
  - Developers
  - Content Authors/Graphic Designers/Web Masters

# JSP Architecture

Java Server Pages are part of a 3-tier architecture. A server (generally referred to as application or web server) supports the Java Server Pages. This server will act as a mediator between the client browser and a database.



## JSP Architecture Flow

- The user goes to a JSP page and makes the request via internet in user's web browser.
- The JSP request is sent to the Web Server.
- Web server accepts the requested .jsp file and passes the JSP file to the JSP Servlet Engine.
- If the JSP file has been called the first time then the JSP file is parsed otherwise servlet is instantiated. The next step is to generate a servlet from the JSP file. The generated servlet output is sent via the Internet from web server to users web browser.
- Now in last step, HTML results are displayed on the users web browser.

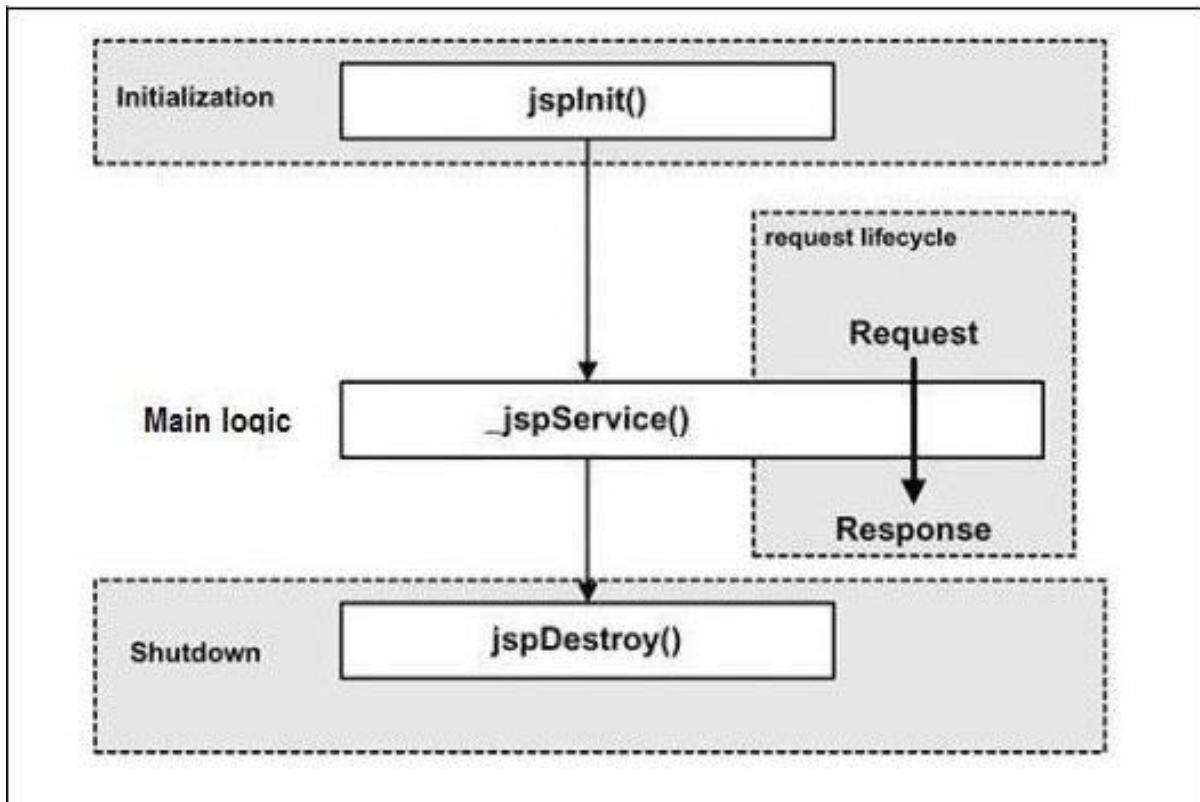
JSP page in a way is just another way to write a servlet without having to be a Java programming expert. Except for the translation phase, a JSP page is handled exactly like a regular servlet. Now that we saw a big picture in JSP architecture section, let's dive a more deeper and understand how a JSP file is treated in a container and what phases it passes through.

## JSP Life Cycle

A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle.

- Compilation
- Initialization
- Execution
- Cleanup



## JSP Compilation

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps –

- Parsing the JSP.
- Turning the JSP into a servlet.
- Compiling the servlet.

## JSP Initialization

When a container loads a JSP it invokes the `jspInit()` method before servicing any requests. If you need to perform JSP-specific initialization, override the `jspInit()` method.

```
public void jspInit(){  
    // Initialization code...  
}
```

Typically, initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the `jspInit` method.

## JSP Execution

- This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.
- Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the `_jspService()` method in the JSP.
- The `_jspService()` method takes an `HttpServletRequest` and `HttpServletResponse` as its parameters as follows

```
void _jspService(HttpServletRequest request,  
HttpServletResponse response) {  
    // Service handling code...  
}
```

The `_jspService()` method of a JSP is invoked on request basis. This is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods, i.e, GET, POST, DELETE, etc.

## JSP Cleanup

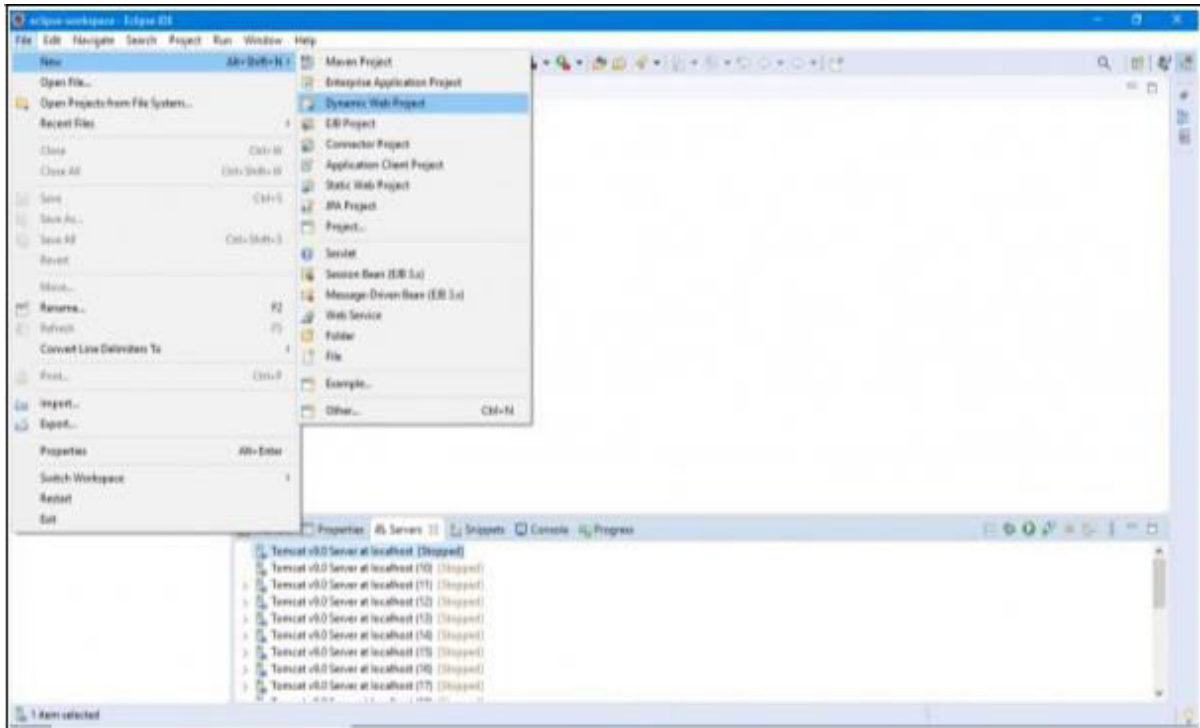
The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The `jspDestroy()` method is the JSP equivalent of the destroy method for servlets. Override `jspDestroy` when you need to perform any cleanup, such as releasing database connections or closing open files.

## Simple JSP Example using Eclipse

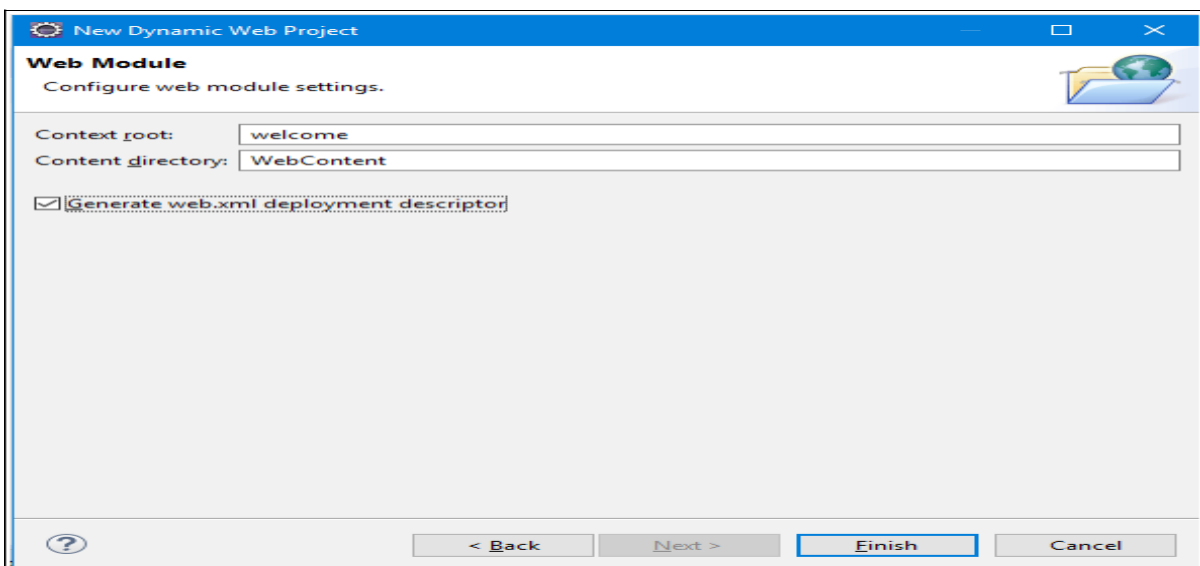
We are writing a simple JSP code using Eclipse IDE and executing it using a web application server (Tomcat).

**Step 1:** Create a Dynamic project as shown

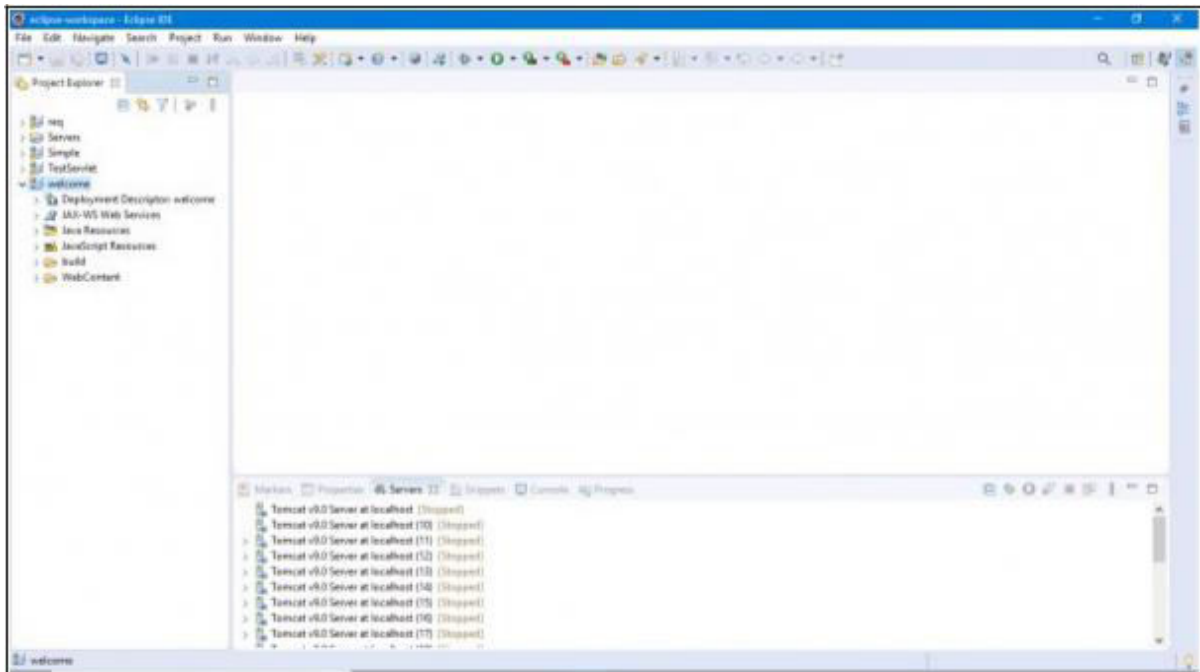


**STEP 2:** Type the project name of your choice and click

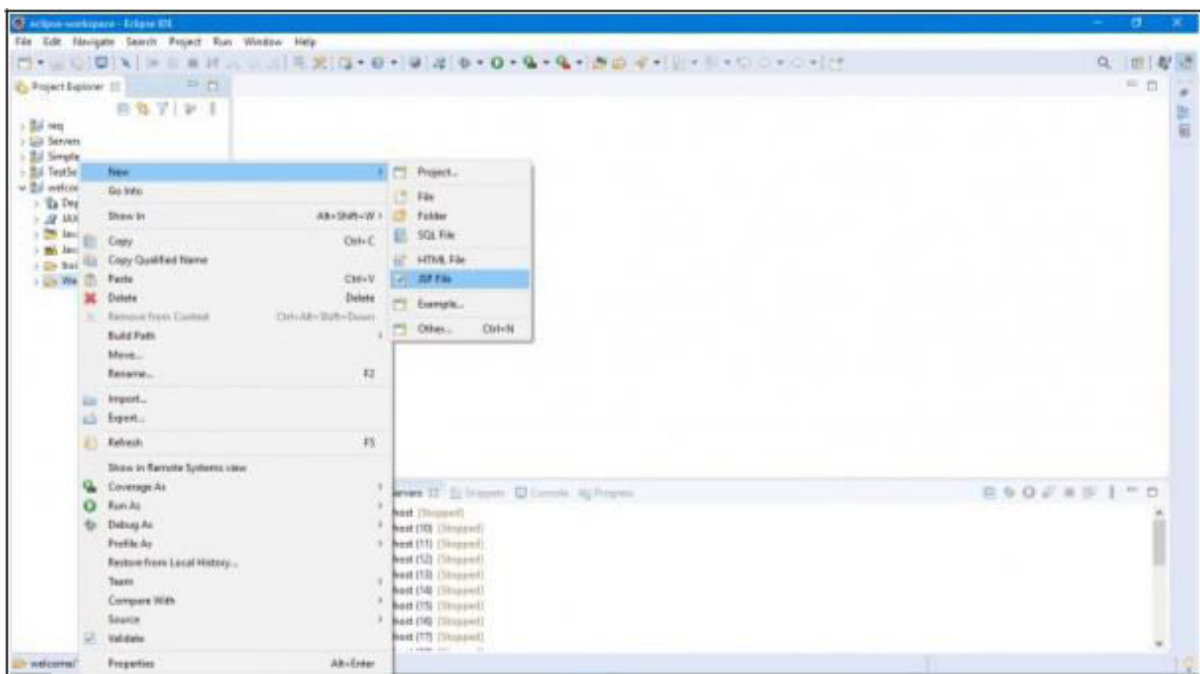
**Step 3:** Now select the checkbox “Generate web.xml deployment descriptor” and click on Finish button.



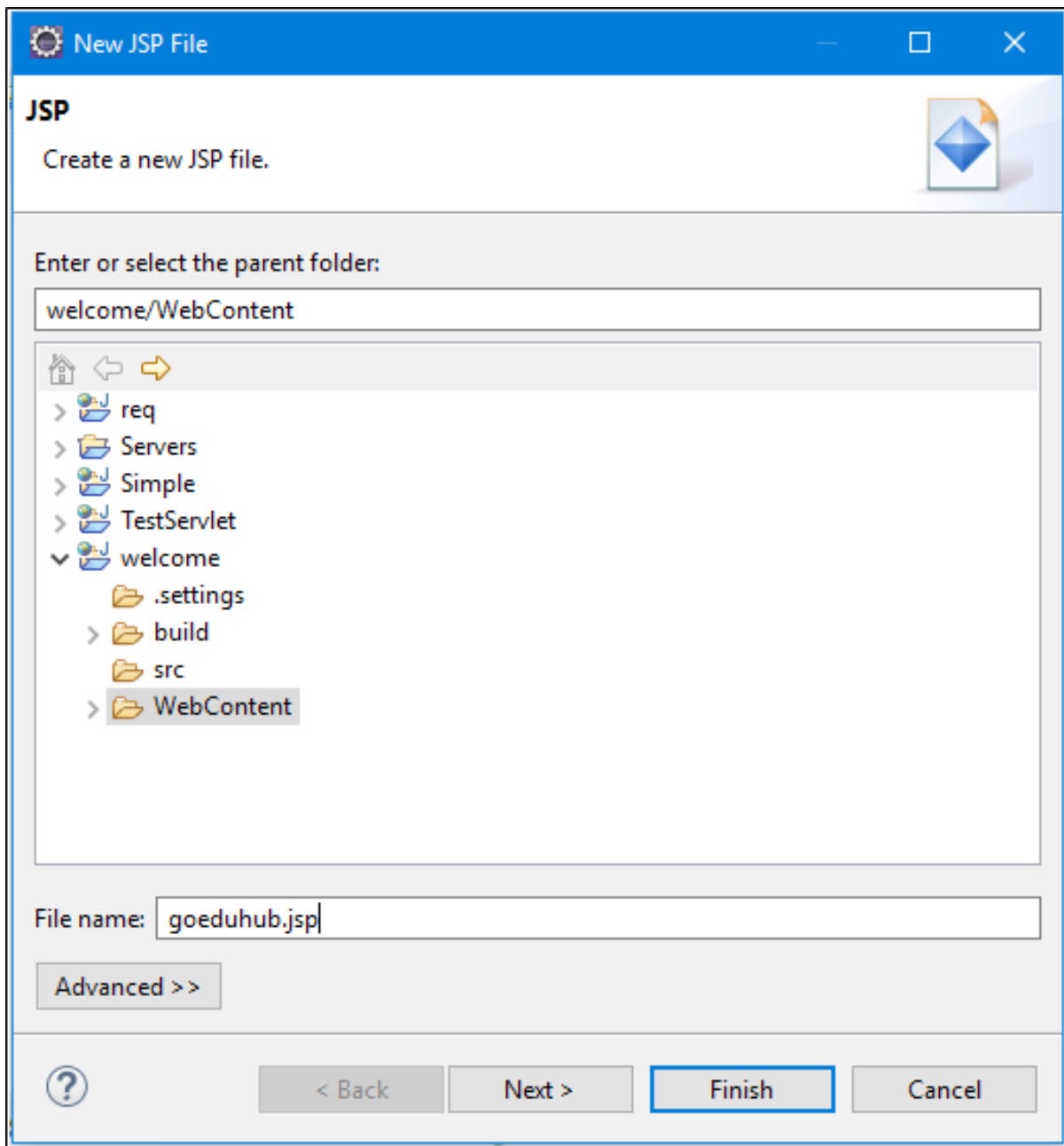
**Step 4:** Once the project is created, you can see the project structure as in the screen below.



**Step 5:** Now create a JSP file under the WebContent folder as shown below.



**Step 6:** Name the file as goeduhub.jsp and click on Finish button.



**Step 7:** Now open the file goeduhub.jsp and paste the below contents in it.

### server.jsp

```
<%@ page language="java"
contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
```



## server.jsp

```
<body>
  <% out.print("Welcome to Goeduhub ");%>
  <p>Login Date: <%= (new java.util.Date())%></p>
</body>
</html>
```

**Step 8:** Now to execute this JSP, we need to configure a web server (in our case it is Tomcat 9). Now select the file goeduhub.jsp and select Run As > Run on Server as shown below.

Action elements are basically predefined functions. Following table lists out the available JSP Actions –

Syntax	Purpose
jsp:include	Includes a file at the time the page is requested.
jsp:useBean	Finds or instantiates a JavaBean.
jsp:setProperty	Sets the property of a JavaBean.
jsp:getProperty	Inserts the property of a JavaBean into the output.
jsp:forward	Forwards the requester to a new page.
jsp:plugin	Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.
jsp:element	Defines XML elements dynamically.
jsp:attribute	Defines dynamically-defined XML element's attribute.
jsp:body	Defines dynamically-defined XML element's body.
jsp:text	Used to write template text in JSP pages and documents.

## Expression Language in JSP

The Expression Language (EL) simplifies the accessibility of data stored in the Java Bean component, and other objects like request, session, application etc. There are many implicit objects, operators and reserve words in EL. The Syntax of EL in a Jsp is:

```
$  
{expression  
}
```

## Basic Operators of EL

Expression language in JSP supports most of operators supported by Java.

Types of Operator:

- Arithmetic operators
- Relational Operators
- Logical Operators
- Empty Operators

### Arithmetic operators

Operator Symbol	Description
+	Addition
–	Unary operator or minus
/ and div	Division
*	Multiplication
% and mod	Remainder

### Relational Operator

This table lists all the relational operators supported by JSP EL.

Operator Symbol	Description
= = and eq	Equals
!= and ne	Not equals
< and lt	Less than

Operator Symbol	Description
> and gt	Greater than
<= and le	Less than or equal
>= and ge	Greater than or equal

## Logical Operator

This table lists all the logical operators supported by JSP EL.

Operator Symbol	Description
&& and AND	and
and or	Or
! and not	not

## Implicit Object in JSP EL

JSP developers can directly use implicit objects in an EL Expression.

Implicit Object	Description
pageContext	It is used to manipulate page attributes and access object request, session etc
param	Request parameter to a single string parameter value.
paramValues	Request parameter to an array of values
Header	Request Header names to a single string header value
headerValues	Request Header names to an array of values
pageScope	Request page-scoped attribute names to the to their values
requestScope	Request request-scoped attribute names to the to their values
sessionScope	Request session-scoped attribute names to the to their values

Implicit Object	Description
e	
applicationScope	Request application-scoped attribute names to the to their values
cookie	Request cookie names to a single Cookie value
initParam	Context initialization parameter names to their string parameter values

## Example Code

### goeduhub.jsp

```
<html>
<head>
<title>Expression Example</title>
</head>
<body><h3>Welcome to Goeduhub</h3>
<%
session.setAttribute("abc","goeduhub");
%>
<a href="welcome.jsp">Visit here</a>
</body>
</html>
```

### welcome.jsp

```
<html>
<head>
<title>Insert title here</title>
</head>
<body>
Value is ${sessionScope.abc }
</body>
</html>
```