APPLET

An **applet** is a Java program that runs in a Web browser.

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Any applet in Java is a class that extends the **java.applet.Applet** class.

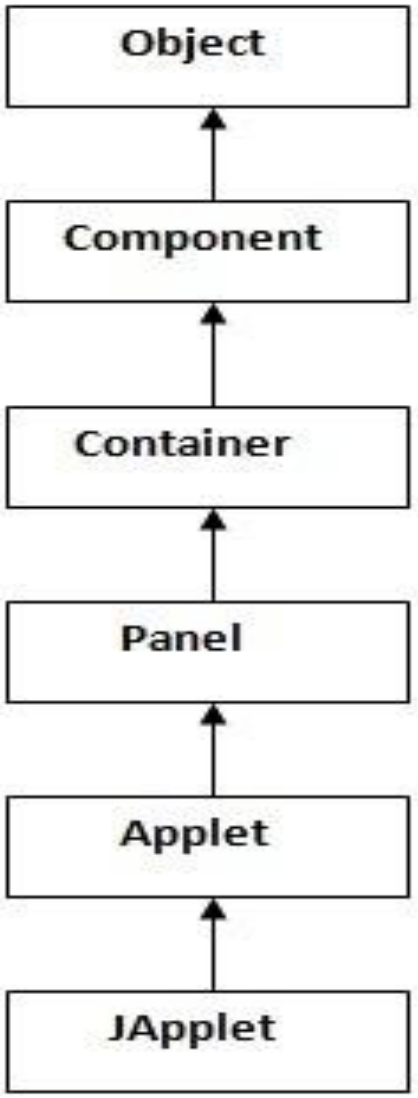
Advantage of Applet

There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

Hierarchy of Applet :

- As displayed in the diagram, Applet class extends Panel. Panel class extends Container, which is the subclass of Component.
- Where Object class is base class for all the classes in java.
- JApplet class is extension of Applet class.



Lifecycle of Applet

There are 5 lifecycle methods of Applet, Those are

public void init(): is used to initialized the Applet. It is invoked only once.

public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet.

public void paint(Graphics g): is invoked immediately after the start() method, and this method helps to create Applet's GUI such as a colored background, drawing and writing.

public void stop(): is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.

public void destroy(): is used to destroy the Applet. It is invoked only once.

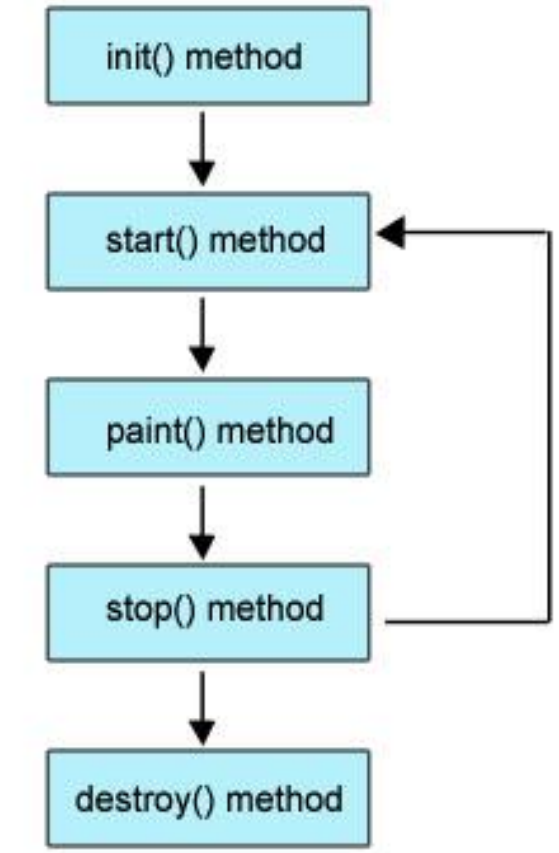


Figure: Life cycle of Applet

Remember:

java.applet.Applet class provides 4 methods (**init,start,stop & destroy**) and **java.awt.Graphics** class provides 1 method (**paint**) to create Applet.

Simple example of Applet

- To execute an Applet, First Create an applet and compile it just like a simple java program.

First.java

```
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
public void paint(Graphics g){
g.drawString("Welcome to Applet",50,150);
}
}
```

Compile:

```
D:\> javac First.java
D:\>
```

After successful compilation, we get **First.class** file.

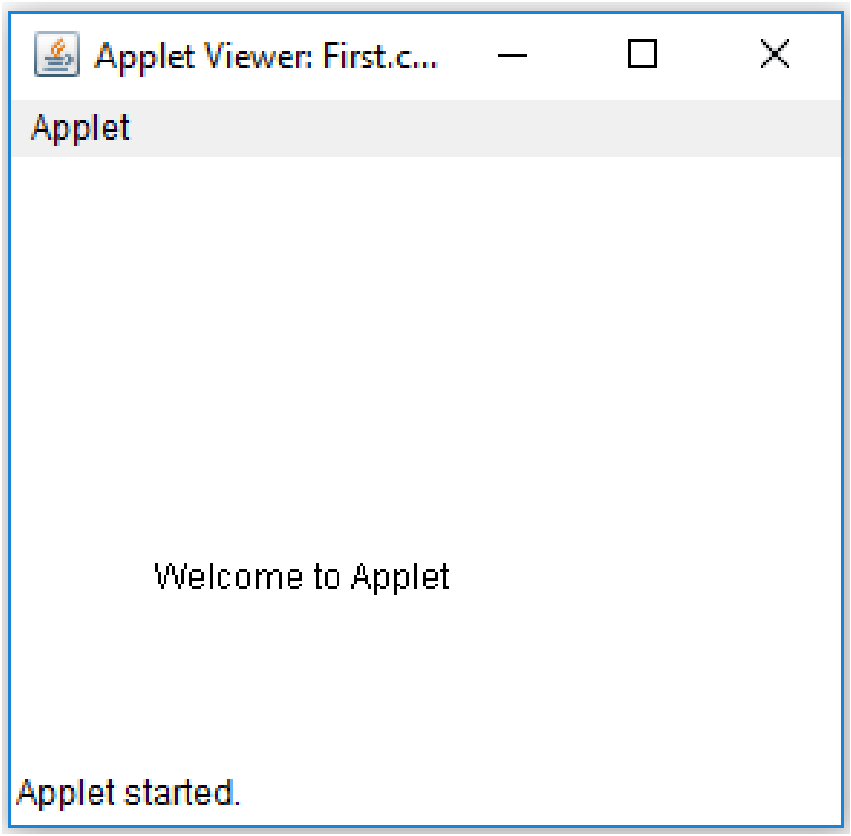
- After that create an html file and place the applet code in html file.

First.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

Execute:

D:\> appletviewer First.html



Displaying Graphics in Applet

- **java.awt.Graphics** class provides many methods for graphics programming.

The Commonly used methods of Graphics class:

- **drawString(String str, int x, int y):** is used to draw the specified string.
- **drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
- **fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
- **drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
- **fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
- **drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
- **setColor(Color c):** is used to set the graphics current color to the specified color.
- **setFont(Font font):** is used to set the graphics current font to the specified font.

Example: GraphicsDemo.java

```
import java.applet.Applet;
import java.awt.*;
public class GraphicsDemo extends Applet
{
public void paint(Graphics g)
{
g.setColor(Color.red);
g.drawString("Welcome",50, 50);
g.drawLine(20,30,20,300);
g.drawRect(70,100,30,30);
g.fillRect(170,100,30,30);
g.drawOval(70,200,30,30);
g.setColor(Color.pink);
g.fillOval(170,200,30,30);
}
}
```

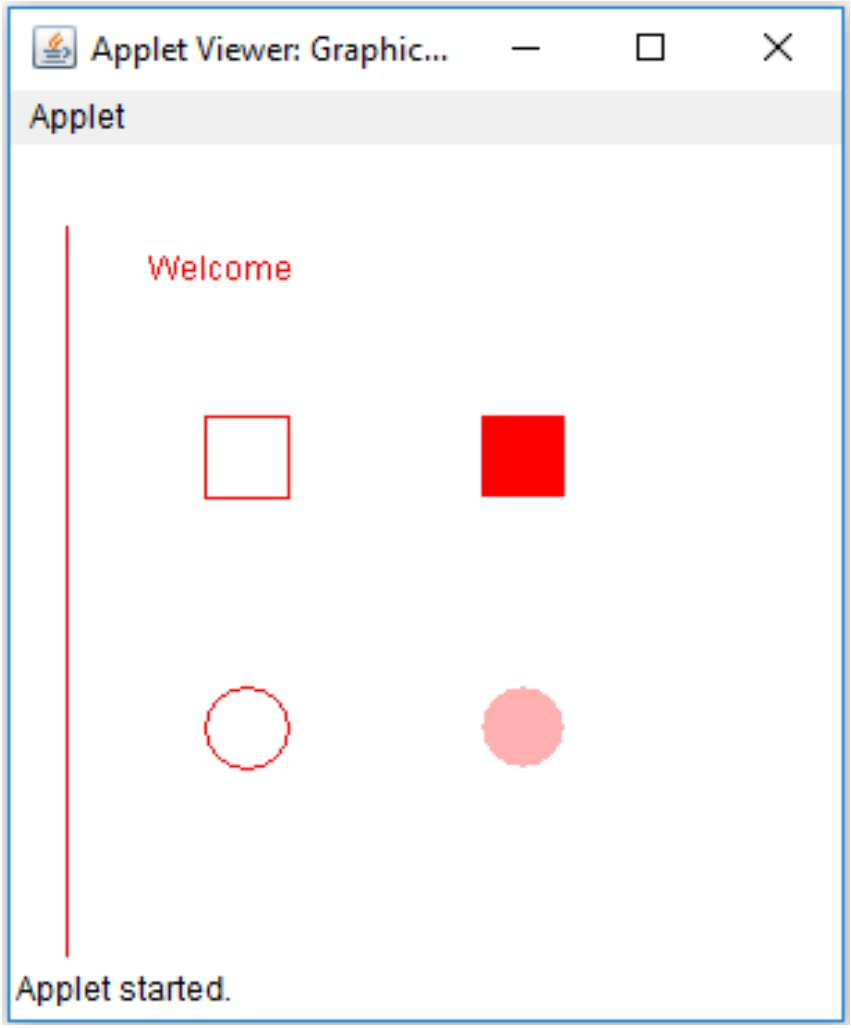
GraphicsDemo.html

```
<html>
<body>
<applet code="GraphicsDemo.class" width="300" height="300">
</applet>
</body>
</html>
```

Execution:

D:\> javac GraphicsDemo.java

D:\> appletviewer GraphicsDemo.html



Components of Applets

- The components of **AWT** are the components of **Applet**,i.e we can use AWT components (Button,TextField,Checkbox, TextArea,Choice & etc.…) in applet.

- As we perform **event handling** in AWT or Swing, we can perform it in applet also.

- Let's see the simple example of components and event handling in applet that prints a message by click on the button.

JApplet Class

- As we prefer Swing to AWT.
- Now we can use JApplet that can have all the controls of swing.
- The JApplet class extends the Applet class.
- The components of **swing** are the components of **JApplet**,i.e we can use swing components (JButton,JTextField,JCheckBox, JTextArea,JList & etc....) in JApplet.

Thank You

GUI Programing with Java

Java Provides the following Frameworks for building GUI-based applications. Those are

- AWT-(Abstract Window Toolkit)
- Swing

AWT-(Abstract Window Toolkit)

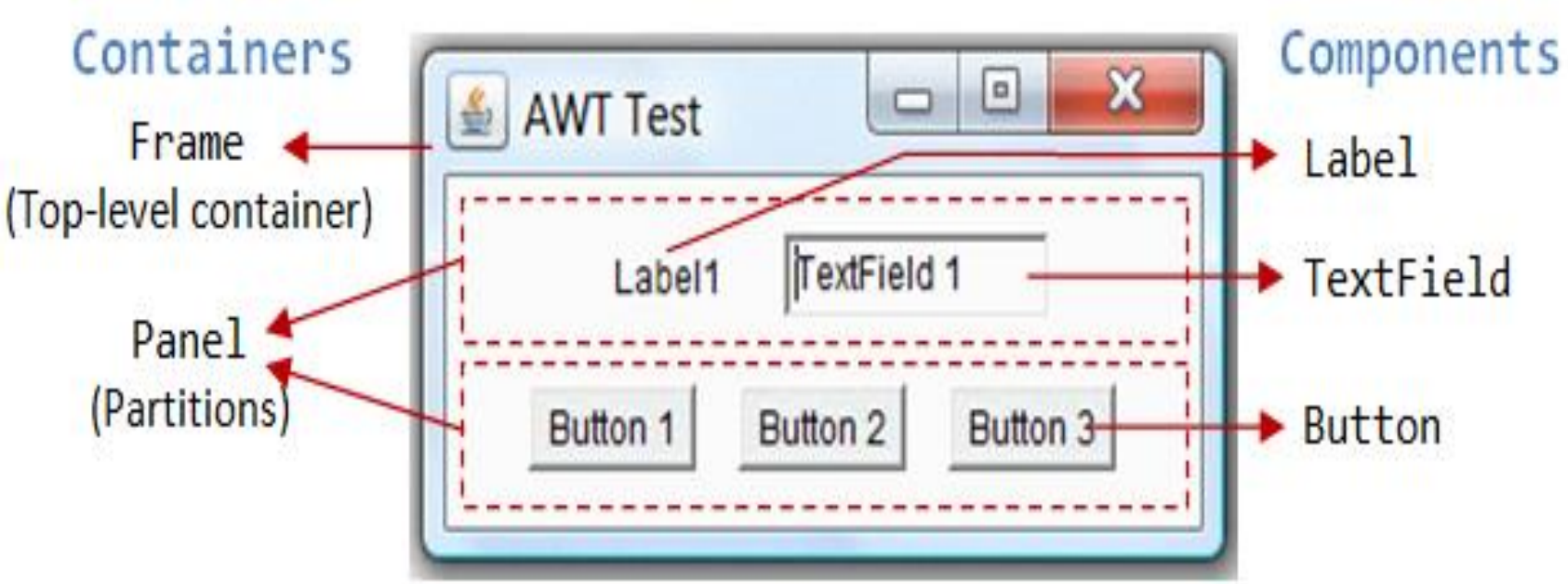
- **AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

- **AWT** (Abstract Window Toolkit) API consists 12 packages along with 370 classes. Fortunately, only 2 packages - **java.awt** and **java.awt.event** - are commonly-used.

- The **java.awt** package provides various classes such as TextField, Label, TextArea, CheckBox, Choice, List etc.

- The **java.awt.event** package provides Event classes, such as ActionEvent, MouseEvent, KeyEvent and WindowEvent.

AWT Containers and Components



- Container:

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc.

The Container class extends Frame and Panel.

- Frame:

  The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

- Panel:

  The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

- Component:

  Component is an abstract class that contains various classes such as Button, Label,Checkbox,TextField,Menu and etc.

Useful Methods of Component class

Method	Description
add(Component c)	inserts a component on this component.
setSize(int width,int height)	sets the size (width and height) of the component.
setLayout(LayoutManager m)	defines the layout manager for the component.
setVisible(boolean status)	changes the visibility of the component, by default false.
setTitle(String text)	Sets the title for component

→ A Simple AWT Application:

- You need a **frame**. There are two ways to create a **frame** in AWT.
- **By extending Frame class** (inheritance)

Ex:

```
class Example extends Frame
{
  ……..
}
```

- **By creating the object of Frame class** (association)

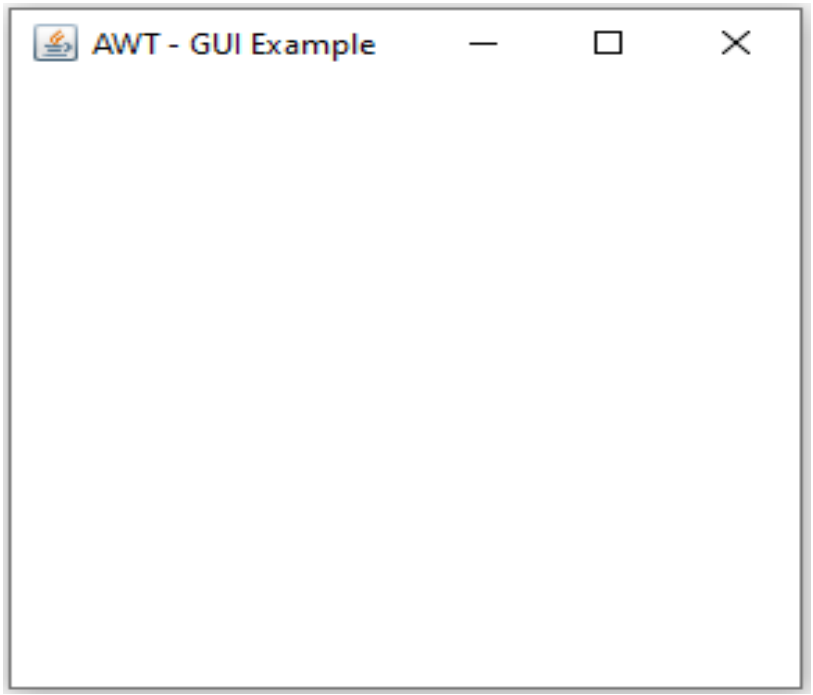
Ex:

```
class Example
{
Frame obj=new Frame();
  ……..
}
```

Example:

SimpleGUIExample.java

```
import java.awt.*;
public class SimpleGUIExample {
public static void main(String[] args) {
        // To create frame
        Frame f=new Frame("AWT - GUI Example");
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        }
}
```



AWT Components or Elements

- Button:

The **Button** class is used to create a labeled button. The application result in some action when the button is pressed.

Syntax:

Button b=**new** Button(“Text");

(Or)

Button b1,b2;

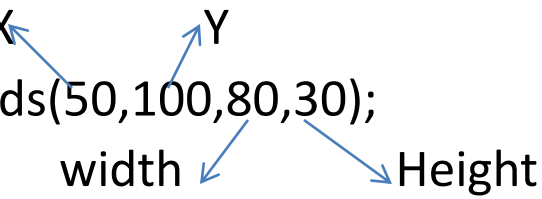
b1=new Button(“Text”);

b.setBounds(50,100,80,30);

setBounds(int x,int y,int width,int height)

This method is used to declare location ,width & height of all components of AWT.

Example: setBounds(50,100,80,30);



- ## Label:

The **Label** class is a component for placing text in a container. It is used to display a single line of read only text.

Syntax:

```
Label l1=new Label("Text");
(or)
Label l1,l2;
l1=new Label("Text");
```

- ## TextField:

The **TextField** class is a text component that allows the user to enter a single line text.

Syntax:

```
TextField t1=new TextField("Text");
(or)
TextField t1,t2;
t1=new TextField("Text");
```

- TextArea :

The **TextArea** class is a multi line region that allows the user to enter multiple line of text.

Syntax:

```
TextArea t1=new TextArea(“Text”);
(or)
TextArea t1,t2;
t1=new TextArea(“Text”);
```

- Checkbox :

The **Checkbox** class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

Syntax:

```
Checkbox c1=new Checkbox(“Text”);
(or)
Checkbox c1,c2;
c1=new Checkbox(“Text”);
```

- Choice :

The **Choice** class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu.

Syntax:

```
Choice c=new Choice();
c.add("Item 1");
c.add("Item 2");
c.add("Item 3");
```

- Scrollbar :

The **Scrollbar** class is used to create a scrollbar.

Syntax:

```
Scrollbar s=new Scrollbar();
```

- AWT doesn't allow the user to close window directly…
- We need code to close the window

```
//Code for  Close  the window
addWindowListener(new WindowAdapter(){
public void windowClosing(WindowEvent we)
{
 System.exit(0);
}
});
```

Note: We need to import one package "java.awt.event.*".

Layout Managers

- In Java, Layout Managers is used for arranging the components in order.

- Layout Manager is an interface that is implemented by all the classes of layout managers.

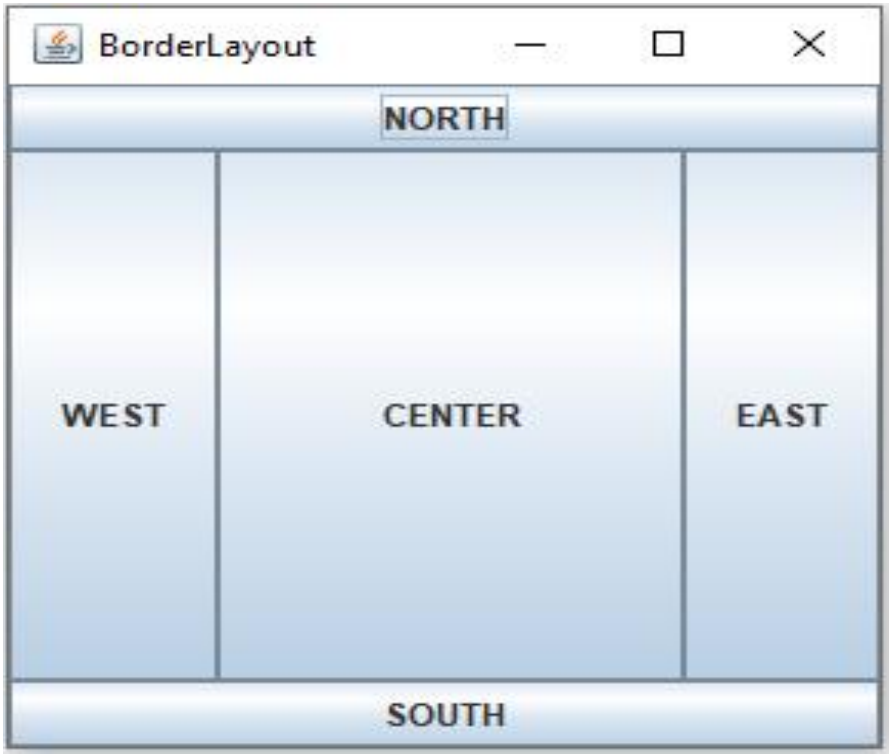
- There are following classes that represents the layout managers

- java.awt.**BorderLayout**
- java.awt.**FlowLayout**
- java.awt.**GridLayout**
- java.awt.**CardLayout**

BorderLayout

- The **BorderLayout** is used to arrange the components in five regions: **NORTH, SOUTH, EAST, WEST** and **CENTER**.

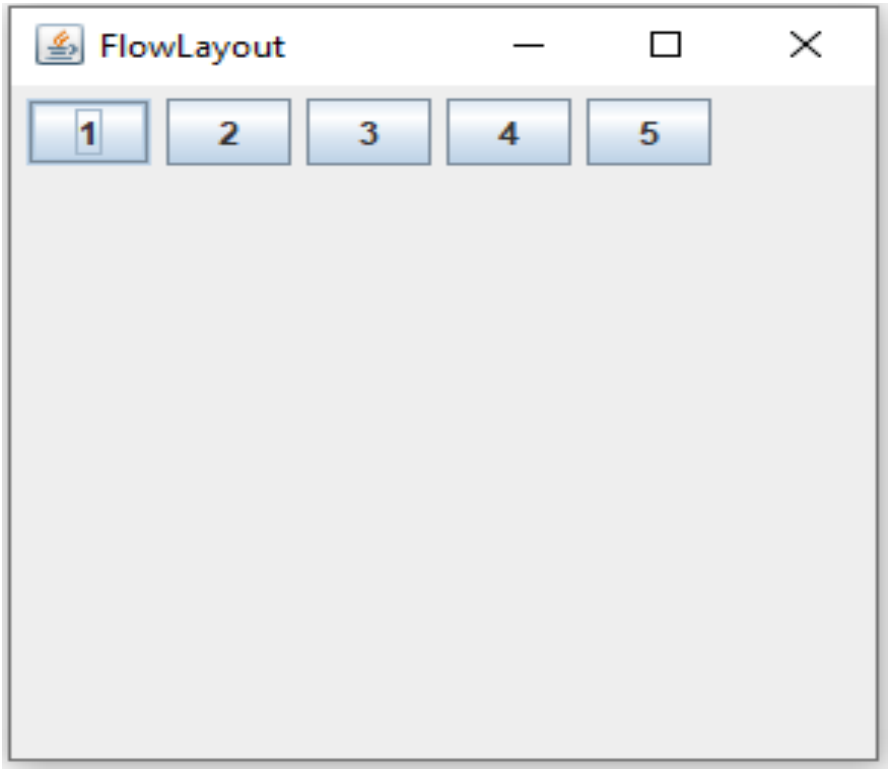
- Each region (area) may contain one component only. It is the **default layout** of frame or window.



FlowLayout

- The **FlowLayout** is used to arrange the components **in a line**, one after another (in a flow).

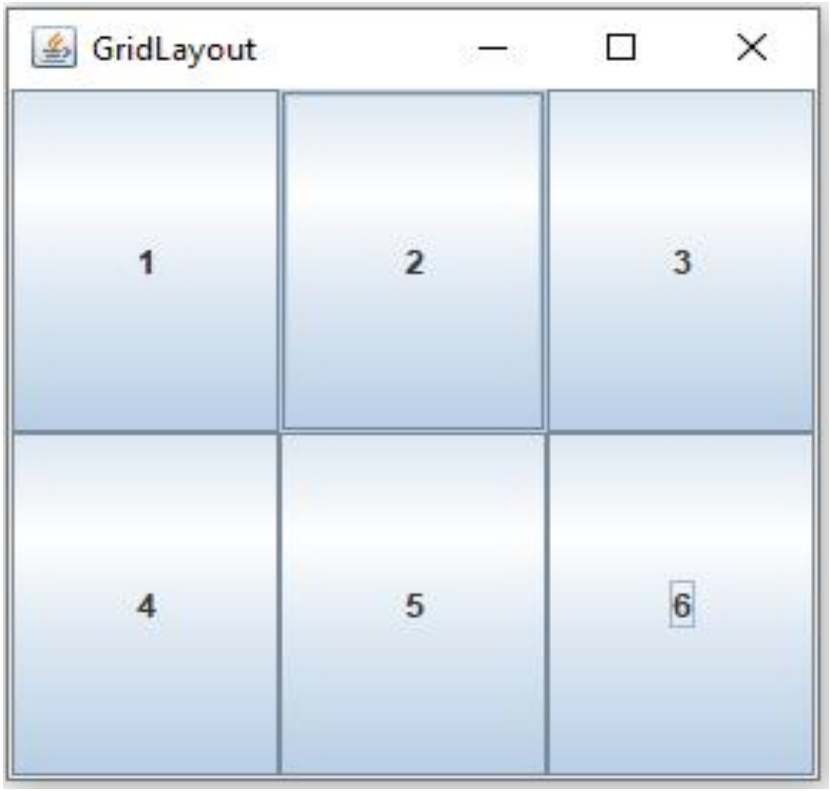
- Fields of FlowLayout are **LEFT,RIGHT** and **CENTER**.



GridLayout

- The **GridLayout** is used to arrange the components in rectangular grid.

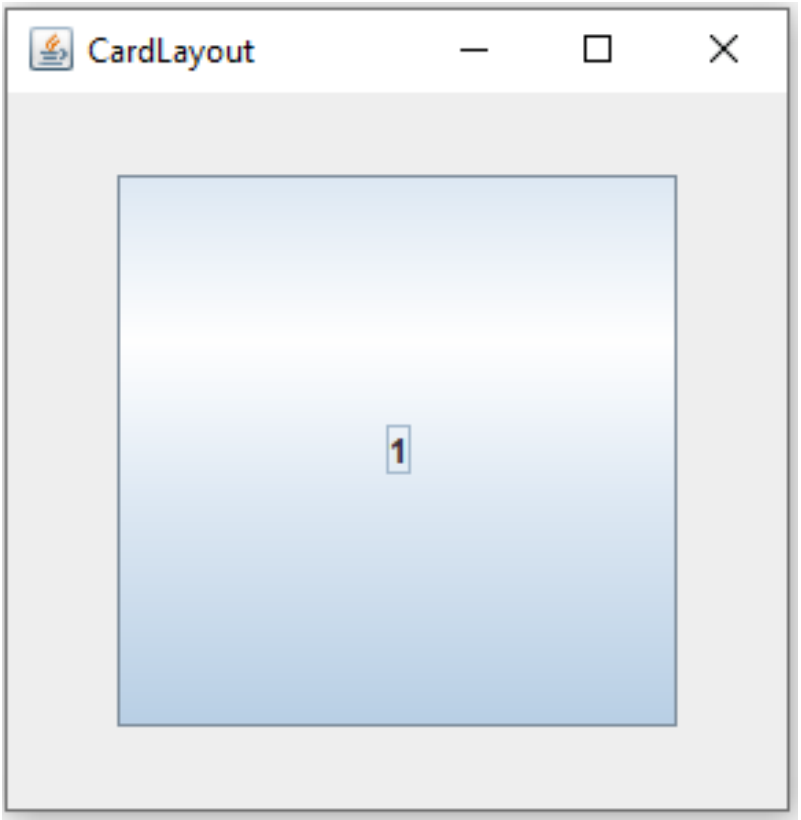
- One component is displayed in each rectangle.



CardLayout

- The **CardLayout** class manages the components in such a manner that only one component is visible at a time.

- It treats each component as a card that is why it is known as CardLayout.



Event Handling

- **Event:** Changing the state of an object(component) is known as an event. For example, click on button, moving mouse etc.

- **Events** are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard and selecting an item from list.

- **Def:** **Event Handling** is the mechanism that controls the event and decides what should happen if an event occurs.

- This mechanism have the code which is known as event handler that is executed when an event occurs.

- The **java.awt.event** package provides many event classes and Listener interfaces for event handling.

Event Classes	Description	Listener Interface
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved,clicked,pressed or released and also when it enters or exit a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener

Steps to perform Event Handling

Following steps are required to perform event handling:

- Register the **component** with the **Listener**.

  By using **addActionListener(ActionListener a)**

  Example:

```
Button b=new Button("Submit");
b.setBounds(100,50,80,30);
b.addActionListener(this);
```

- Provide **event handling code**.

  By using **actionPerformed(ActionEvent e)** method of ActionListener Interface.

  Example:

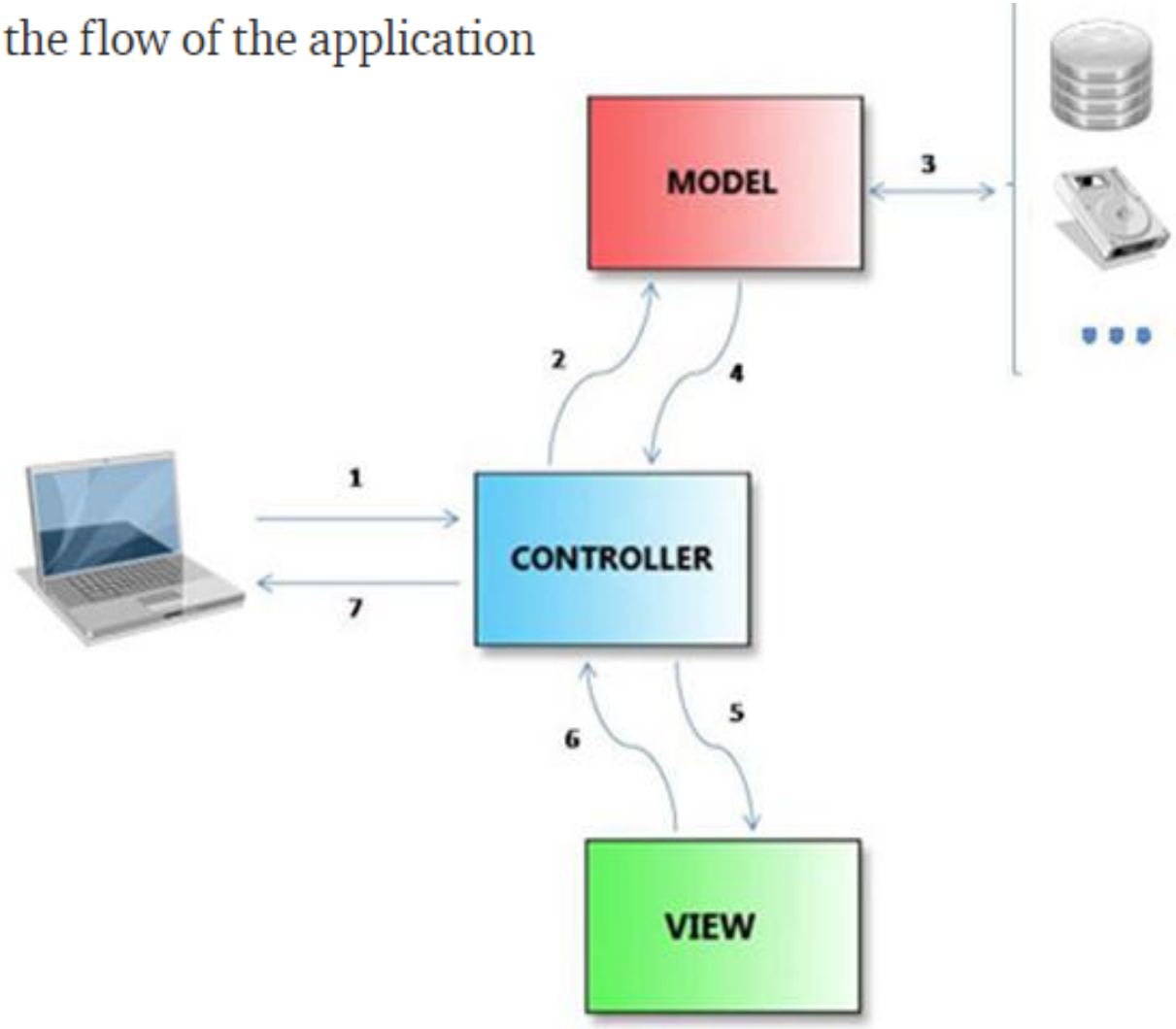
```
Public void actionPerformed(ActionEvent e)
{
tf.setText("welcome");
}
```

Simple Example:

```
import java.awt.*;
import java.awt.event.*;
class EventExample extends Frame implements ActionListener{
TextField tf;
EventExample(){
tf=new TextField(); //create components
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
b.addActionListener(this);//register listener  &passing current instance
add(b);add(tf); //add components and set size, layout and visibility
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new EventExample();
}
}
```

MVC architecture:

- *Model* — Represents the business layer of the application
- *View* — Defines the presentation of the application
- *Controller* — Manages the flow of the application



Swing

- **Swing** is a framework or API that is used to create GUI (or) window-based applications.

- It is an advanced version of AWT (Abstract Window Toolkit) API and entirely written in java.

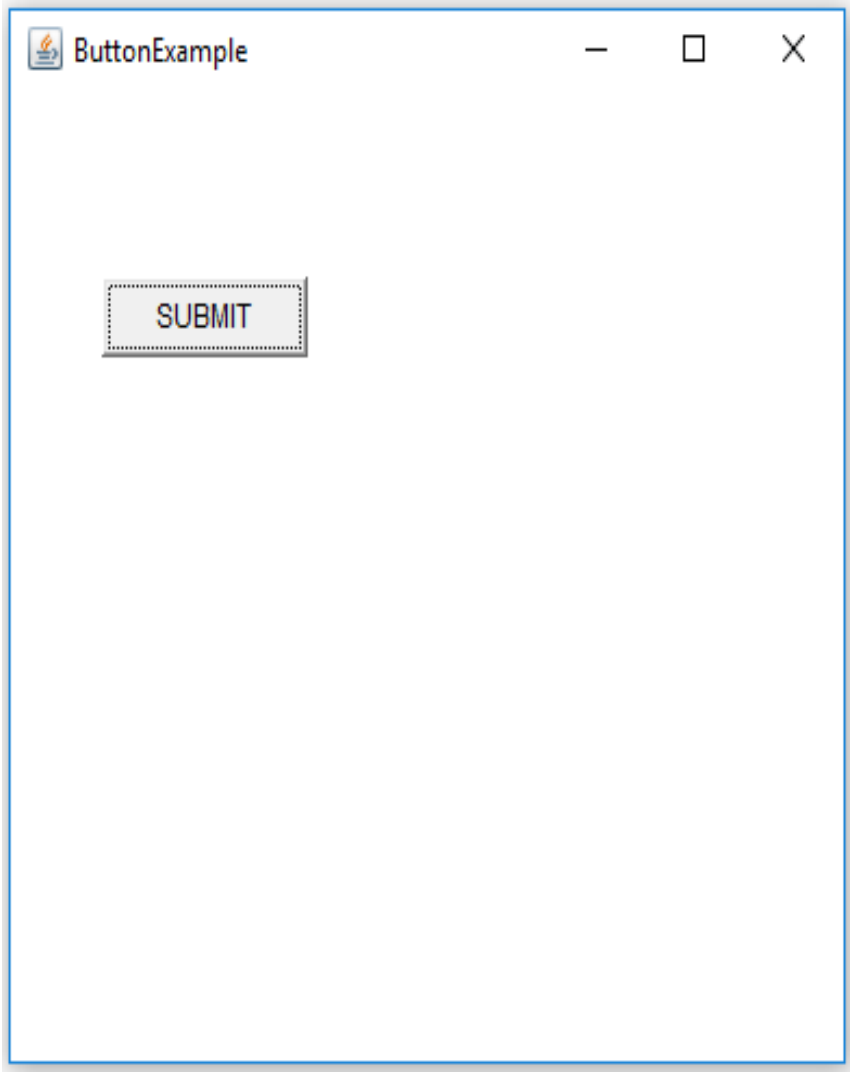
- Unlike AWT, Java Swing provides platform-independent and lightweight components.

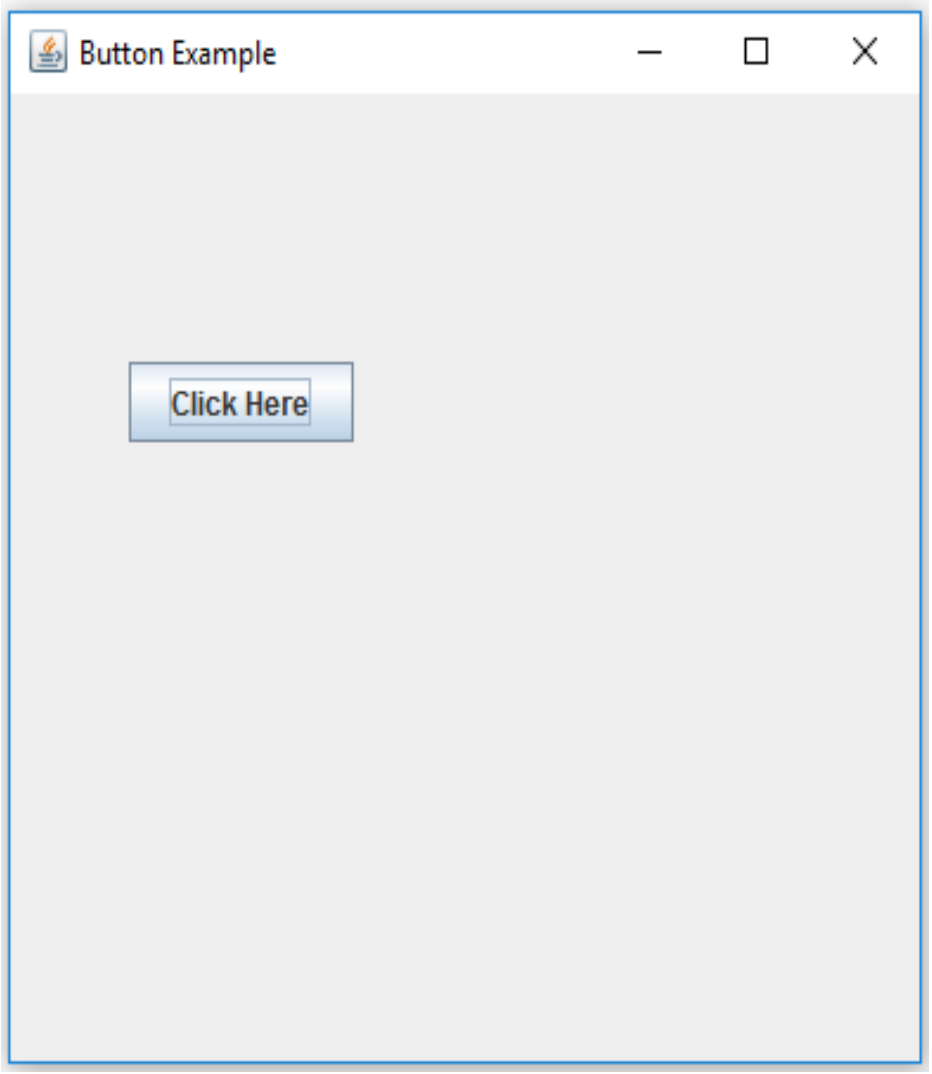
- The **javax.swing** package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JPasswordField etc.

Difference between AWT and Swing

AWT	Swing
AWT is an API to develop GUI applications in Java	Swing is a part of Java Foundation Classes and is used to create various applications.
The components of AWT are platform dependent.	The components of Swing are platform independent.
AWT components requires **java.awt** package.	Swing components requires **javax.swing** package.
AWT components are heavyweight.	Swing components are lightweight.
AWT provides less components than Swing.	Swing provides more components than AWT.
MVC pattern is not supported by AWT.	MVC pattern is supported by Swing.

AWT vs Swing





→To create simple **swing** example, you need **a frame**. In swing, we use **JFrame class** to create a frame.

There are **two ways** to create a frame in swing.

- **By extending JFrame class** (inheritance)

  Ex:

```
class Example extends JFrame
{
  ……..
}
```

- **By creating the object of JFrame class** (association)

  Ex:

```
class Example
{
JFrame obj=new JFrame();
  ……..
}
```

A Simple Swing Example

We can write the code of swing inside the main or constructor.

In Main() Method:

SwingExample.java

```
import javax.swing.*;
public class SwingExample {
public static void main(String[] args) {
// creating a frame
JFrame f=new JFrame("Simple Swing Example");
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
}
```

In Constructor()

```
import javax.swing.*;
class SampleExample extends JFrame {
//constructor
SampleExample() {
setSize(300,300);
setLayout(null);
setVisible(true);
setTitle("SimpleExample");
}
public static void main(String args[]){
SampleExample f=new SampleExample();
}
}
```

Components of Swing

JButton:

The JButton class is used to create a labeled button .The application result in some action when the button is pushed.

Example:

```
JButton b=new JButton(“Text");
(Or)
JButton b1,b2;
b1=new JButton(“Text”);
b.setBounds(50,100,80,30);
```

JLabel:

The JLabel class is a component for placing text in a container. It is used to display a single line of read only text.

Example :

```
JLabel l1=new JLabel(“Text”);
(or)
JLabel l1,l2;
l1=new JLabel(“Text”);
```

JTextField:

The JTextField class is a text component that allows the user to enter a single line of text.

Example :

```
JTextField t1=new JTextField(“Text”);
(or)
JTextField t1,t2;
t1=new JTextField(“Text”);
```

JTextArea :

The JTextArea class is a multi line region that displays text. It allows the editing of multiple line text.

Example :

```
JTextArea t1=new JTextArea(“Text”);
(or)
JTextArea t1,t2;
t1=new JTextArea(“Text”);
```

JCheckBox :

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

Example :

```
JCheckBox c1=new JCheckBox(“Text”);
(or)
JCheckBox c1,c2;
c1=new JCheckBox(“Text”);
```

JPasswordField:

The JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text.

Example :

```
JPasswordField pwd = new JPasswordField();
pwd.setBounds(100,50,80,30);
```

JRadioButton

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

- It should be added in ButtonGroup to select one radio button only.

Example :

```
ButtonGroup bg=new ButtonGroup();
JRadioButton r1=new JRadioButton("Male");
JRadioButton r2=new JRadioButton("Female");
bg.add(r1);bg.add(r2);
```

JComboBox:

The JComboBox class is used to show popup menu of items. Item selected by user is shown on the top of a menu.(like Choice class in AWT)

Example :

```
String country[]={"India","Aus","U.S.A","England","Newzealand"};
JComboBox cb=new JComboBox(country);
cb.setBounds(50, 50,90,20);
```

JPanel:

The JPanel is a simplest container class. It provides space in which an application can attach any other component.

Example :

```
JPanel panel=new JPanel();
panel.setBounds(40,80,200,200);
panel.setBackground(Color.gray);
JButton b1=new JButton("Button 1");
b1.setBounds(50,100,80,30);
panel.add(b1);
```

JDialog:

The JDialog control represents a top level window with a border and a title used to take some form of input from the user.

- Unlike JFrame, it doesn't have maximize and minimize buttons.

Example :

```
JFrame f= new JFrame();
JDialog d=new JDialog(f , "Dialog", true);
JButton b = new JButton ("OK");
d.add(b);
```

JTabbed Pane:

The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon.

Example :

```
JPanel p1=new JPanel();
JPanel p2=new JPanel();
JTabbedPane tp=new JTabbedPane();
tp.add(“Title",p1);
tp.add(“Title",p2);
```

JMenuBar,JMenu & JMenuItem:

- The **JMenuBar** class is used to display menu bar on the window or frame. It may have several menus.

- The **JMenu** class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

- The **JMenuItem** class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

Example :

```
JMenuBar mb=new JMenuBar();
JMenu menu = new JMenu(“Menu");
JMenuItem i1 = new JMenuItem(“item1");
JMenuItem i2 = new JMenuItem(“item2");
JMenuItem i3 = new JMenuItem(“item3");
```

SERVLET Tutorial

A Servlet is the Java technology used to extend and improve Web server functionality. Servlets are able to provide a platform independent way to build web-based applications founded upon components without the performance limitations.

Servlets idea was propsed by **James Gosling** in 1995. **Servlet API** was developed by **James Duncan Davidson** with the first release of the **Java Web Servlet Development Kit (JWSDK)** in June of 1997, and the Java Servlet API 2.1. It was the first used Servlets with Sun´s Java Web Server to develop a website.

Java servlets are a key component of server-side Java development. A servlet is a small, pluggable extension to a server that enhances the server´s functionality. Servlets allow developers to extend and customize any Java-enabled server-a web server, a mail server, an application server, or any custom server-with a hitherto unknown degree of portability, flexibility, and ease.

Introduction to Servlet

Servlets are protocol and platform independent server-side software components, written in Java. They run inside a Java enabled server or application server, such as the WebSphere Application Server. Servlets are loaded and executed within the Java Virtual Machine (JVM) of the Web server or application server, in much the same way that applets are loaded and executed within the JVM of the Web client. Since servlets run inside the servers, however, they do not need a graphical user interface (GUI). In this sense, servlets are also faceless objects.

The Power of Servlets

What makes servlets a viable choice for web development? We believe that servlets offer a number of advantages over other approaches, including: portability, power, efficiency, endurance, safety, elegance, integration, extensibility, and flexibility.

What is Servlet

Servlets are small programs that execute on the server side of a Web connection and dynamically extend the functionality of a Web server just as applets.

- 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.

Servlet Architecture

The architecture, is the communication interface, protocol used, requirements of client and server, the programming with the languages and software involved. Basically, it performs the below-mentioned tasks.

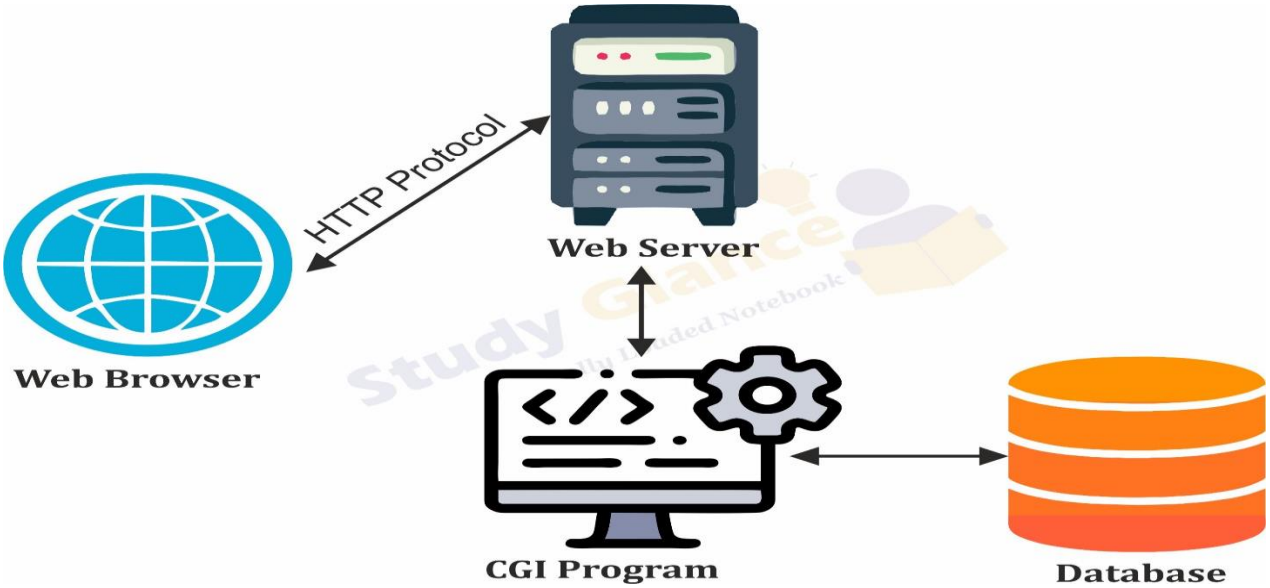
1. The clients send the request to the webserver.
2. The web server receives the request.
3. The web server passes the request to the corresponding servlet.
4. The servlet processes the request and generates the response in the form of output.
5. The servlet sends the response back to the webserver.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

Servlets more closely resemble Common Gateway Interface (CGI) scripts or programs than applets in terms of functionality. As in CGI programs, servlets can respond to user events from an HTML request, and then dynamically construct an HTML response that is sent back to the client.

Common Gateway Interface

The Common Gateway Interface (CGI) is the standard process that uses a set of rules to propagate the user´s request to the web resources such as web server or web application program and respond to the user through the web interface. CGI includes several working scripts and programs for web communication. CGI is the mechanism that is part of the Hypertext Transport Protocol (HTTP). One of the examples of CGI flow is the Web browsers send the forms data to the backend server, and CGI connects to the application program on the web-server and the program response to the web browser.

CGI Architecture



Features of CGI

- It is a very well defined and supported standard.
- CGI scripts are generally written in either Perl, C, or maybe just a simple shell script.
- CGI is a technology that interfaces with HTML.
- CGI is the best method to create a counter because it is currently the quickest
- CGI standard is generally the most compatible with today´s browsers

Advantages of CGI

- The advanced tasks are currently a lot easier to perform in CGI than in Java.
- It is always easier to use the code already written than to write your own.
- CGI specifies that the programs can be written in any language, and on any platform, as long as they conform to the specification.
- CGI-based counters and CGI code to perform simple tasks are available in plenty.

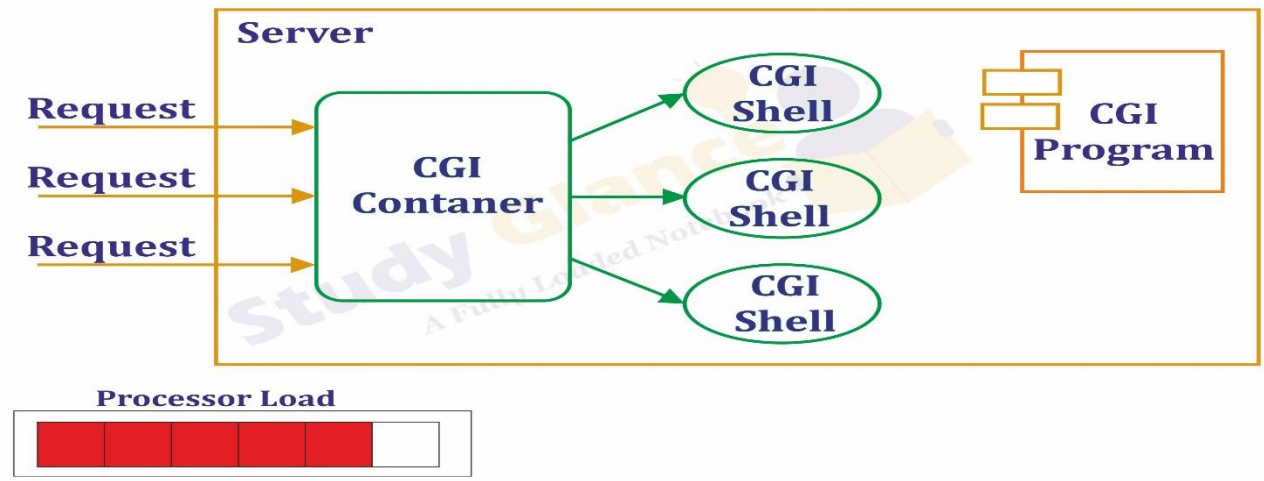
Disadvantages of CGI

- In Common Gateway Interface each page load incurs overhead by having to load the programs into memory.
- Generally, data cannot be easily cached in memory between page loads.
- There is a huge existing code base, much of it in Perl.
- CGI uses up a lot of processing time.

Common Gateway Interface(CGI) vs Servlet

Common Gateway Interface

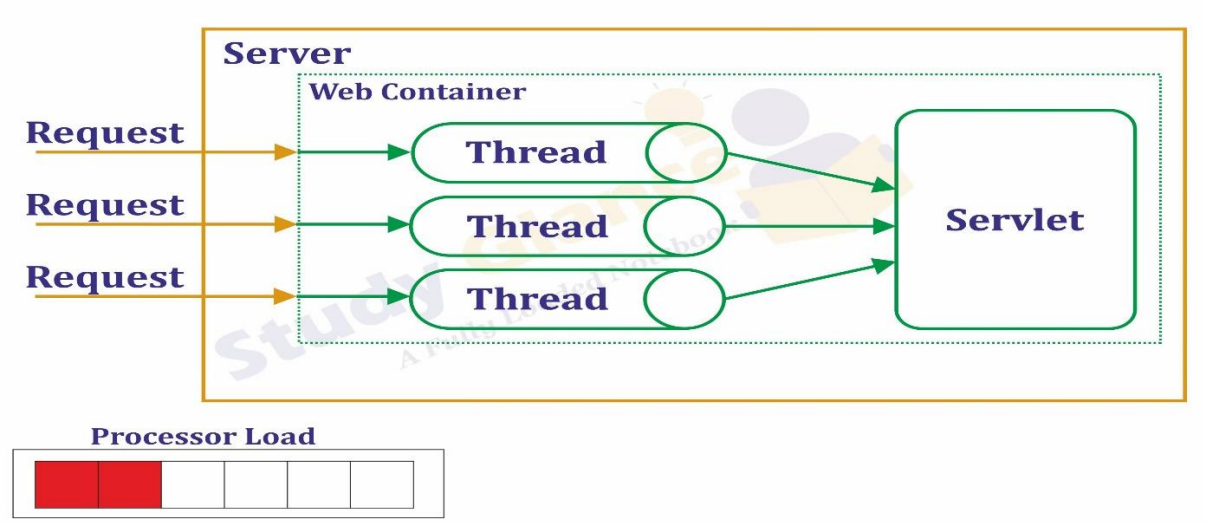
The Common Gateway Interface (CGI) provides the middleware between servers and external databases or information sources. The Web server typically passes the form information to a small application program that processes the data and may send back a confirmation message. This process or convention for passing data back and forth between the server and the application is called the common gateway interface. The following image describes how a web server acts as an intermediate between the CGI program and the client browser.



A new process is started for each client request. The creation of the process for every such request requires time and significant server resources which limits the number of requests a server can handle

Servlet

Servlets are server based java application that can link directly to the Web server. Servlets are thread based applications and work on server level and they share data among each other. All the programs of Servlets are written in JAVA and they get to run on JAVA Virtual Machine. The following image describes how a request from clients is served with the help of threads:

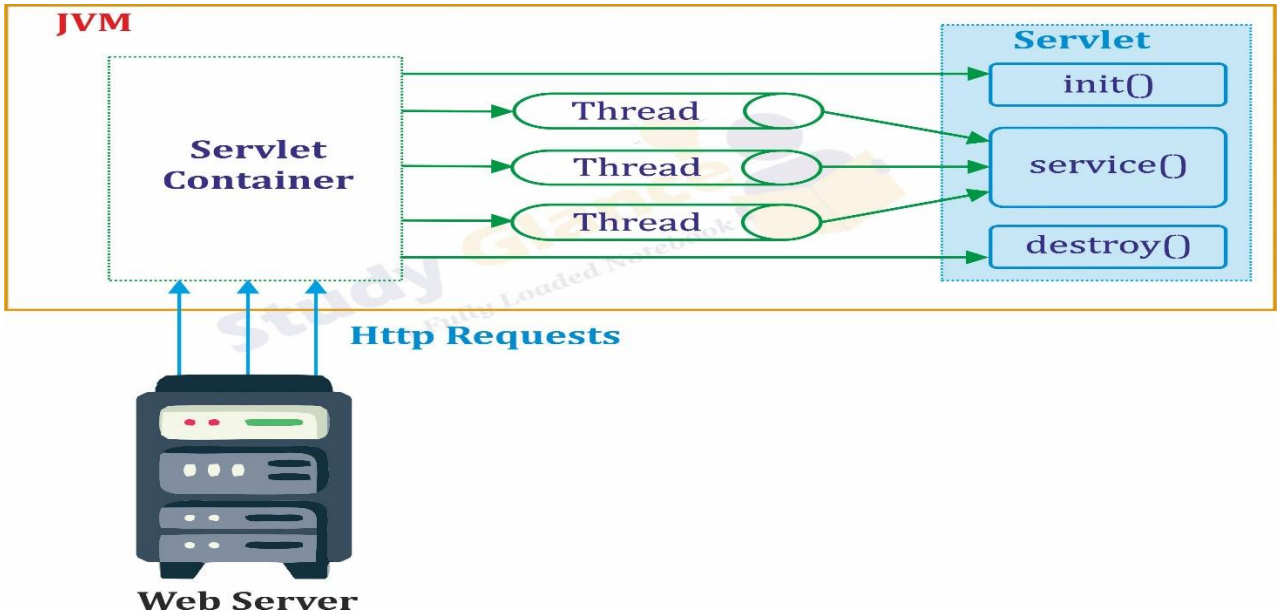


CGI	Servlet
The CGI programs are written in native OS.	Servlet is written in java class & its runs in JVM.
CGI creates a process base for each request.	Servlet creates a new thread to process each request.
The CGI is a language-independent interface that allows a server to start an external process.	The main purpose of servlets is to add up the functionality to a web server.
A CGI program needs to be loaded and started on each CGI request.	Servlets stay in the memory while serving the requests.
CGI is not able to read the HTTP servers.	Servlets are useful to read and set the HTTP servers.
CGI are platform dependent.	Servlets are platform independent.
Servlets are more secure and useful for Data sharing.	CGI is less secure and is not useful for data sharing.

Servlet Life Cycle

The Life Cycle of a Servlet

Three methods are central to the life cycle of a servlet. These are **init( ), service( ), and destroy( )**. They are implemented by every servlet and are invoked at specific times by the server. Let us consider a typical user scenario to understand when these methods are called.



First, assume that a user enters a Uniform Resource Locator (URL) to a Web browser. The browser then generates an HTTP request for this URL. This request is then sent to the appropriate server.

Second, this HTTP request is received by the Web server. The server maps this request to a particular servlet. The servlet is dynamically retrieved and loaded into the address space of the server.

Third, the server invokes the **init( )** method of the servlet. This method is invoked only when the servlet is first loaded into memory. It is possible to pass initialization parameters to the servlet so it may configure itself.

Fourth, the server invokes the **service( )** method of the servlet. This method is called to process the HTTP request. You will see that it is possible for the servlet to read data that has been provided in the HTTP request. It may also formulate an HTTP response for the client.

The servlet remains in the server's address space and is available to process any other HTTP requests received from clients. The service( ) method is called for each HTTP request.

Finally, the server may decide to unload the servlet from its memory. The algorithms by which this determination is made are specific to each server. The server calls the **destroy()** method to relinquish any resources such as file handles that are allocated for the servlet. Important data may be saved to a persistent store. The memory allocated for the servlet and its objects can then be garbage collected.

Servlet API

To create Java Servlets, we need to use Servlet API which contains all the necessary interfaces and classes. Servlet API has 2 packages namely

- **javax.servlet**
- **javax.servlet.http**

javax.servlet

This package support Generic servlet which is protocol independent. These interfaces and classes describe and define the contracts between a servlet class and the runtime environment provided by a servlet container.

Classes available in javax.servlet package

Classes	Description
`Generic Servlet`	Defines a generic, protocol-independent servlet.
`Servlet InputStream`	Provides an input stream for reading binary data from a client request, including an efficient readLine method for reading data one line at a time.
`Servlet OutputStream`	Provides an output stream for sending binary data to the client.
`Servlet ContextEvent`	This is the event class for notifications about changes to the servlet context of a web application.
`Servlet ContextAttributeEvent`	This is the event class for notifications about changes to the attributes of the servlet context of a web application.
`Servlet RequestEvent`	Events of this kind indicate lifecycle events for a Servlet Request.
`Servlet RequestAttributeEvent`	This is the event class for notifications of changes to the attributes of the servlet request in an application.

Interface available in javax.servlet package

Interface	Description
`Servlet`	Defines methods that all servlets must implement.
`ServletConfig`	A servlet configuration object used by a servlet container to pass information to a servlet during initialization.

Interface	Description
`ServletContext`	Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.
`ServletRequest`	Defines an object to provide client request information to a servlet.
`ServletResponse`	Defines an object to assist a servlet in sending a response to the client.
`RequestDispatcher`	Defines an object that receives requests from the client and sends them to any resource (such as a servlet, HTML file, or JSP file) on the server.

javax.servlet.http

This package describe and define the contracts between a servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming servlet container.

Classes	Description
`Cookie`	Creates a cookie, a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server.
`HttpServlet`	Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site.
`HttpServletRequestWrapper`	Provides a convenient implementation of the HttpServletRequest interface that can be subclassed by developers wishing to adapt the request to a Servlet.
`HttpServletResponseWrapper`	Provides a convenient implementation of the HttpServletResponse interface that can be subclassed by developers wishing to adapt the response from a Servlet.

`HttpSessionBindingEvent`	Events of this type are either sent to an object that implements HttpSessionBindingListener when it is bound or unbound from a session, or to a HttpSessionAttributeListener that has been configured in the deployment descriptor when any attribute is bound, unbound or replaced in a session.
`HttpSessionEvent`	This is the class representing event notifications for changes to sessions within a web application.

Classes available in javax.servlet package

Interface	**Description**
`HttpServletRequest`	Extends the ServletRequest interface to provide request information for HTTP servlets.
`HttpServletResponse`	Extends the ServletResponse interface to provide HTTP-specific functionality in sending a response.
`HttpSession`	Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.
`HttpSessionActivationListener`	Objects that are bound to a session may listen to container events notifying them that sessions will be passivated and that session will be activated.
`HttpSessionAttributeListener`	This listener interface can be implemented in order to get notifications of changes to the attribute lists of sessions within this web application.
`HttpSessionBindingListener`	Causes an object to be notified when it is bound to or unbound from a session.
`HttpSessionListener`	Implementations of this interface are notified of changes to the list of active sessions in a web application.

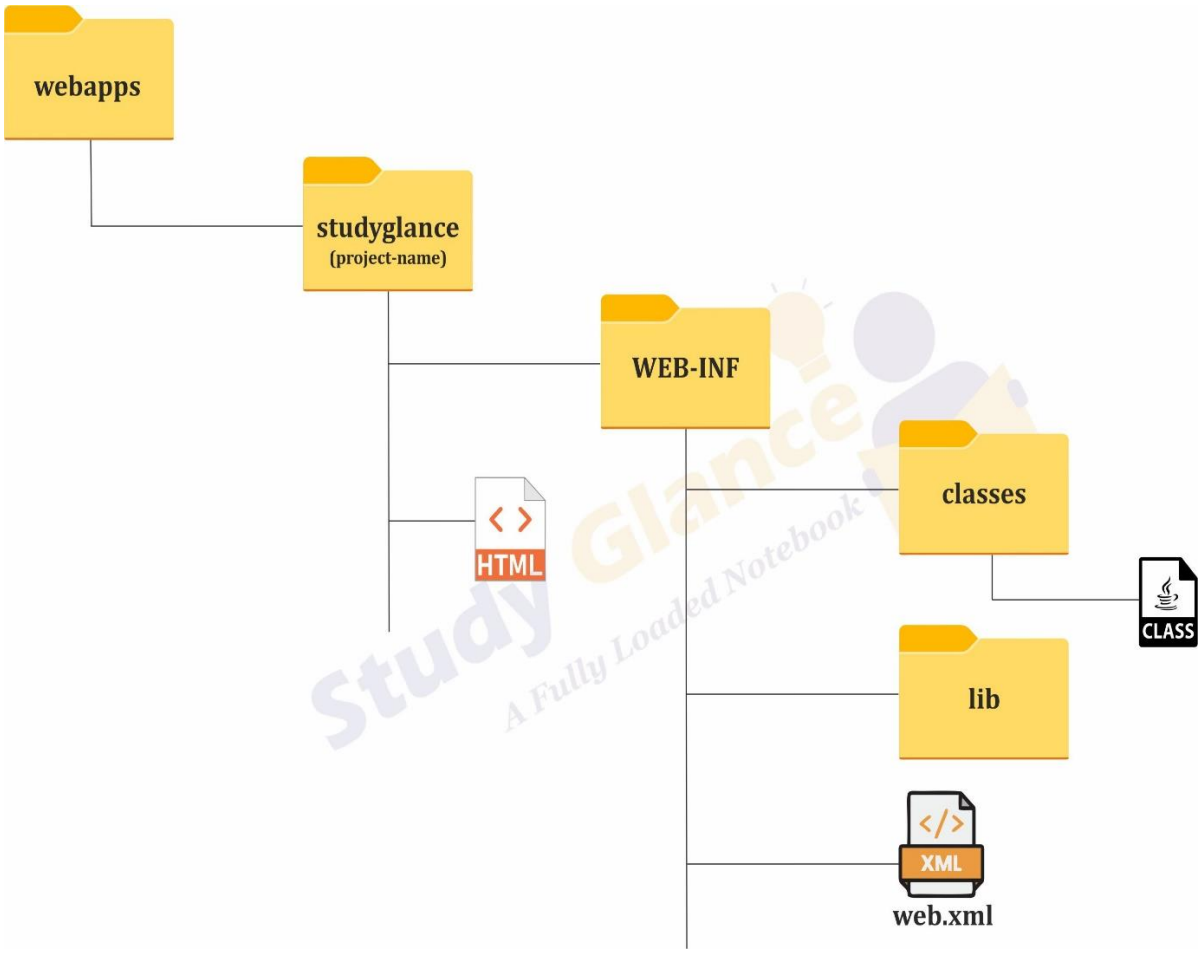
Creating and Running a Servlet Program

Steps for creating and running a servlet

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server
6. Access the servlet

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.



Create a Servlet

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

NOTE: The mostly used approach is by extending HttpServlet because it provides http request

HelloWorld.java

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloWorld extends HttpServlet {

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

      // Set response content type
      response.setContentType("text/html");

      // Actual logic goes here.
      PrintWriter out = response.getWriter();
      out.println("<h1>" + message + "</h1>");
   }
}
```

Note: HttpServlet contains all the life cycle methods of Servlet and the service() method is writen in the following two methods, they are

public void doGet (HttpServletRequest, HttpServletResponse) throws ServletException, IOException

public void doPost (HttpServletRequest, HttpServletResponse) throws ServletException, IOException

Compile the Servlet

To compile the servlet program we have to set a CLASSPATH

```
..\tomcat\lib\servlet-api.jar
```

Now compile the HelloWorld.java using following command

```
javac HelloWorld.java
```

If the compilation is successfull then HelloWorld.class file will be generated. Copy *.class file into document **root/WEB-INF/classes** folder

Create a deployment descriptor

1. Whenever client makes a request to a servlet that request is received by server and server goes to a predefined file called web.xml for the details about a servlet.
2. web.xml file always gives the details of the servlets which are available in the server.
3. If the requested servlet is available in web.xml then server will go to the servlet, executes the servlet and gives response back to client.
4. If the server is not able to find the requested servlet by the client then server generates an error (resource not found).

web.xml

```
<web-app>
	<servlet>
		<servlet-name>demo</servlet-name>
		<servlet-class>HelloWorld</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>demo</servlet-name>
		<url-pattern>/servletdemo</url-pattern>
	</servlet-mapping>
</web-app>
```

Start the server

To start Apache Tomcat server, double click on the startup.bat available at **../tomcat/bin** directory.

Access the servlet

Open broser and write **http://hostname:portno/contextroot/url-pattern**. For example:

```
http://localhost:8080/study/servletdemo
```

Reading Form Parameters in Servlet

Servlets parse the form(client) data automatically using the following methods depending on the situation −
getParameter() − You call request.getParameter() method to get the value of a form parameter.
getParameterValues() − Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
getParameterNames() − Call this method if you want a complete list of all parameters in the current request

Client pass some information from browser to web server uses GET Method or POST Method.

If a client send the data to the servlet, that data will be available in the object of HttpServletRequest interface. In case of getParameter() method we have to pass input parameter name and it will give the value.

```
request.getParameter("name")
```

If you are not aware of input parameter name? or if you have more input values its really tedious to use getParameter() method so we use getParameterNames(). This method returns an Enumeration that contains the parameter names in an unspecified order. Once we have an Enumeration, we can loop down the Enumeration in standard way by, using hasMoreElements() method to determine when to stop and using nextElement() method to get each parameter name.

```
Enumeration en=request.getParameterNames();
while(en.hasMoreElements())
{
	String param_name = (String) en.nextElement();
	String value=request.getParameter(param_name);
	.......
}
```

login.html

```
<html lang="en">
 <head>
  <title>Login page</title>
 </head>
 <body>
 <h1> Login Form </h1>
 <form method="post" action="login">
	Username: <input type="text" name="username">
	Password: <input type="password" name="pass">
 </form>
```

```
 </body>
</html>
```

LoginDemo.java

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class LoginDemo extends HttpServlet {

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

      // Set response content type
      response.setContentType("text/html");

      // Actual logic goes here.
      PrintWriter out = response.getWriter();
      String name=request.getParameter("username");   //will return
value
         out.println("Welcome "+name);
         out.close()
   }
}
```

web.xml

```
<web-app>
       <servlet>
              <servlet-name>login</servlet-name>
              <servlet-class>LoginDemo</servlet-class>
       </servlet>
       <servlet-mapping>
              <servlet-name>login</servlet-name>
              <url-pattern>/login</url-pattern>
       </servlet-mapping>
</web-app>
```

Reading Initialization Paraeters in Servlet

Both **ServletContext** and **ServletConfig** are basically the configuration objects which are used by the servlet container to initialize the various parameter of a web application. But they have some difference in terms of scope and availability.

Difference between ServletConfig vs. ServletContext

ServletConfig	ServletContext
ServletConfig object is one per servlet class.	ServletContext object is global to the entire web application.
Object of ServletConfig will be created during the initialization process of the servlet.	Object of ServletContext will be created at the time of web application deployment
We have to give the request explicitly in order to create the ServletConfig object for the first time.	ServletContext object can be available even before giving the first request.
Scope: As long as a servlet is executing, the ServletConfig object will be available, it will be destroyed once the servlet execution is completed.	Scope: As long as a web application is executing, the ServletContext object will be available, and it will be destroyed once the application is removed from the server.
ServletConfig object is used while only one servlet requires information shared by it.	ServletContext object is used while application requires information shared by it.
getServletConfig() method is used to obtain Servletconfig object.	getServletContext() method is used to obtain ServletContext object.
In web.xml — tag will be appear under tag.	In web.xml — tag will be appear under tag.

ServletConfig

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file. If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

Methods of ServletConfig interface

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

public Enumeration getInitParameterNames(): Returns an enumeration of all the initialization parameter names.
public String getServletName(): Returns the name of the servlet

web.xml

```
<web-app>
       <servlet>
              <servlet-name>Config</servlet-name>
              <servlet-class>ConfigDemo</servlet-class>
              <init-param>
                     <param-name>username</param-name>
                     <param-value>studyglance</param-value>
              </init-param>
       </servlet>
       <servlet-mapping>
              <servlet-name>Config</servlet-name>
              <url-pattern>/welcome</url-pattern>
       </servlet-mapping>
</web-app>
```

ConfigDemo.java

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class ConfigDemo extends HttpServlet {

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

      // Set response content type
      response.setContentType("text/html");

      // Actual logic goes here.
      PrintWriter out = response.getWriter();

         //get ServletConfig object.
         ServletConfig config=getInitServletConfig();

         //get init parameter from ServletConfig object.
      String name= config.getParameter("username");
         out.println("Welcome "+name);
         out.close()
```

```
    }
}
```

ServletContext

ServletContext 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 element.

Methods commonly used in ServletContext interface

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.

web.xml

```
<web-app>
       <context-param>
              <param-name>driver</param-name>
              <param-value>com.mysql.jdbc.Driver</param-value>
       </context-param>
       <servlet>
              <servlet-name>Context</servlet-name>
              <servlet-class>ContextDemo</servlet-class>
       </servlet>
       <servlet-mapping>
              <servlet-name>Context</servlet-name>
              <url-pattern>/welcome</url-pattern>
       </servlet-mapping>
</web-app>
```

ConfigDemo.java

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class ContextDemo extends HttpServlet {
```

```
   public void doPost(HttpServletRequest request, HttpServletRespons
e response)
      throws ServletException, IOException {

      // Set response content type
      response.setContentType("text/html");

      // Actual logic goes here.
      PrintWriter out = response.getWriter();

         //get ServletContext object.
         ServletContext context=getServletContext();

         //get init parameter from ServletConfig object.
      String driverName= context.getInitParameter("driver");
         out.println("driver name is = "+driverName);
         out.close()
   }
}
```

Session Tracking in Servlet

Session tracking is a mechanism that servlets use to maintain state about a series of requests from the same user (that is, requests originating from the same browser) across some period of time. 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.

Session Tracking Techniques

There are four techniques used in Session tracking:

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

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.

```
<input type="hidden" name="uname" value="Studyglance">
```

Advantage of Hidden Form Field

It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

1. It is maintained at server side.
2. Extra form submission is required on each pages.
3. Only textual information can be used.

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.

Advantage of URL Rewriting

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

Cookies in Servlet

A cookie is a piece of data from a website that is stored within a web browser(Client side) that the website can retrieve at a later time. Cookies are used to tell the server that users have returned to a particular website.

How cookies work

When you send a request to the server, the server sends a reply in which it embeds the cookie which serves as an identifier to identify the user. So, next time when you visit the same website, the cookie lets the server know that you are visiting the website again.

Cookies Used For?

Session management. For example, cookies let websites recognize users and recall their individual login information and preferences, such as sports news versus politics.

Personalization. Customized advertising is the main way cookies are used to personalize your sessions. You may view certain items or parts of a site, and cookies use this data to help build targeted ads that you might enjoy.

Tracking. Shopping sites use cookies to track items users previously viewed, allowing the sites to suggest other goods they might like and keep items in shopping carts while they continue shopping.

Commonly used Methods in Cookie

public String getName( ) Returns the name of the cookies.
public String getPath( ) Returns the path of the server to which the browser returns the cookie.
public String getValue( ) Returns the value of the cookie.
public int getMaxAge( ) Returns the maximum age limit to the cookie, specified in seconds.
public void setMaxAge(int expiry) Sets the maximum age of the cookies in seconds.
public void setValue(String newValue) Allocates a new value to a cookie after the cookie is created.

Create a Cookie Object

The constructor of Cookie class creates the cookie object with corresponding cookie name and value.

```
//creating cookie object
Cookie cookie = new Cookie("username","Studyglance");

//adding cookie to the response
Response.addCookie(cookie);
```

Get Cookie from client request

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

cookiedemo.html

```
<html>
	<head>
		<title>Cookie Demo</title>
	</head>
	<body>
		<form action="servlet1" method="post">
			Enter Your Name:<input type="text" name="userNa
me"/><br/><br/>
			<input type="submit" value="SUBMIT"/>
		</form>
	</body>
</html>
```

SetCookieServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SetCookieServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletRespon
se response){
        try{
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            String name=request.getParameter("userName");
            //creating cookie object
            Cookie ck=new Cookie("uname",name);
            //adding cookie in the response
            response.addCookie(ck);
            out.print("Cookie Saved");
            //creating submit button
            out.print("<br><form action='servlet2' method='post'>");
            out.print("<input type='submit' value='View the cookie v
alue'>");
            out.print("</form>");
            out.close();
        }
        catch(Exception e){
            System.out.println(e);
        }
    }
```

```
}
```

GetCookieServlet.java

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

public class GetCookieServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletRespon
se response){
        try{
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            Cookie ck[]=request.getCookies();
            out.println("<h3>Reading the cookies</h3>");
            out.println("<br/>");
            out.println("Name of the cookie : " + ck[0].getName() +
"<br/>");
            out.println("Value in cookie : " + ck[0].getValue());
            out.close();
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

web.xml

```
<web-app>
       <servlet>
              <servlet-name>set</servlet-name>
              <servlet-class>SetCookieServlet</servlet-class>
       </servlet>
       <servlet-mapping>
              <servlet-name>set</servlet-name>
              <url-pattern>/servlet1</url-pattern>
       </servlet-mapping>
       <servlet>
              <servlet-name>get</servlet-name>
              <servlet-class>GetCookieServlet</servlet-class>
       </servlet>
       <servlet-mapping>
              <servlet-name>get</servlet-name>
              <url-pattern>/servlet2</url-pattern>
       </servlet-mapping>
</web-app>
```

HTTPSession in Servlet

A session contains information specific to a particular user across the whole application. When a user enters into a website (or an online application) for the first time HttpSession is obtained via request.getSession(), the user is given a unique ID to identify his session.

The HttpSession stays alive until it has not been used for more than the timeout value specified in tag in deployment descriptor file(web.xml). The default timeout value is 30 minutes, this is used if you don't specify the value in tag. This means that when the user doesn't visit web application time specified, the session is destroyed by servlet container. The subsequent request will not be served from this session anymore, the servlet container will create a new session.

Commonly used Methods of Servlet HttpSession

setAttribute(String name, Object value): Binds an object to this session, using the name specified.
setMaxInactiveInterval(int interval): Specifies the time, in seconds, between client requests before the servlet container will invalidate this session.
getAttribute(String name): Returns the object bound with the specified name in this session, or null if no object is bound under the name.
getAttributeNames(): Returns an Enumeration of String objects containing the names of all the objects bound to this session.
getId(): Returns a string containing the unique identifier assigned to this session.
invalidate(): Invalidates this session then unbinds any objects bound to it.
removeAttribute(String name): Removes the object bound with the specified name from this session.

sessionemo.html

```
<html>
	<head>
		<title>session Demo</title>
	</head>
	<body>
		<form action="servlet1" method="post">
			Enter Your Name:<input type="text" name="userNa
me"/><br/><br/>
			<input type="submit" value="SUBMIT"/>
		</form>
	</body>
</html>
```

SetSessionServlet.java

```
import java.io.*;
```

```
import javax.servlet.*;
import javax.servlet.http.*;
public class SetSessionServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletRespon
se response){
        try{
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            String name=request.getParameter("userName");
            // Create a session object.
                      HttpSession session = request.getSession(true)
;

            session.setAttribute("uname",name);
            //creating submit button
            out.print("<br><form action='servlet2' method='post'>");
            out.print("<input type='submit' value='View the session'
>");
            out.print("</form>");
            out.close();
        }
        catch(Exception e){
            System.out.println(e);
        }
    }
}
```

GetSessionServlet.java

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

public class GetSessionServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletRespon
se response){
        try{
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            HttpSession session=request.getSession(false);
                     String name=(String)session.getAttribute("uname
");
            out.print("Hello!.. Welcome "+name);
            out.close();
        }catch(Exception e){
            System.out.println(e);
        }
    }
```

```
}
```

web.xml

```
<web-app>
	<servlet>
		<servlet-name>set</servlet-name>
		<servlet-class>SetSessionServlet</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>set</servlet-name>
		<url-pattern>/servlet1</url-pattern>
	</servlet-mapping>
	<servlet>
		<servlet-name>get</servlet-name>
		<servlet-class>GetSessionServlet</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>get</servlet-name>
		<url-pattern>/servlet2</url-pattern>
	</servlet-mapping>
</web-app>
```

Java Server Page(JSP)

Introduction to JSP

- **JSP** is a server side technology. It is used for creating dynamic web applications, using java as programming language.

- This is mainly used for implementing presentation layer (GUI Part) of an application. A complete JSP code is more like a HTML with bits of java code in it.

- A JSP page consists of **HTML tags**(generates Static Content) and **JSP tags** (generates Dynamic Content).

- The JSP pages are easier to maintain than Servlet because we can separate designing(**Presentation Logic**) and development(**Business Logic**).

- JSP pages are opposite of Servlets as a servlet adds HTML code inside Java code, while JSP adds Java code inside HTML using JSP tags.

- **Easy to maintain**
  - JSP can be easily managed because we can easily separate our business logic with presentation logic.
  - In Servlet technology, we mix our business logic with the presentation logic.

- **Fast Development: No need to recompile and redeploy**
  - If JSP page is modified, we don't need to recompile and redeploy the project.
  - The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

- **Less code than Servlet**
  - In JSP, we can use many tags such as action tags, implicit objects, custom tags, etc. that reduces the code.

To Work with JSP: (Environment Setup)

1. We need to install Java Software Development Kit (SDK)

2. After successful installation we need to set the **PATH** and **JAVA_HOME** environment variables.

- ✓ **PATH**="C:\Program Files\Java\jdk1.8.0_191\bin;.;"
- ✓ **JAVA_HOME=**"C:\Program Files\Java\jdk1.8.0_191".

3. After that, we need to install Tomcat server(ignore this if XAMPP already installed)

The Anatomy of JSP

- The JSP is almost like a regular web page with the inclusion of dynamic behavior through JSP Elements.

- A JSP page consists of **HTML tags**(generates Static Content) and **JSP tags** (generates Dynamic Content).

- Hence We can say that a JSP page has two components

JSP Elements(tags):

- These are the elements that are understood and translated by JSP Container. These elements are used to generate dynamic content.

Template Data(HTML tags):

- These are the elements that are not understood and translated by JSP Container. The template data is used to generate static content.

A Simple JSP Program:

1. Open text editor (notepad,edit+,notepad++,..)
2. Write jsp program and save it with **.jsp** extension in the following location

   In Windows

   C:/xampp/tomcat/webapps/foldername/filename.jsp

   In Linux(ubuntu)

   var/lib/tomcat/webapps/foldername/filename.jsp

3. To execute, Open any browser and give following as **URL**

   Port No of Tomcat Server

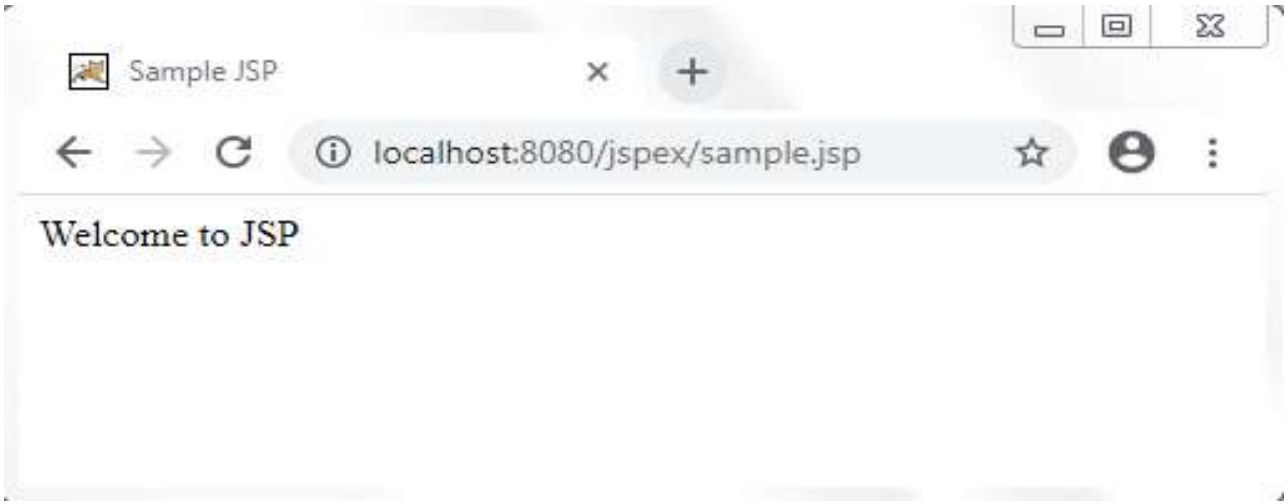
   localhost:8080/foldername/filename.jsp

Example:

"**sample.jsp**"

```
<html>
<head>
<title>Sample JSP</title>
</head>
<body>
<% out.println("Welcome to JSP"); %>
</body>
</html>
```

Output:

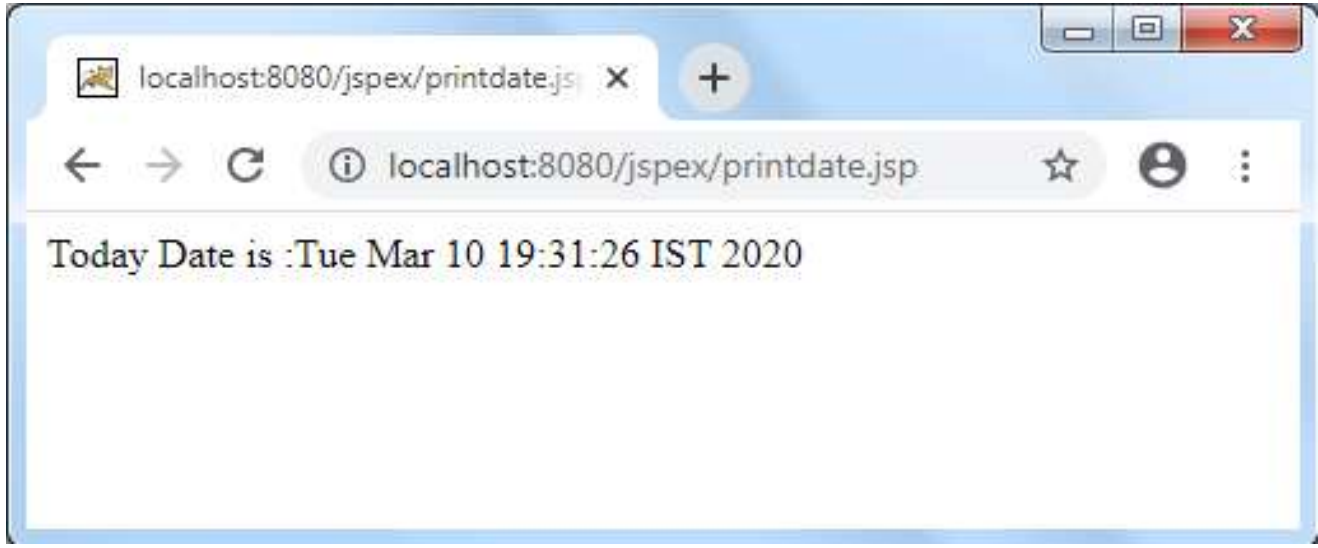


Example: To display Current System Date and Time

"printdate.jsp"

```
<%@page import="java.util.Date"%>
<html>
<body>
<%
Date d=new Date();
out.println("Today Date is :"+d);
 %>
</body>
</html>
```

Output:



JSP Processing

- Any Web server needs a container to run any web component (or) to provide interface to run web components.

- The JSP also needs a JSP container to execute or process a JSP page.

- In general, a JSP container acts as an interface between JSP page and Server. i.e. it receives all JSP requests and generates the responses.

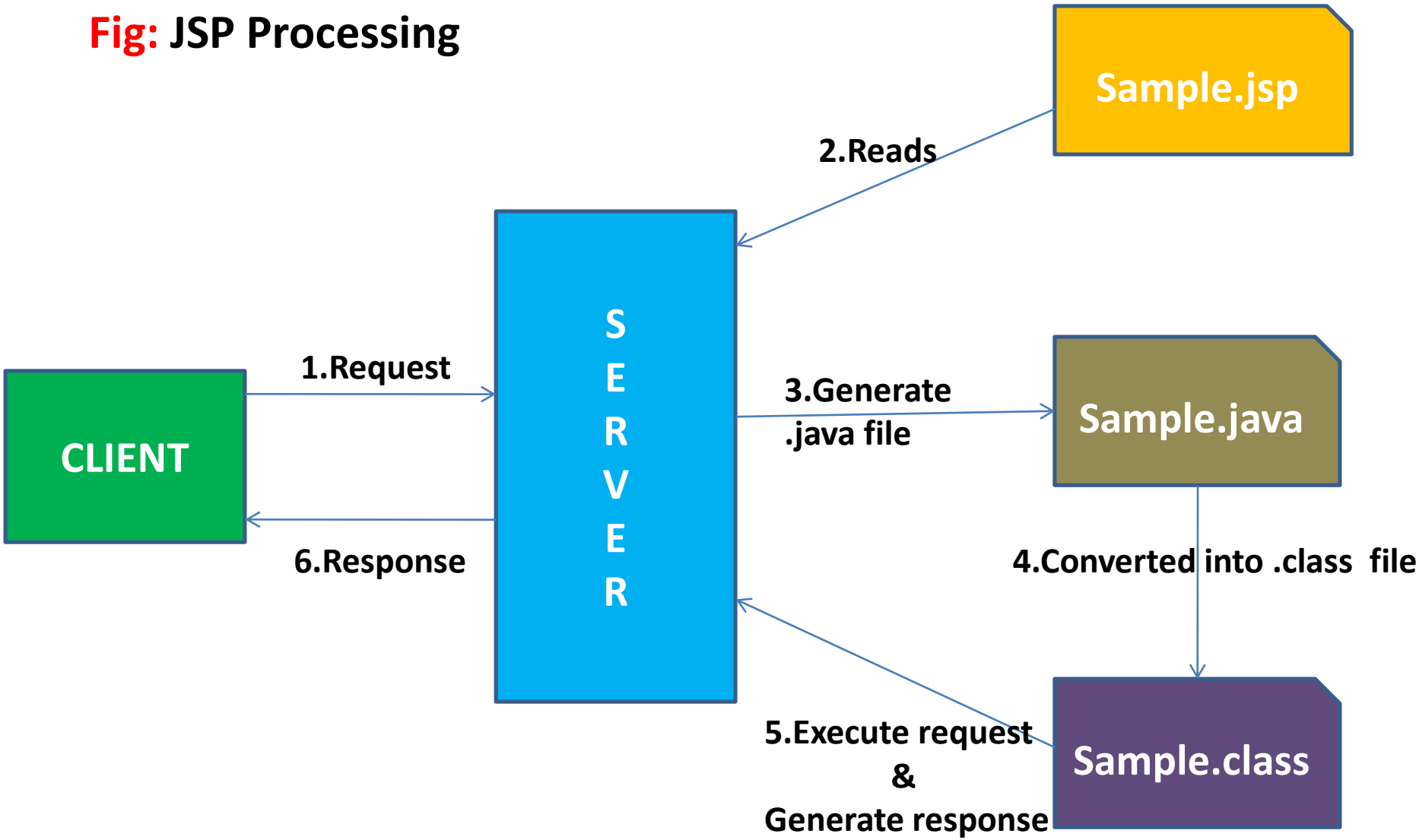
JSP Processing

- In JSP processing there are two phases ,those are
  - Translation Phase
  - Request Processing Phase

- **In translation phase**, The JSP page converted into servlet code or into (.java file).

- **In request processing phase**, Servlet code can be converted into .class file to process the request.

Fig: JSP Processing



Implicit Objects in JSP

- These objects are created by the web container that are available to all the jsp pages.
- These objects are also called as Pre-Defined Variables in JSP.
- JSP implicit objects are created during the translation phase of JSP to the servlet.
- There are **9 types** of implicit objects available in the JSP :

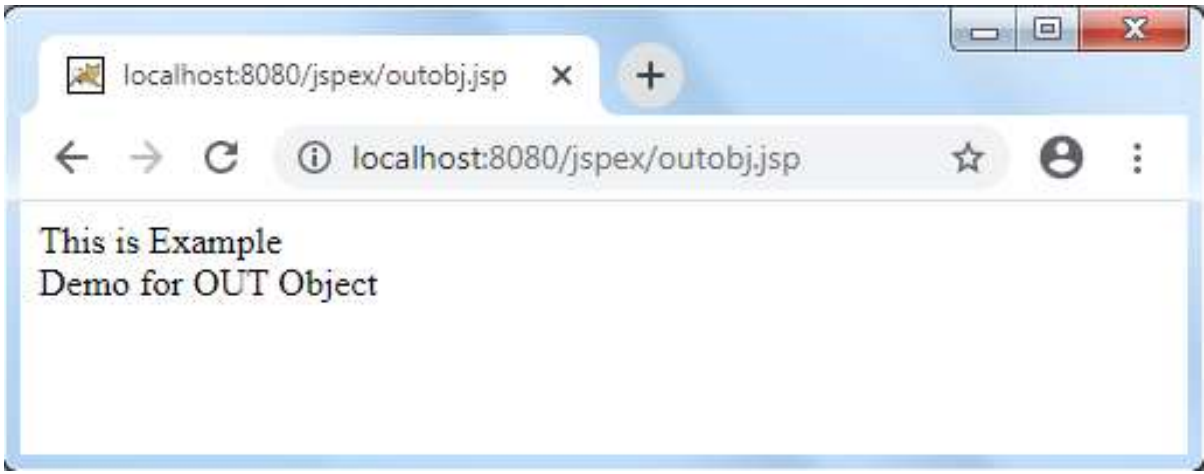
1.out
2.request
3.response
4.session
5.exception
6.application
7.pageContext
8.page
9.config

Out Object:

- **Out** is one of the implicit objects to write the data to the buffer and send output to the client(browser) in response.
- It is the object of **JspWriter** Class.

Example: **"outobj.jsp"**

```
<html>
<body>
<% out.print("This is Example <br/> ");
 out.print(" Demo for OUT Object");
 %>
</body>
</html>
```



Request Object:

- The **request** is an implicit object of type HttpServletRequest. It was created for each jsp request by the web container.
- It can be used to get request information from web page or document.
- By using this **request** object ,we can able read values from input elements in JSP.

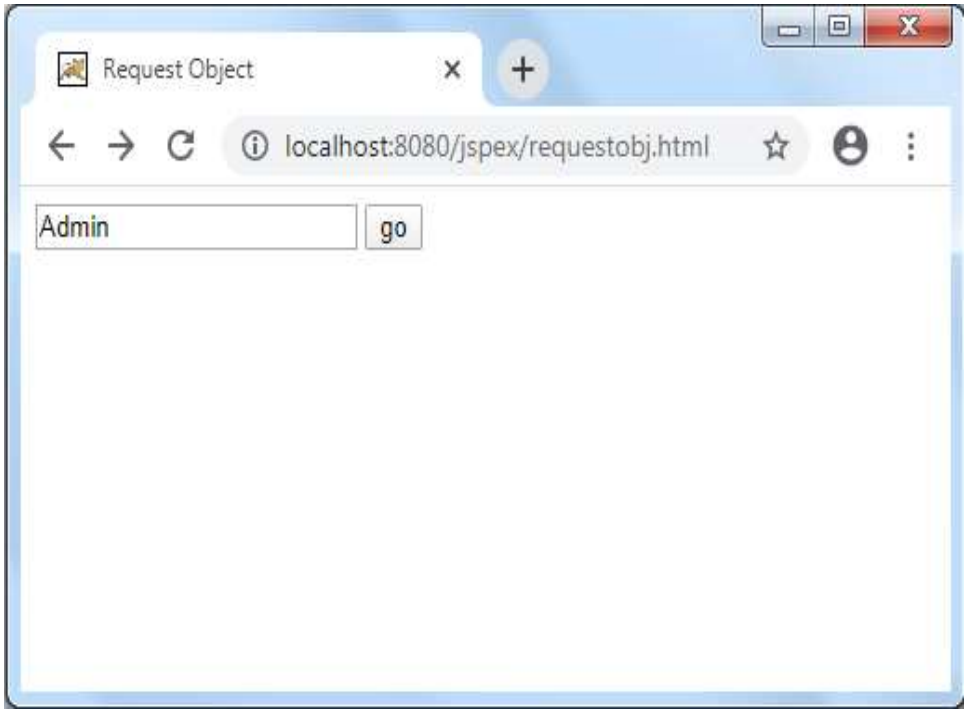
Example: **"requestobj.html"**

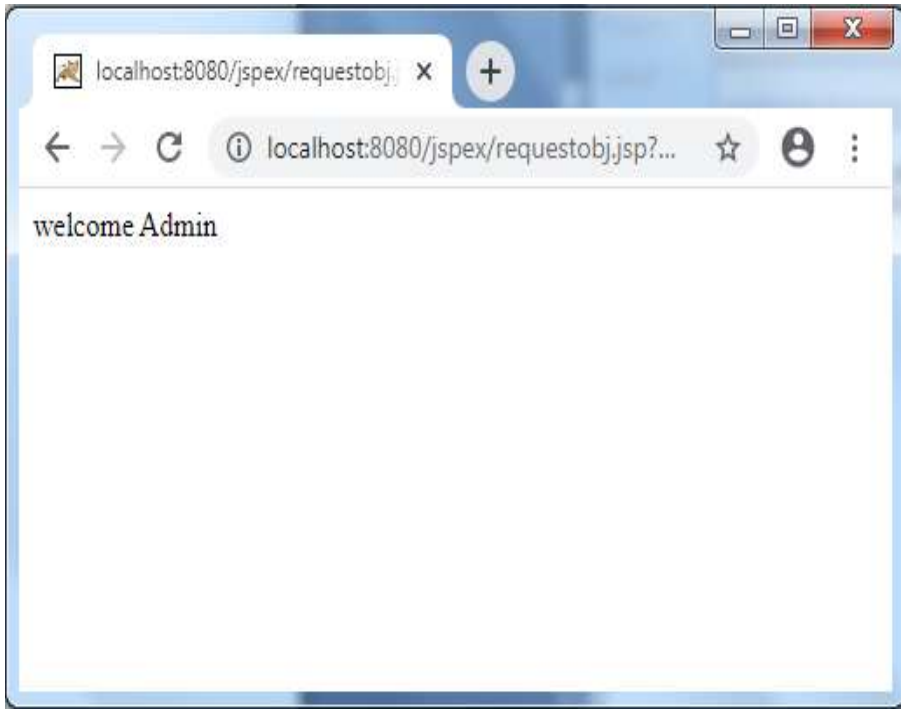
```
<html>
 <head>
 <title>Request Object</title>
 </head>
 <body>
<form action="requestobj.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
 </body>
</html>
```

requestobj.jsp

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

Output:





Response Object:

- In JSP, **response** is an implicit object of type HttpServletResponse. It is created by the web container for each jsp request.
- It can be used to add or manipulate response such as redirect response to another resource, send error etc.

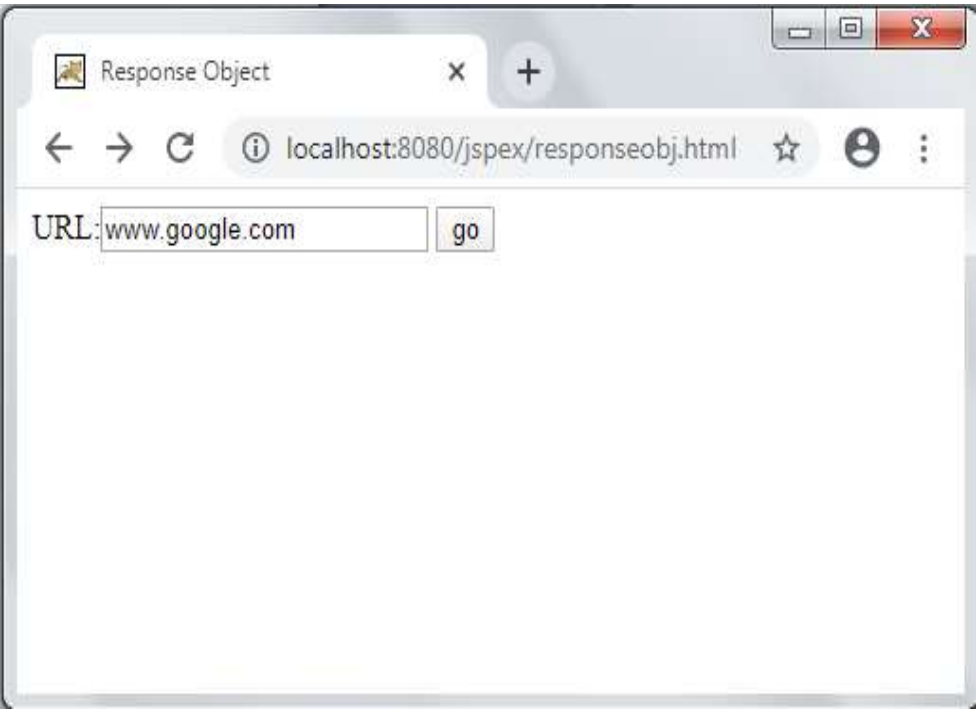
Example:

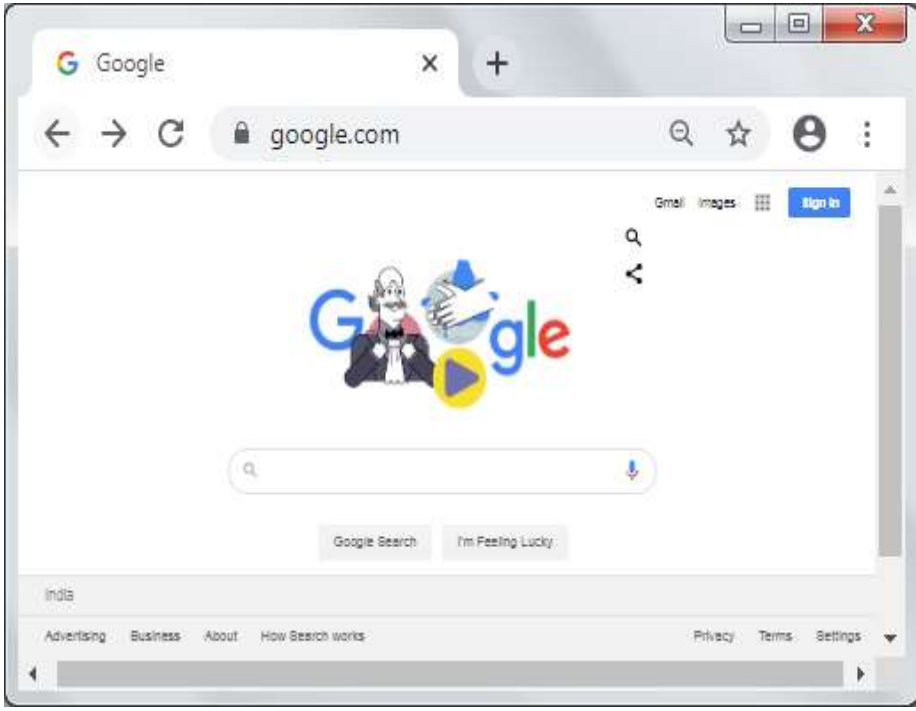
```
<html>
<head>
<title>Response Object</title>
</head>
<body>
<form action="responseobj.jsp">
URL:<input type="text" name="url">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

responseobj.jsp

```
<%
String url=request.getParameter("url");
response.sendRedirect("http://"+url);
%>
```

Output:





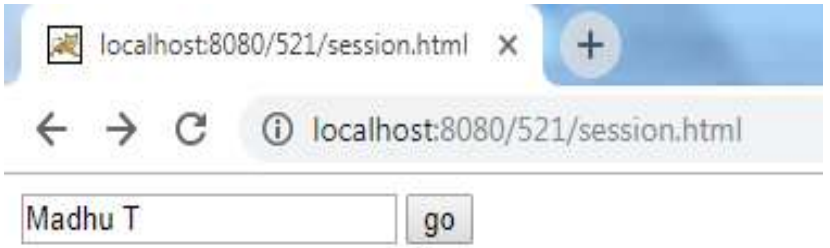
Session Object:

- In JSP, session is an implicit object of type HttpSession. We can use this object to set, get or remove attribute or to get session information.

Example:

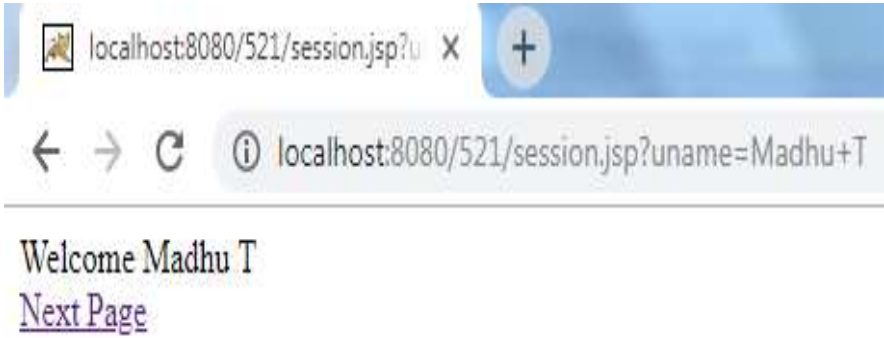
Session.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go">
</form>
</body>
</html>
```



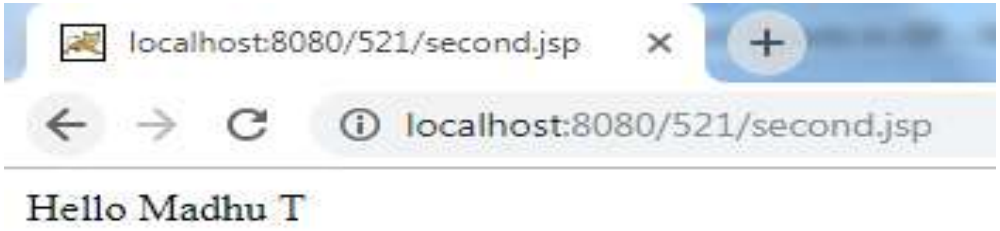
session.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("Welcome "+name);
session.setAttribute("user",name);
%>
<br/>
<a href="second.jsp">Next Page</a>
</body>
</html>
```



second.jsp

```
<html>
<body>
<%
 String name=(String)session.getAttribute("user");
out.print("Hello "+name);
 %>
</body>
</html>
```



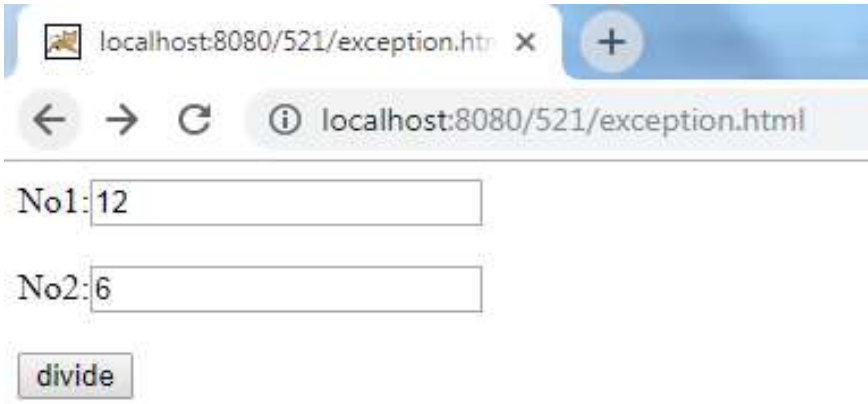
Exception Object:

- The exception is normally an object that is thrown at runtime. Exception Handling is the process to handle the runtime errors. There may occur exception any time in your web application.
- In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.

Example:

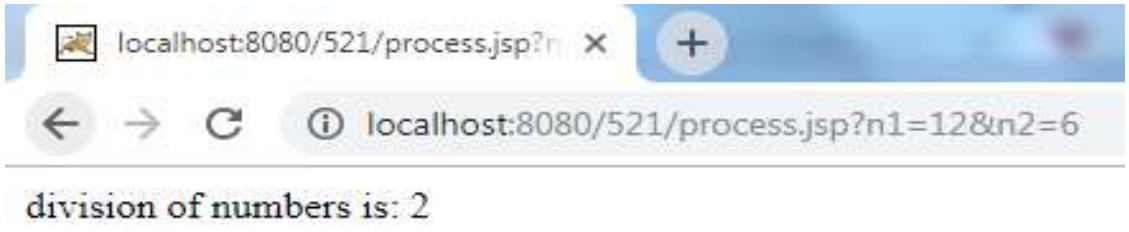
exception.html

```
<html>
<body>
<form action="process.jsp">
No1:<input type="text" name="n1" />
No2:<input type="text" name="n2" />
<input type="submit" value="divide"/>
</form>
 </body>
</html>
```



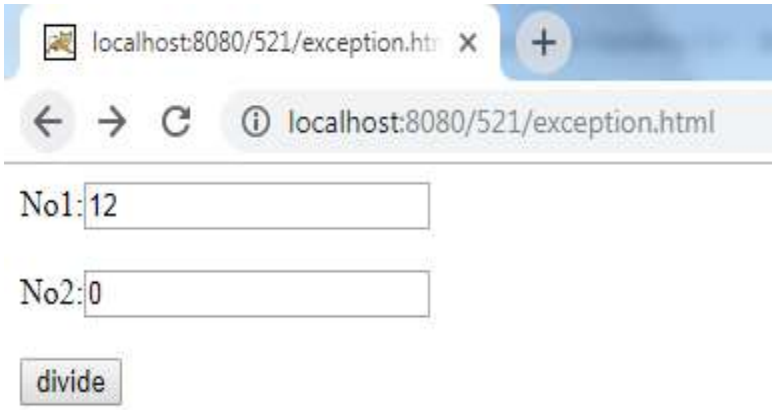
process.jsp

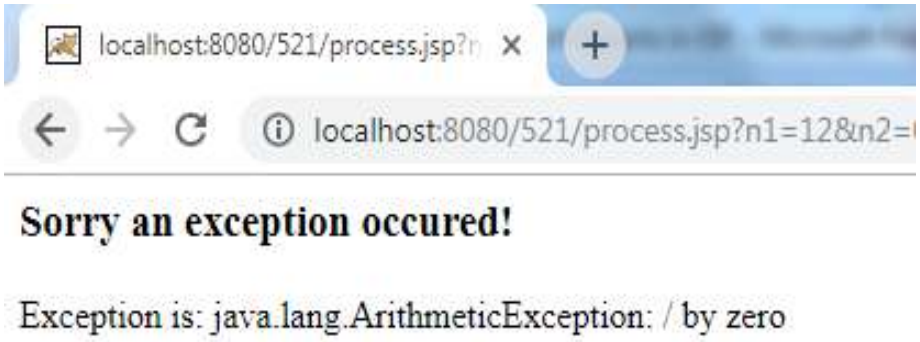
```
<%@ page errorPage="error.jsp" %>
<%
String num1=request.getParameter("n1");
String num2=request.getParameter("n2");
int a=Integer.parseInt(num1);
int b=Integer.parseInt(num2);
int c=a/b;
out.print("division of numbers is: "+c);
%>
```



error.jsp

```
<html>
<body>
<%@ page isErrorPage="true" %>
<h3>Sorry an exception occured!</h3>
Exception is: <%= exception %>
</body>
</html>
```





Application Object:

- This is basically used for getting initialization parameters and for sharing the attributes & their values across the entire JSP application.
- which means any attribute set by **application implicit object** would be available to all the JSP pages.

- **getAttribute(String attributeName):** It returns the object stored in a given attribute name.

  Ex: String s = (String)application.getAttribute("MyAttr");

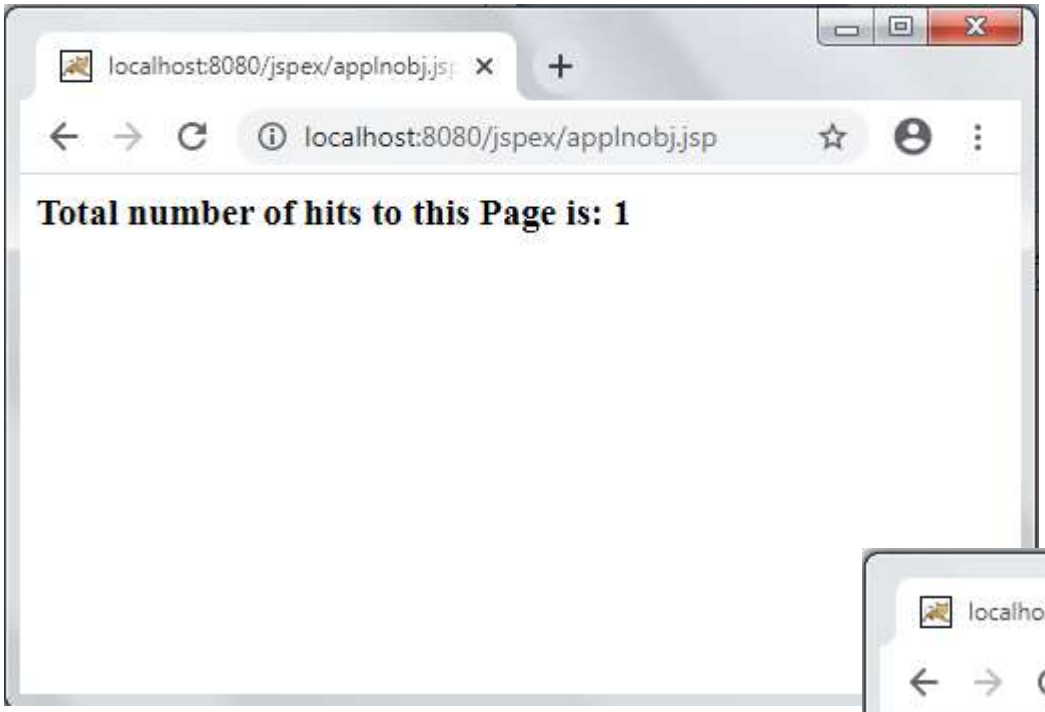
- **setAttribute(String attributeName, Value):** It sets the value of an attribute or in other words it stores an attribute and its value in application context, which is available to use across JSP application.

  Ex:application.setAttribute(“MyAttr”, “value”);

applnobj.jsp

```
<html>
<%
Integer counter= (Integer)application.getAttribute("numOfVisits");
 if( counter ==null || counter == 0 ){
  counter = 1;
 }else{
  counter = counter+ 1;
 }
 application.setAttribute("numOfVisits", counter);
%>
<h3>Total number of hits to this Page is: <%= counter%></h3>
</html>
```

Output:


localhost:8080/jspex/applnobj.jsp
localhost:8080/jspex/applnobj.jsp
Total number of hits to this Page is: 1


localhost:8080/jspex/applnobj.jsp
localhost:8080/jspex/applnobj.jsp
Total number of hits to this Page is: 2

pageContext Object:

- In JSP, **pageContext** is an implicit object of type PageContext class.
- The **pageContext** object can be used to set, get or remove attribute from one of the following scopes:

✓ **page** – Scope: PAGE_CONTEXT

✓ **request** – Scope: REQUEST_CONTEXT

✓ **session** – Scope: SESSION_CONTEXT

✓ **application** – Scope: APPLICATION_CONTEXT

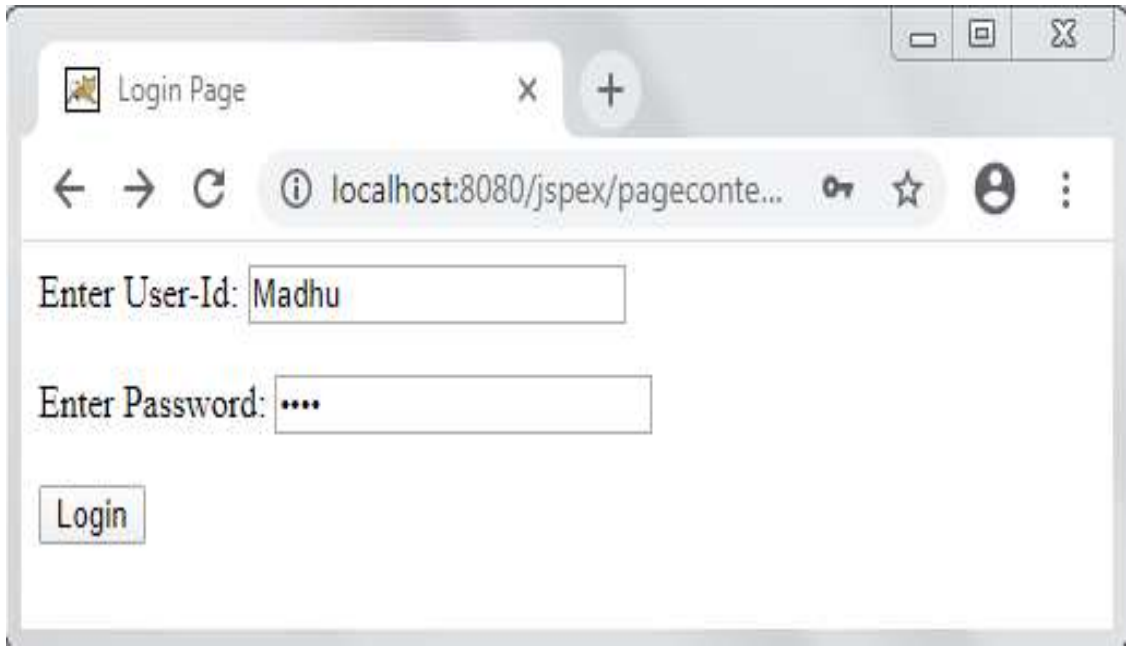
- In JSP, page scope is the default scope.

Methods of pageContext Implicit Object

- getAttribute (String AttributeName, int Scope)
- setAttribute(String AttributeName, AttributeValue, int Scope)

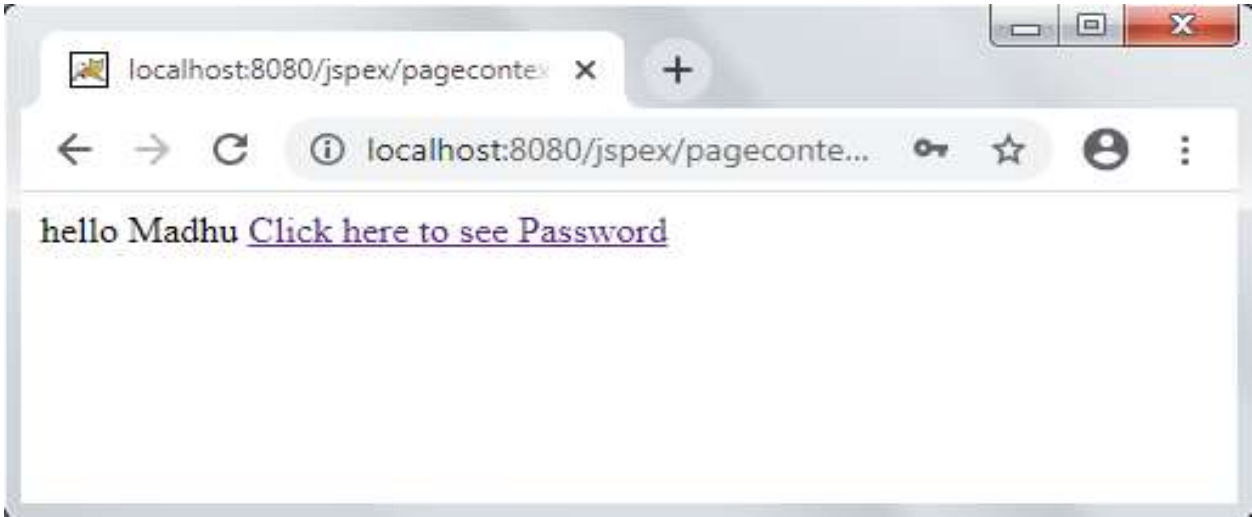
pagecontextobj.html

```
<html>
<head><title> Login Page</title></head>
<body>
<form action="pagecontextobj.jsp">
Enter User-Id: <input type="text" name="uid"><br><br>
Enter Password: <input type="password" name="upass"><br><br>
<input type="submit" value="Login">
</form>
</body>
</html>
```



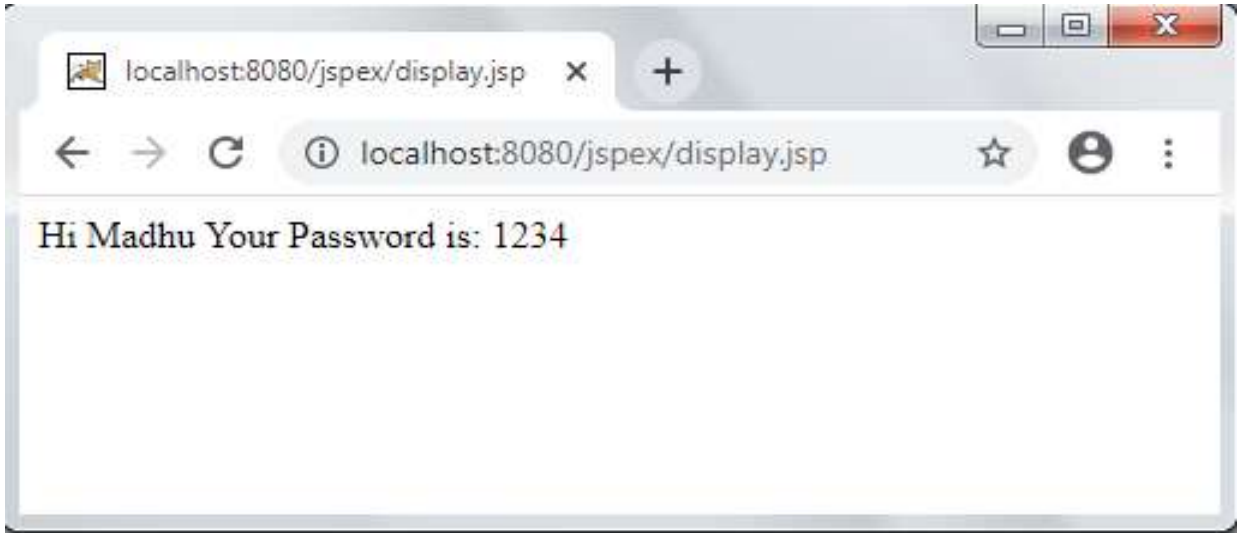
pagecontextobj.jsp

```
<html>
<body>
<%  String id=request.getParameter("uid");
String pass=request.getParameter("upass");
out.println("hello "+id);
pageContext.setAttribute("uname", id, PageContext.SESSION_SCOPE);
pageContext.setAttribute("pwd", pass, PageContext.SESSION_SCOPE);
%>
<a href="display.jsp">Click here to see Password </a>
</body>
</html>
```



display.jsp

```
<html>
<body>
<%
String uname= (String) pageContext.getAttribute("uname",
                                                 PageContext.SESSION_SCOPE);
String pwd= (String) pageContext.getAttribute("pwd",
                                                 PageContext.SESSION_SCOPE);
out.println("Hi "+uname);
out.println("Your Password is: "+pwd);
%>
</body>
</html>
```

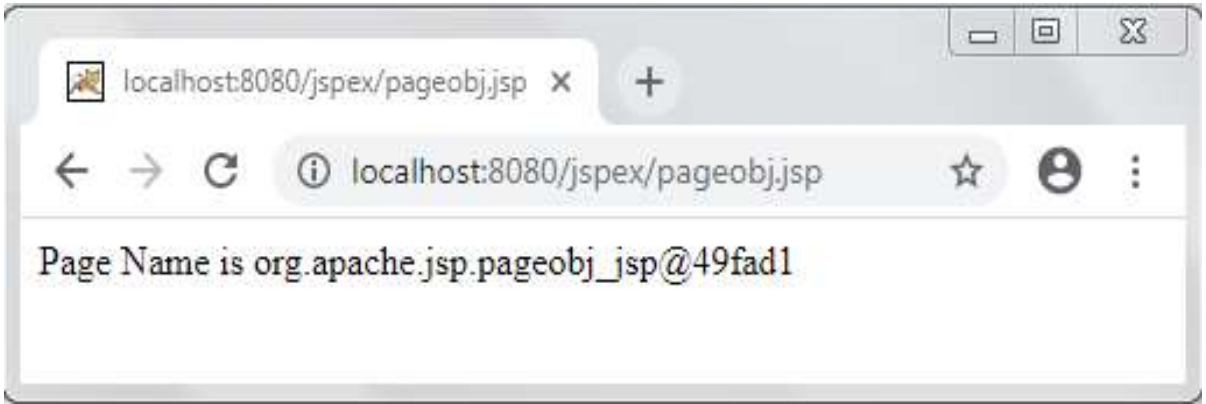


page Object:

- Page implicit variable holds the currently executed servlet object for the corresponding jsp.
- Acts as this object for current jsp page.
- This object is very rarely used.

Example:

```
<html>
<body>
<% String pageName = page.toString();
out.println("Page Name is " +pageName);%>
</body>
</html>
```



config Object:

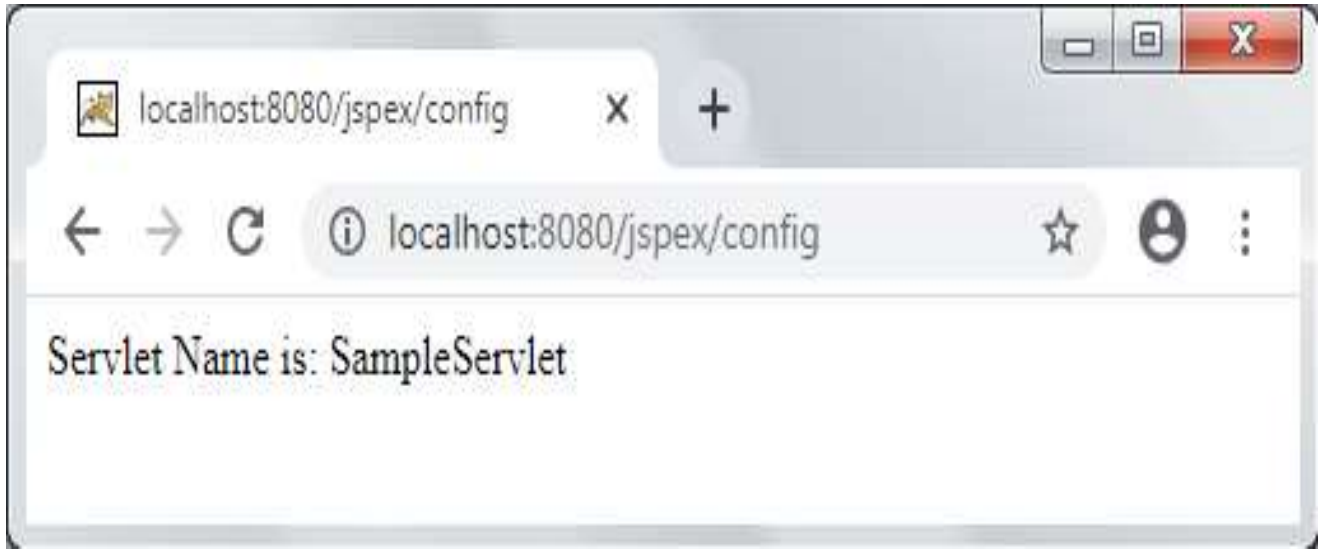
- In JSP, config is an implicit object of type ***ServletConfig***. This object can be used to get initialization parameter for a particular JSP page.
- The config object is created by the web container for each jsp page.
- Generally, it is used to get initialization parameter from the web.xml file.

Example: **“configobj.jsp”**

```
<html>
<body>
<%
String sname=config.getServletName();
out.print("Servlet Name is: "+sname);
%>
</body>
</html>
```

web.xml

```
<web-app>
<servlet>
<servlet-name>SampleServlet</servlet-name>
<jsp-file>/configobj.jsp</jsp-file>
</servlet>
<servlet-mapping>
<servlet-name>SampleServlet</servlet-name>
<url-pattern>/config</url-pattern>
</servlet-mapping>
</web-app>
```



Java Server Page(JSP)
Elements

JSP Elements

- JSP Elements are used to add dynamic content to JSP page.

- In other words, JSP Elements allows us to write java code in HTML document.

- All JSP Elements may be written in one of two forms
  - The **XML** Form
  - The **<%** …. **%>** form

- There are several types of JSP Elements, those are
  1) Directive Elements
  2) Scripting Elements
  3) Action Elements

Directive Elements

- Directive elements in JSP provides the special information to the JSP engine. It gives the information about the JSP page.

- Each JSP page goes through the two phases i.e. translation phase and request processing phase.

- At the translation phase,the JSP engine translates the JSP page into a Servlet, directive elements are handled at the translation time.

- Directive Elements gives special instruction to Web Container at the time of page translation.

  Syntax:

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

Directive Elements

- JSP support 3 types of Directive Elements: **page**, **include** and **taglib**.

page directive:

- The page directive defines attributes (or) properties that apply to an entire JSP page. such as the size of the allocated buffer, imported packages, defining what type of page it is etc.
- Syntax of JSP page directive

  <%@ **page** attribute="value" %>
- Page directive supports so many attributes ,some of them are,
- import
- contentType
- buffer
- session
- isErrorPage

Directive Elements

include directive:

- JSP include directive is used to include other files into the current jsp page. These files can be html files, other sp files etc.

- The advantage of using an include directive is that it allows code re-usability.
- The syntax of an include directive :

```
<%@ include file = "file location"%>
```

taglib directive:

- JSP taglib directive is used to define the tag library and custom tags, which we can use in JSP.

- **Syntax of taglib directive:**

```
<%@ taglib uri="uri" prefix="value"%>
```

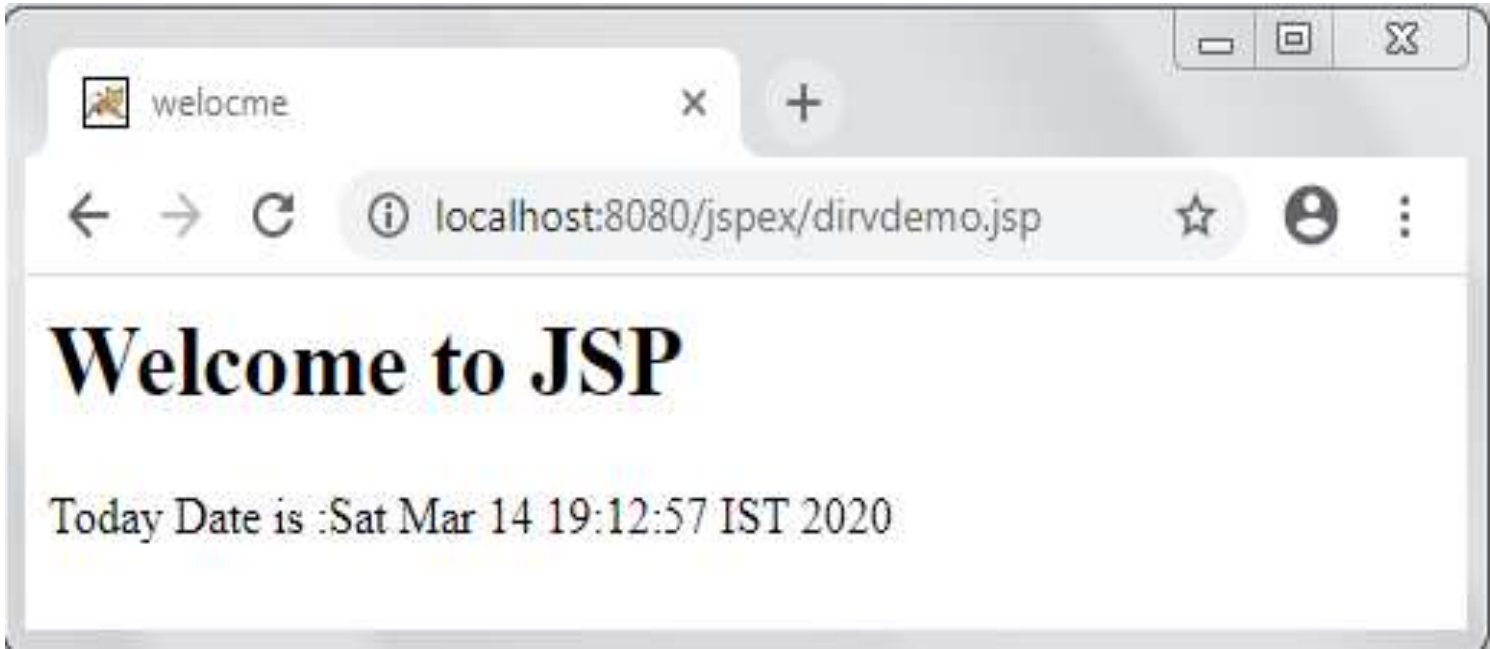
Example: “dirvdemo.jsp”

```
<html>
<body>
<!-- JSP include directive -->
<%@include file="welcome.html" %>
<!-- JSP page directive -->
 <%@page import="java.util.Date" %>
<%
Date d=new Date();
out.println("Today Date is :"+d);
%>
</body>
</html>
```

Example: **“welcome.html”**

```
<html>
<head>
<title>welocme</title>
</head>
<body>
<h1> Welcome to JSP</h1>
</body>
</html>
```

Output:



Scripting Elements

- The scripting elements provides the ability to insert java code inside the jsp.
- JSP Scripting element are written inside **<% %>** tags. These code inside **<% %>** tags are processed by the JSP engine during translation of the JSP page.
- Any other text in the JSP page is considered as HTML code or plain text.
- There are five different types of scripting elements

Scripting Element	Example
Comment	<%-- comment --%>
Directive	<%@ directive %>
Declaration	<%! declarations %>
Expression	<%= expression %>
Scriptlet	<% scriplets %>

Scripting Elements

Comment:

- JSP comments are used to provide descriptive information or hide code from JSP container.
- The syntax of JSP Comment :

  <%-- comment **--%>**

Example:

```
<%-- this is comment line --%>
```

Declaration:

- JSP declarations are used to define variables or methods.
- The syntax of JSP Declaration :

  <%! Declaration of variable or method **%>**

Example:

```
<%!
        int a=10;
%>
```

Scripting Elements

Expression:

- The code placed within **expression tag** is written to the output stream of the response. So you need not write out.println() to write data. It is mainly used to print the values of variable or method.
- The syntax of JSP Expression:

  <%= Expression **%>**

Example:

```
<%= (2*5) %>
```

Scriptlet:

- Scriptlet Tag allows you to write java code inside JSP page.
- The syntax of JSP Scriptlet :

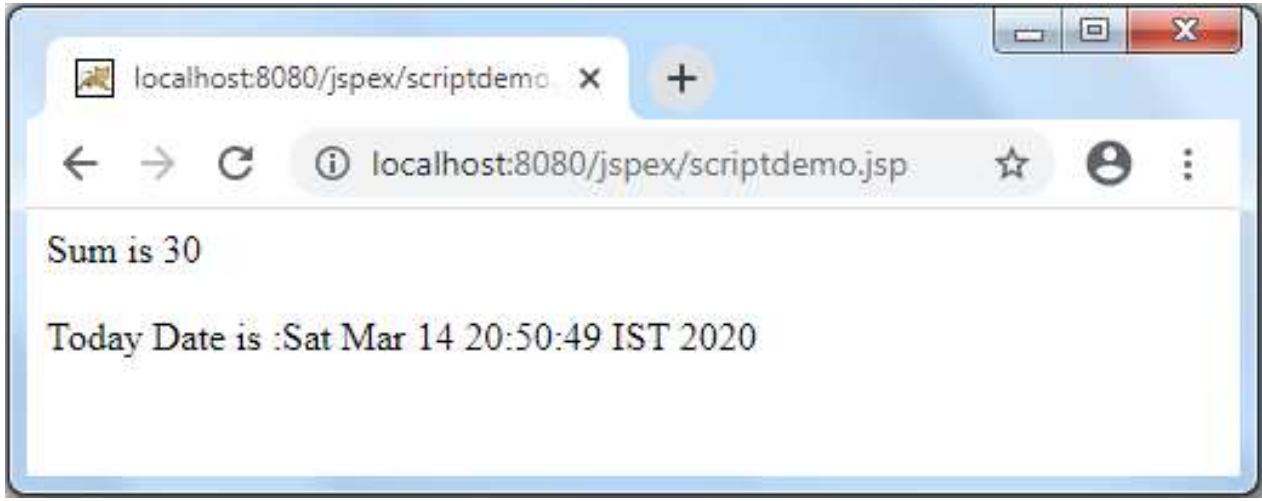
  <% Java Code **%>**

Example:

```
<%
        out.println("Welcome to JSP");
%>
```

Example: “scriptdemo.jsp”

```
<%@page import="java.util.Date" %>
<html>
<body>
<!-- Comments-->
<%-- JSP Comments --%>
<!-- Declaration-->
<%!
int a=10;
int b=20;
%>
<!-- Expression-->
<%="Sum is "+(a+b) %>
<br/><br/>
<!-- Scriptlet-->
<%
Date d=new Date();
out.println("Today Date is :"+d);
%>
</body>
</html>
```



Action Elements

- JSP provides a bunch of standard action elements or tags that we can use for specific tasks such as working with java bean objects, including other resources, forward the request to another resource etc.

Directives vs Actions

- Directives are used during translation phase while actions are used during request processing phase.
- Unlike Directives Actions are re-evaluated each time the page is accessed.

Syntax for the Action element

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

Action Elements

- The list of standard JSP action elements are given below.

Action Element	Description
jsp:include	To include a resource at runtime, can be HTML, JSP or any other file
jsp:useBean	To create a new object of java bean.
jsp:forward	To forward the request to another resource.
jsp:text	To write template text in JSP page.
jsp:element	To define the XML elements dynamically.
jsp:attribute	To define the attributes for XML elements.
jsp:body	To define the body for XML element.

Action Elements

<jsp:include> Element :

- The **jsp:include action tag** is used to include the content of another resource it may be jsp or html.

- The jsp:include tag can be used to include static as well as dynamic pages.

- The jsp include action tag includes the resource at request time so it is **better for dynamic pages.**

Syntax:

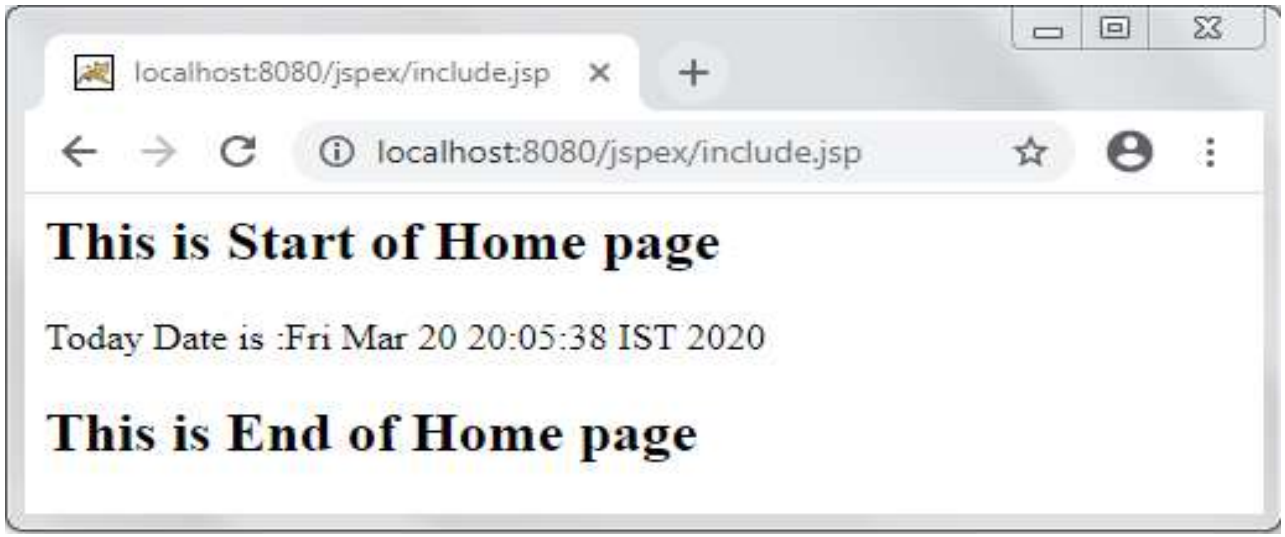
```
<jsp:include page="relativeURL" />
```

Action Elements

Example: "include.jsp"

```
<html>
<body>
<h2>This is Start of Home page</h2>
<jsp:include page="printdate.jsp" />
<h2>This is End of Home page</h2>
</body>
</html>
```

Output:



Action Elements

<jsp:forward> Element :

- The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

- The **forward** action terminates the action of the current page and forwards the request to another resource such as a static page, another JSP page, or a Java Servlet.

- <jsp:forward> is used for redirecting the request.

Syntax:

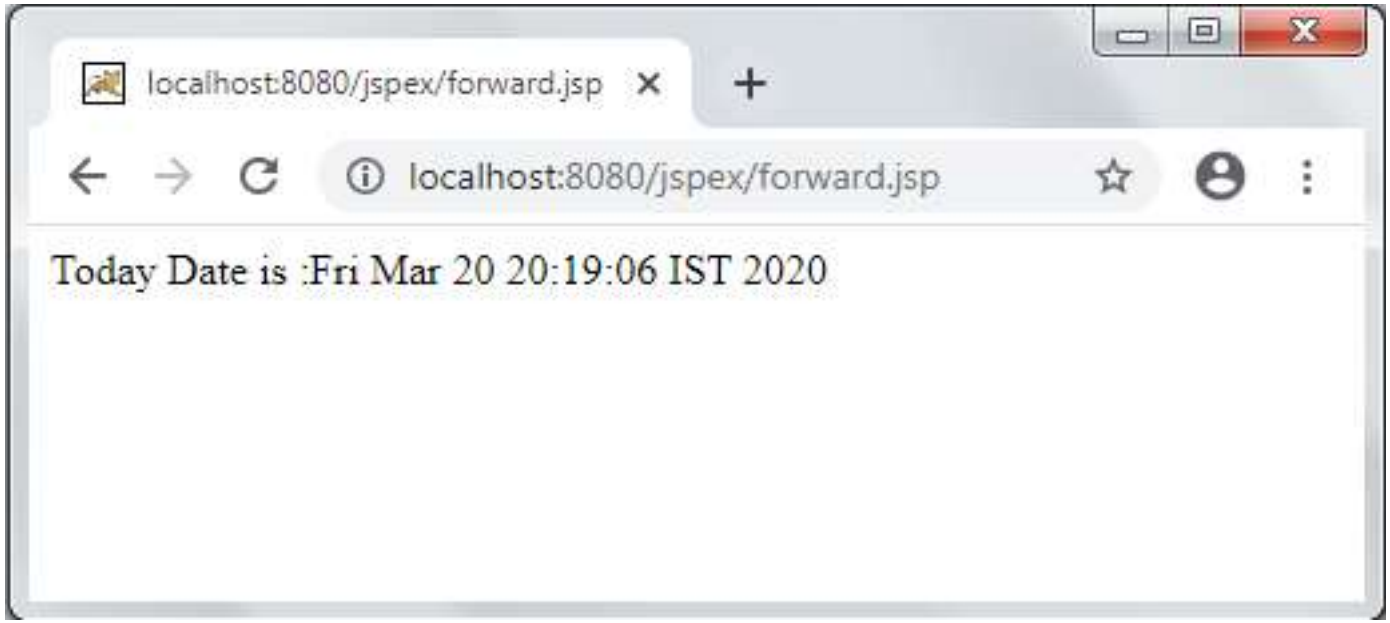
```
<jsp:forward page="URL of the another page" />
```

Action Elements

Example: **“forward.jsp”**

```
<html>
<body>
<h2>This is Start of Home page</h2>
<jsp:forward page="printdate.jsp" />
<h2>This is End of Home page</h2>
</body>
</html>
```

Output:



Using Beans in JSP Pages

Using Beans in JSP Pages

- A Java Bean is a specially constructed Java class written in the Java.
- Java Beans are java classes that have properties, which can be read via a get method or changed via a set method.
- A Java Bean should not have any public variables. All the variables should be accessed using the get/set methods.

JavaBeans Properties

- A JavaBean property is a named attribute that can be accessed by the object. The attribute can be of any Java data type, including the classes that you define.
- JavaBean properties are accessed through two methods in the JavaBean's implementation class.

Using Beans in JSP Pages

get**PropertyName**()

- For example, if property name is *firstName*, your method name would be **getFirstName()** to read that property.

set**PropertyName**()

- For example, if property name is *firstName*, your method name would be **setFirstName()** to write that property.

Accessing JavaBeans

- Following JSP action elements are required to use Java bean in a JSP file.
  - **<jsp:useBean>**
  - **<jsp:getProperty>**
  - **<jsp:setProperty>**

Load Java bean inside a JSP :

- We need to use **<jsp:useBean>** tag to load a bean into JSP.
- The basic syntax is given below:

<jsp:useBean id="objname" class="ClassName" />

Getting the Properties of the Bean :

- After the bean gets loaded into the page, the properties can be accessed by using the **<jsp:getProperty>** action element.
- The basic syntax is given below:

<jsp:getProperty name="objname" property="propertyname"/>

Setting the Properties of the Bean :

- To modify (or) assign value to any variable of the bean object the **<jsp:setProperty>** action element is used .
- The basic syntax is as below

<jsp:setProperty name="objname" property="name" value="anyvalue"/>

Example: "**Employee.java**"

Step 1: Create Java Bean

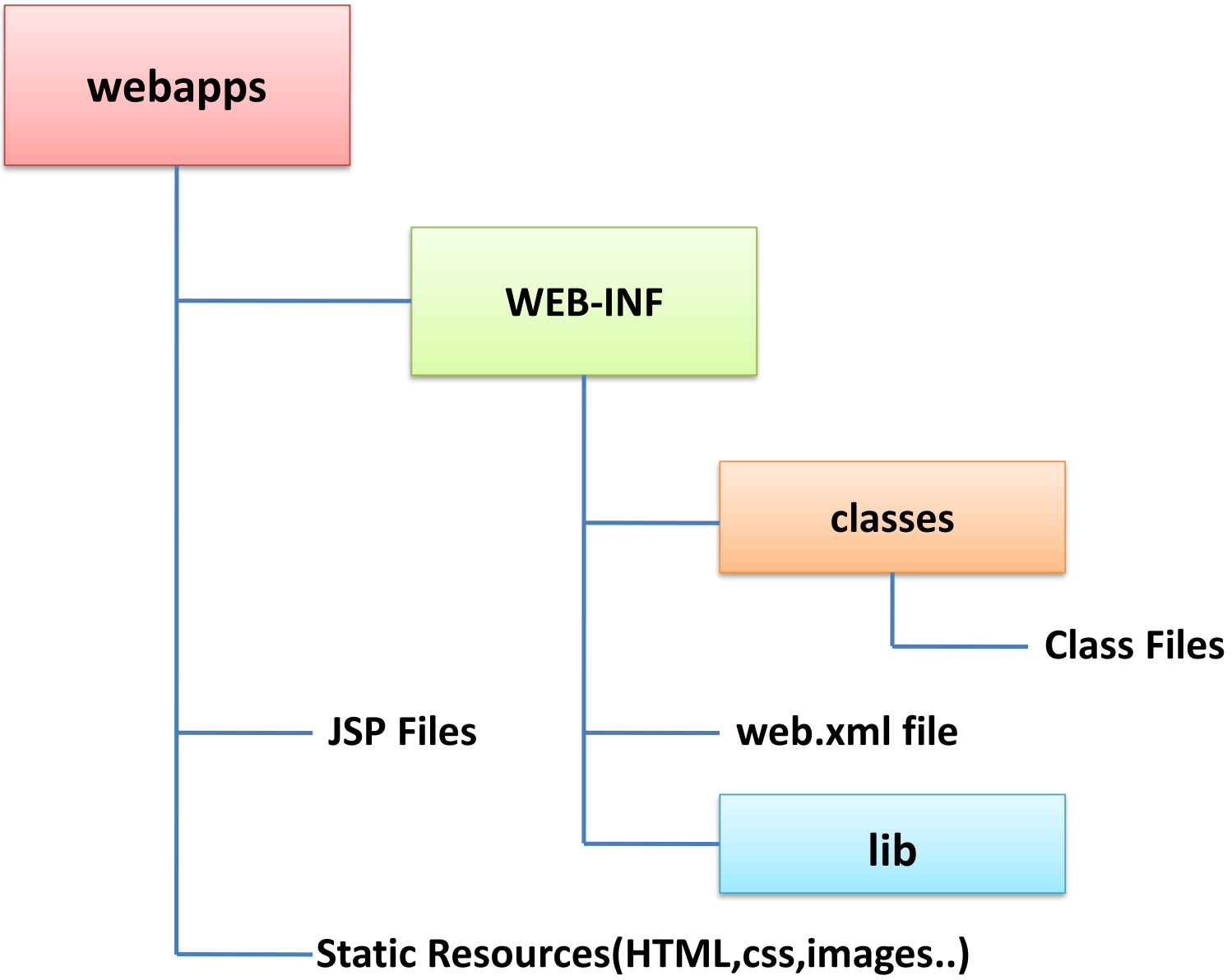
```
package college;
public class Employee {
    private String name = "Madhu";
    private String department = "CSE";
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getDepartment() {
        return department;
    }
    public void setDepartment(String department) {
        this.department = department;
    }
}
```

Example: “jbeandemo.jsp”

Step 2: Create JSP

```
<html>
<head> <title> Java Bean Demo</title></head>
<body>
<jsp:useBean id="employee" class="college.Employee" />
<h2>Old Employee details </h2>
Name: <jsp:getProperty name="employee" property="name" /><br/>
Department :<jsp:getProperty name="employee" property="department" />
<jsp:setProperty name="employee" property="name" value="Durga" />
<jsp:setProperty name="employee" property="department" value="CSE" />
<h2> New Employee details </h2>
Name:<jsp:getProperty name="employee" property="name" /><br/>
Department :<jsp:getProperty name="employee" property="department" />
</body>
</html>
```

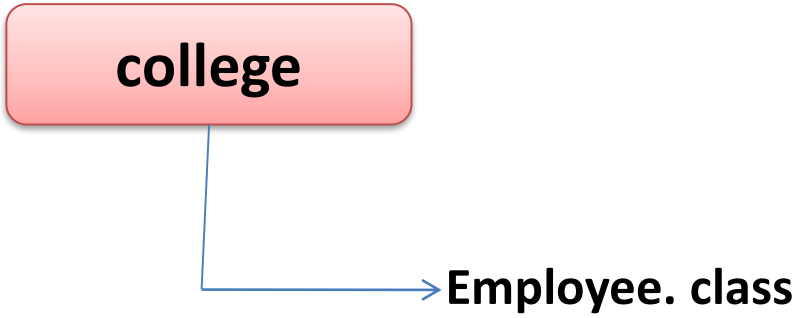
Step 3: Create Folder Structure



Execution Procedure:

1.We need to compile **Employee.java** file.

2.After successful compilation we get package along with class file.



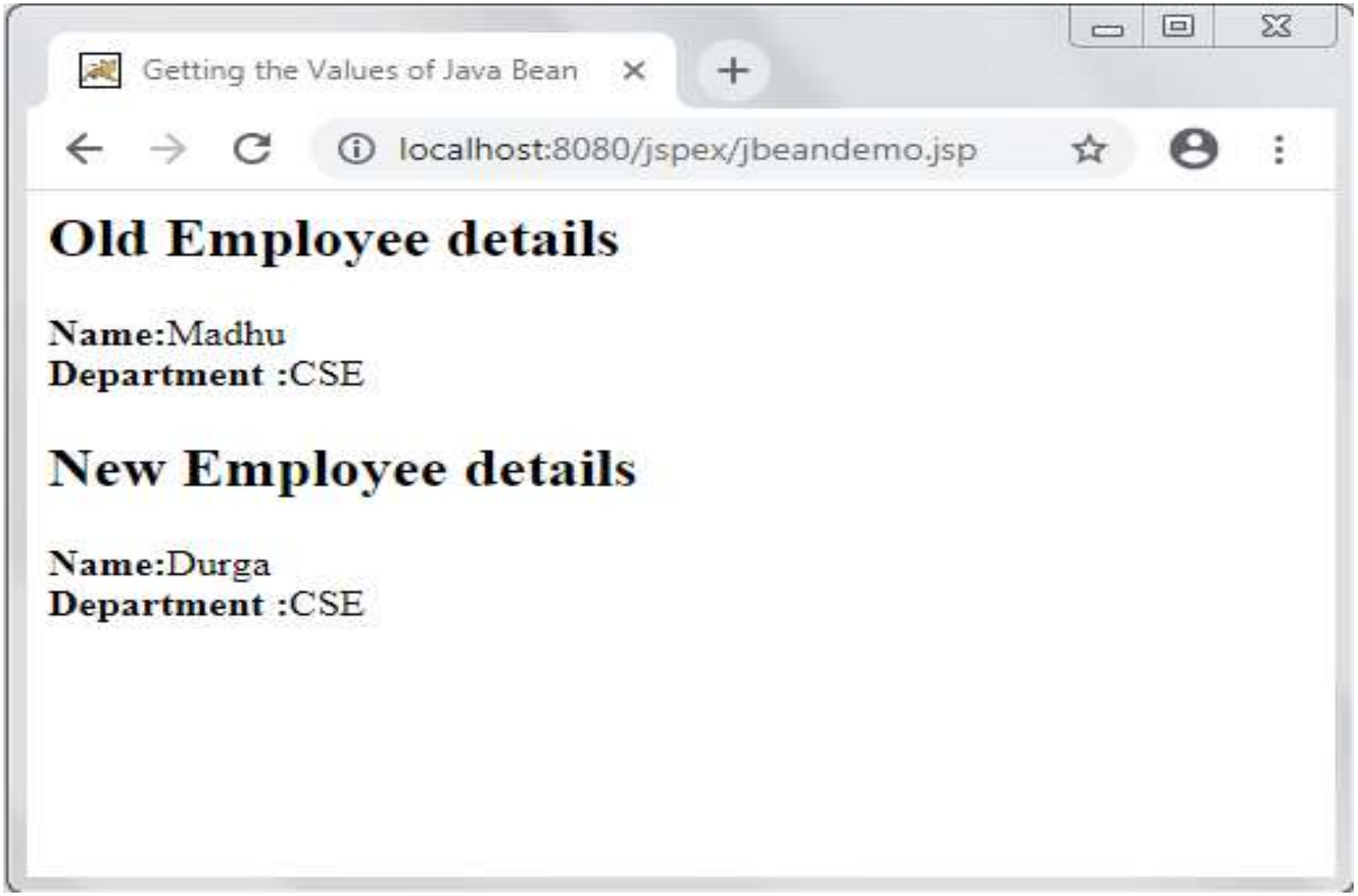
3.We need to copy **Employee. class** along with **college** package into **classes** folder which was in **WEB-INF** folder.

4.After copying, We need to execute JSP file(**jbeandemo.jsp)** using Tomcat server.

i.e., open any browser give following as URL

localhost:8080/foldername/jbeandemo.jsp

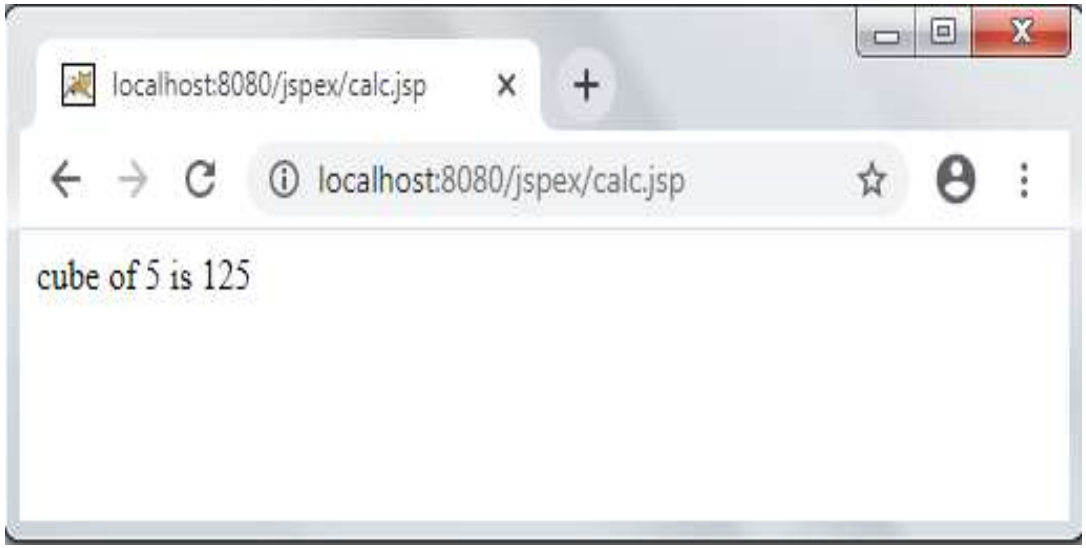
Output:


Getting the Values of Java Bean
localhost:8080/jspex/jbeandemo.jsp
Old Employee details
Name:Madhu
Department :CSE
New Employee details
Name:Durga
Department :CSE

Example: “Calculator.java”

- **Create Java Bean**

```
package mycalc;
public class Calculator{
public int cube(int n)
    {
        return n*n*n;
    }
}
```

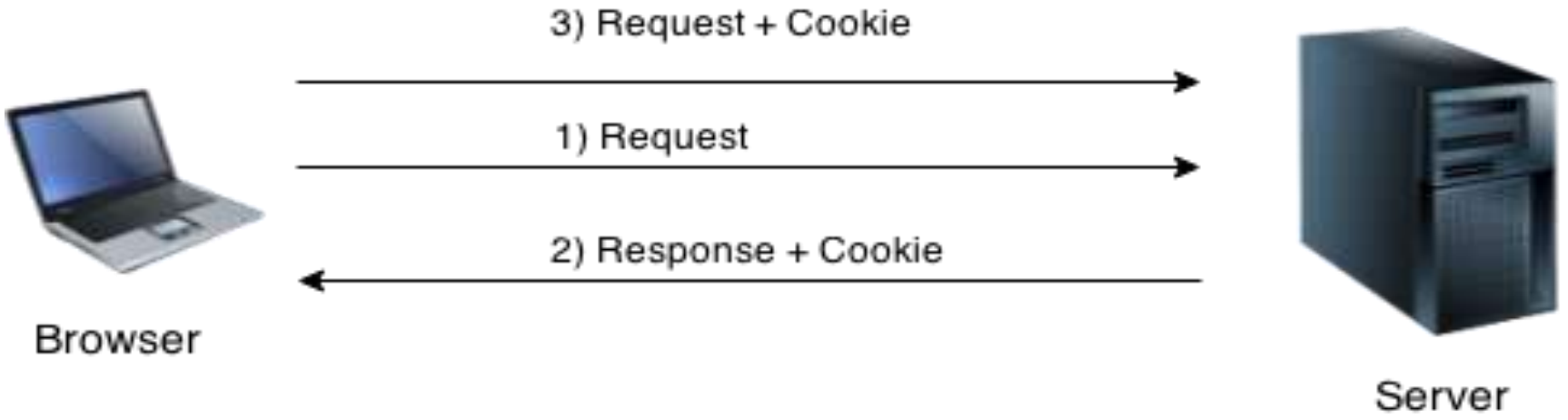


- **Create JSP file** “calc.jsp”

```
<jsp:useBean id="obj" class="mycalc.Calculator"/>
<%
int m=obj.cube(5);
out.print("cube of 5 is "+m);
%>
```

Cookies in JSP

- Cookie is a small piece of information which is stored at client browser. It is used to recognize the user.
- Cookie is created at server side and saved to client browser.
- First time when client sends request to the server, then server embedded the cookie along with Response.
- Next time onwards client sends request along with cookie to the server. Such way, cookie can be received at the server side.



- Cookies are text files stored on the client computer and are used to store information.
- There are given some commonly used methods of the Cookie class.

Method	Description
setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
String getName()	Returns the name of the cookie.
String getValue()	Returns the value of the cookie.
void setName(String name)	sets the name of the cookie.
void setValue(String value)	sets the value of the cookie.
void addCookie(Cookie ck)	used to add cookie in response object.
Cookie[] getCookies()	used to return all the cookies from the browser.

✓ **Create Cookie:**

The following code shows how to create a cookie

```
Cookie ck=new Cookie("user",”abcd");    //creating cookie object
response.addCookie(ck);    //adding cookie in the response
```

✓ **Get Cookie:**

The following code shows how to get all cookies

```
Cookie ck[ ]=request.getCookies();      //get cookie values
```

✓ **Delete Cookie:**

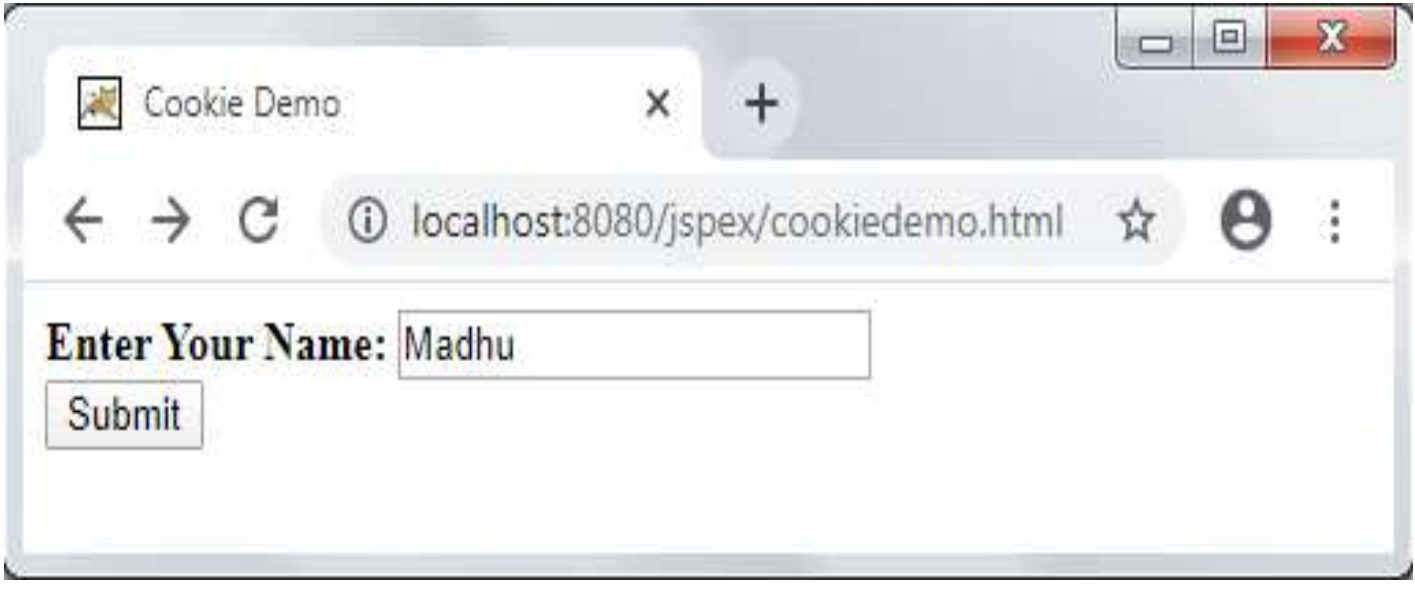
The following code shows how to delete a cookie

```
Cookie ck=new Cookie("user","");//deleting value of cookie
ck.setMaxAge(0);//changing the maximum age to 0 seconds
response.addCookie(ck);//adding cookie in the response
```

Example:

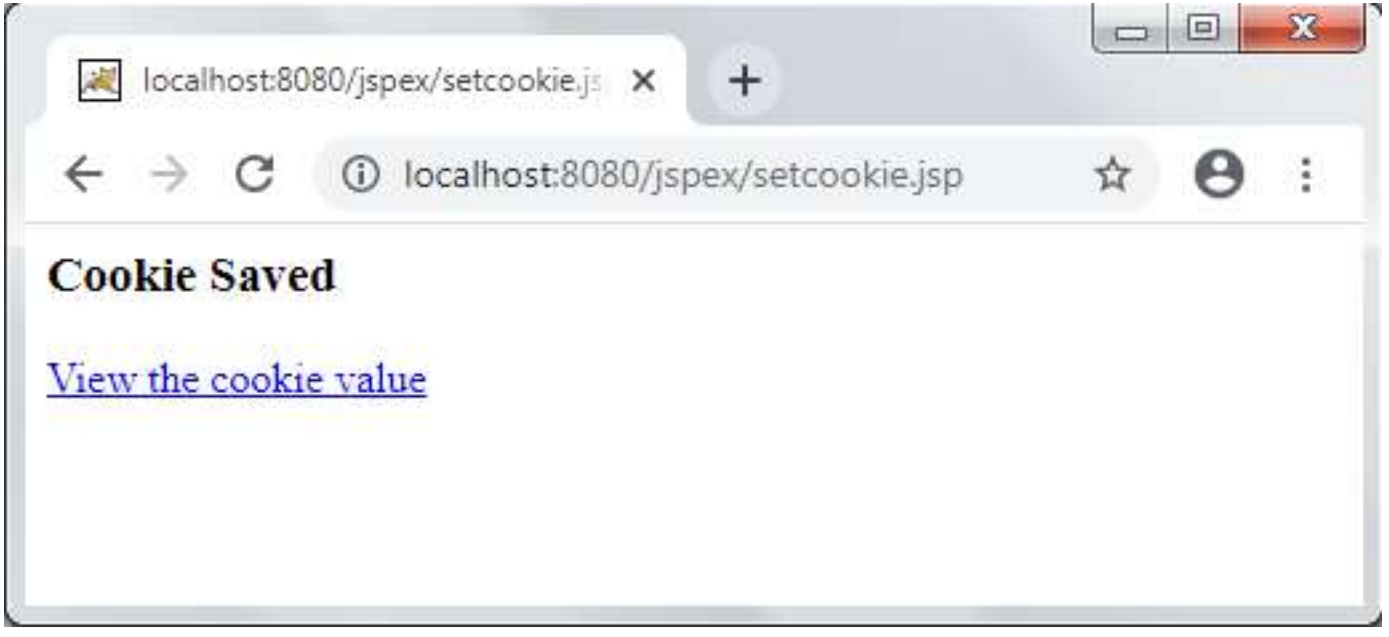
cookiedemo.html

```
<html>
<head>
<title>Cookie Demo</title>
</head>
<body>
<form method="post" action="setcookie.jsp">
<p><b>Enter Your Name: </b>
<input type="text" name="username"><br>
<input type="submit" value="Submit">
</form>
</body>
</html>
```



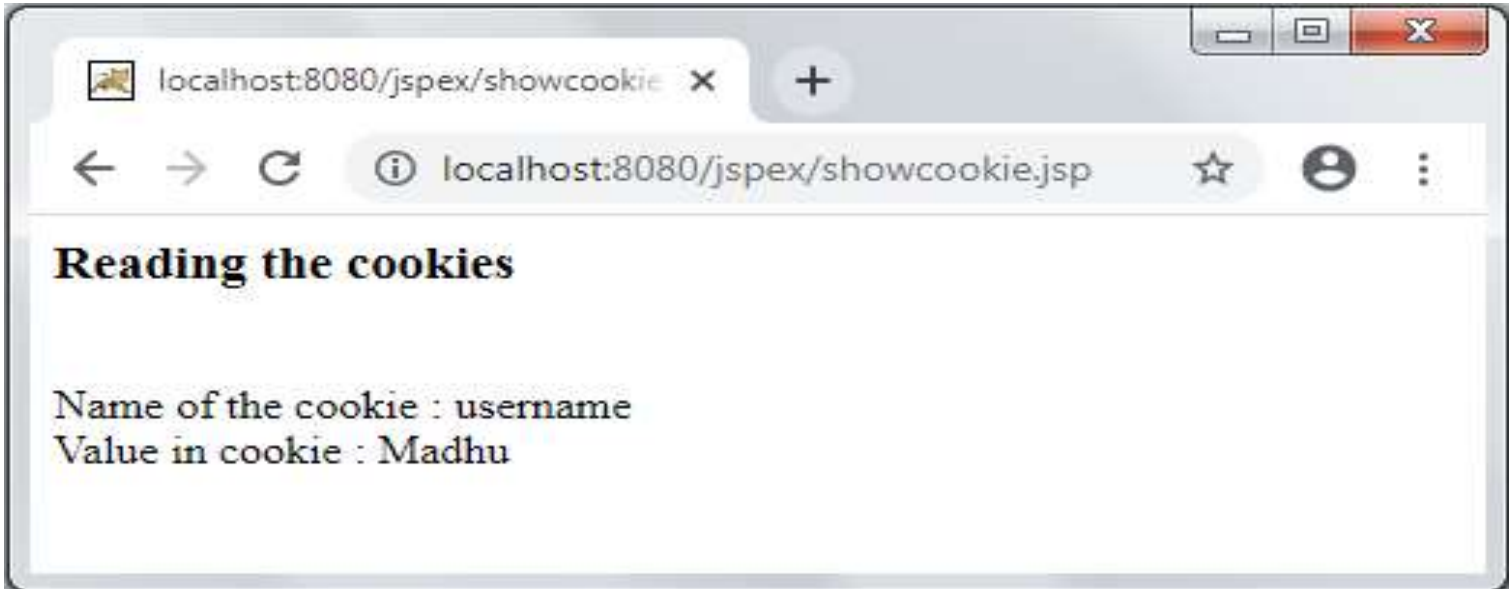
setcookie.jsp

```
<%
String uname=request.getParameter("username");
Cookie ck = new Cookie ("username",uname);
ck.setMaxAge(1* 60);
response.addCookie(ck);
%>
<html>
<body>
<h3>Cookie Saved</h3>
<p><a href="showcookie.jsp“>View the cookie value</a><p>
</body>
</html>
```



showcookie.jsp

```
<html>
<body>
<h3>Reading the cookies</h3>
<%
Cookie[] array= request.getCookies();
for(int i=0; i<array.length; i++)
{
if(array[i].getName().equals("username"))
{
out.println("<br/>");
out.println("Name of the cookie : " + array[i].getName() + "<br/>");
out.println("Value in cookie : " + array[i].getValue());
}
}
%>
</body>
</html>
```



Connecting to Database in JSP

- All the web applications usually interact with different types of databases. Databases are basically used to store various types of information for different purposes.
- JSP pages are looks like regular html pages with the inclusion of dynamic behavior.
- JSP pages allows the developer to interact with database by using JDBC code.
- By using Scriptlets, we can include database programming (JDBC code) in JSP.

- The Architecture of accessing a database by using JSP.

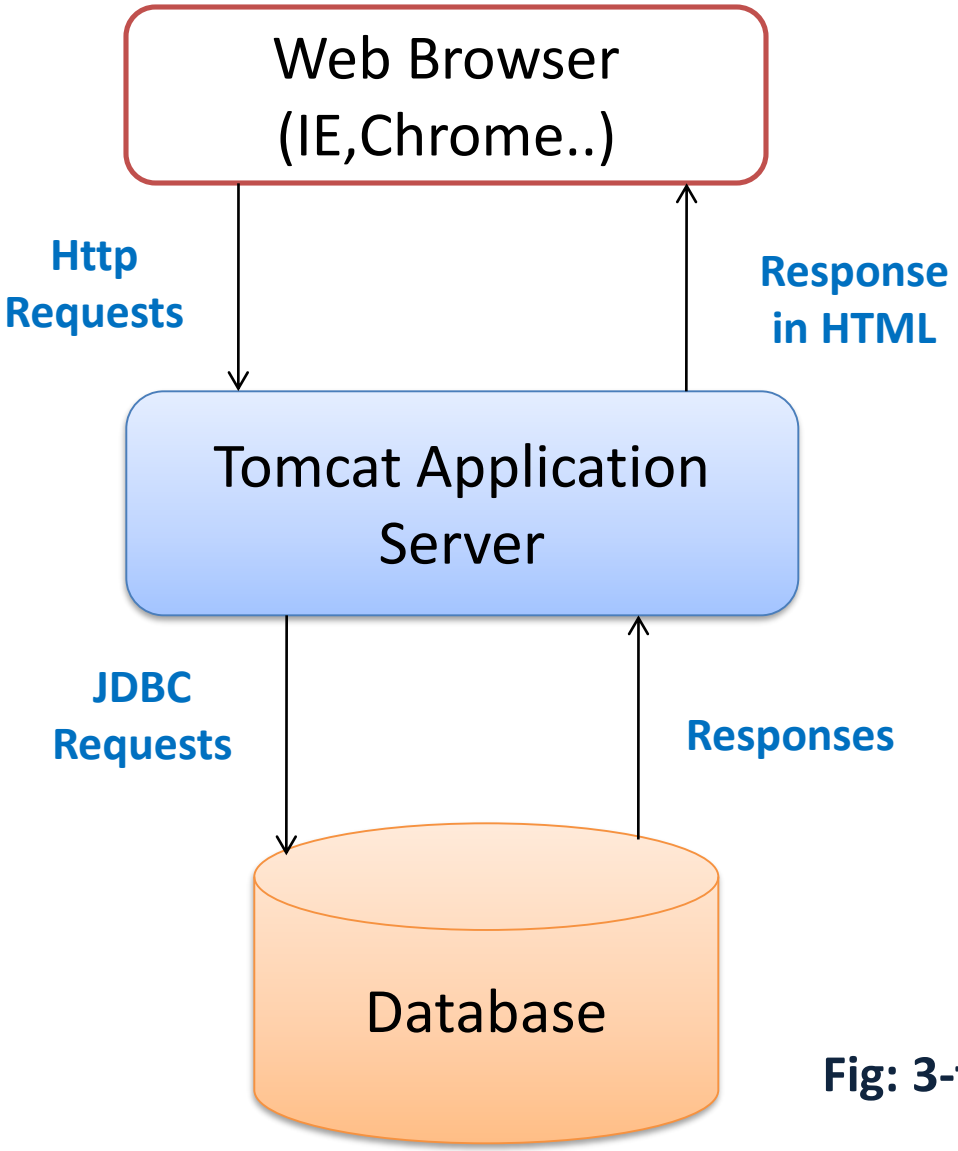


Fig: 3-tier Architecture

- For Database Connectivity we write the following code in jsp page.

✓ **Open Connectivity Code:**

```
<%@page language="java" import="java.sql.*"%>
<%
// Load driver class file of Oracle
class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","password");
 // Load driver class file of MySql
class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
" jdbc:mysql://localhost:3306/TestDb","root","root");
%>
```

- ✓ **Statement Code:**

```
<%
// Create the statement
Statement stmt=conn.CreateStatement();
String query="SELECT * from students"
stmt.executeQuery(query);
%>
```

- ✓ **Close connectivity Code:**

```
<%
// Close the Statement
stmt.close();
// Close the connection
conn.close();
%>
```

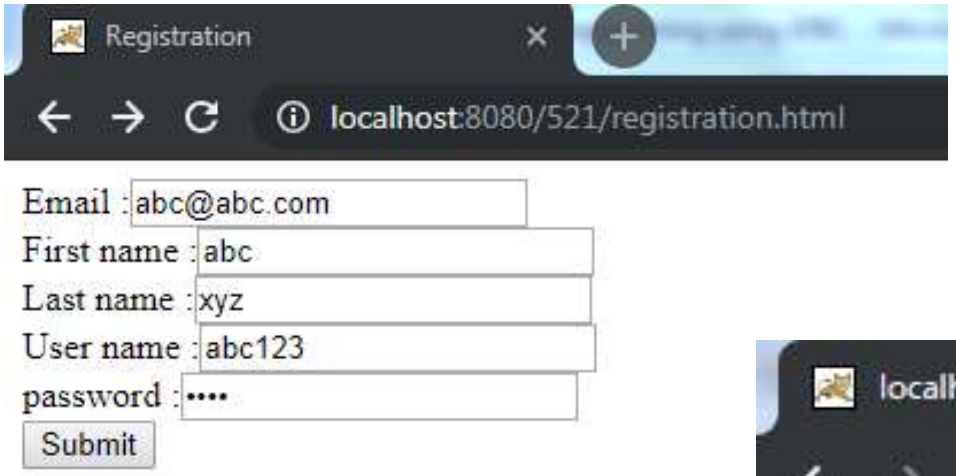
Example: Registration.html

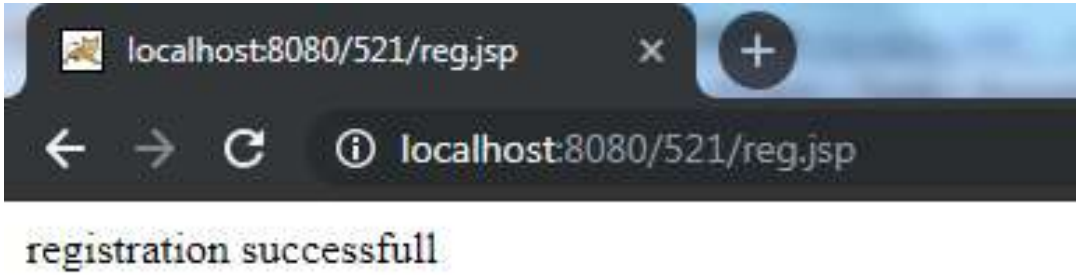
```
<html >
 <head> <title>Registration</title> </head>
 <body>
 <form action="reg.jsp" method="post">
Email :<input type="text" name="email" /><br/>
First name :<input type="text" name="fname" /><br/>
Last name :<input type="text" name="lname" /><br/>
User name :<input type="text" name="userid" /><br/>
password :<input type="password" name="pwd" /><br/>
<input type="submit" />
</form>
 </body>
</html>
```

reg.jsp

```
<%@ page import ="java.sql.*" %>
<%@ page import ="javax.sql.*" %>
<%
String user=request.getParameter("userid");
String pwd=request.getParameter("pwd");
String fname=request.getParameter("fname");
String lname=request.getParameter("lname");
String email=request.getParameter("email");
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection
        ("jdbc:oracle:thin:@localhost:1521:xe","system","system");
Statement st= con.createStatement();
String query="insert into users values
    ('"+user+"','"+pwd+"','"+fname+"','"+lname+"','"+email+"')“;
st.executeUpdate(query);
st.close();
con.close();
out.println("registration successfull");
%>
```

Output:


Registration
localhost:8080/521/registration.html
Email : abc@abc.com
First name : abc
Last name : xyz
User name : abc123
password : ••••
Submit


localhost:8080/521/reg.jsp
localhost:8080/521/reg.jsp
registration successfull