UNIT - 4

Applet:

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.

Life cycle of an applet:

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

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

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

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

5. paint(Graphics g): is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Passing parameters to an applet.

```
import java.applet.Applet; import java.awt.Graphics;

/*<applet code="App004" width=400 height=400>

        <param   name="username"   value="Shiva">

</applet> */

public class App004 extends Applet {

String  s;

public void init()   {    s = getParameter("username");   }

public void paint(Graphics g)   {   g.drawString(s,100,100);   }

}
```

Adding images to an applet

```
import java.awt.*;  import java.applet.*;

 /*  <applet code="App005"  height=600 width=600>   </applet>*/

public class App005 extends Applet {

Image picture;

public void init() {    picture = getImage(getDocumentBase(),"img1.jpg");    }
public void paint(Graphics g) {     g.drawImage(picture, 30,30, this);    }
}
```

Adding sound to an applet.

```
import java.applet.*; import java.awt.*;import java.awt.event.*;
 /*   <applet code="App006"  height=600 width=600>   </applet>*/
public class App006 extends Applet implements ActionListener {
Button   b1,b2;    AudioClip  audioClip;
public void init()  {
b1 = new Button("Play");     add(b1);
b1.addActionListener(this);
b2 = new Button("Stop");     add(b2);
b2.addActionListener(this);
audioClip = getAudioClip(getCodeBase(), "audio.wav");
}
public void actionPerformed(ActionEvent ae)  {
Button source = (Button)ae.getSource();
if (source.getLabel().equals("Play"))    {
audioClip.play();
}
else if(source.getLabel().equals("Stop")) {
audioClip.stop();
}}}
```

Event Handling.

Event: Changing the state of an object is known as an event.

For example, click on button, dragging mouse etc.

Event Handling:

Java uses event delegation model for handling/processing the event.

Terminology in event handling:

1. Source: like button, checkbox, applet,..etc. on which user has generated the event.

2. Type of Event: example if the user clicks on a button, the type of event is called as the action event. If the user clicks on the applet, it is mouse event.

3. Listener: The class(s) who wants to know about a particular event and respond.

4. Event Registration: the classes who are interested in knowing and processing any type of event on any source must register for the same.

Event Delegation Model: When an event is generated on the source, JRE will create an object of corresponding Event class with the information about generate event and delegates to the registered listeners. Now the listener class will invoke the corresponding method which processes the event.

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

Evnet Classes & Listener Interfaces:

ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener

Handling Mouse Events – Example

```
// making use of inner classes

import java.applet.*; import java.awt.*; import java.awt.event.*;

//<applet code="a9" width=400 height=400></applet>

public class a9 extends Applet{

public void init()  {

addMouseListener(new MouseAdapter()  {

                        public void mouseClicked(MouseEvent me)   {

                            showStatus("mouse clicked");

                          }
                        }
                  );
    }
}
```

Handling Keyboard Events – Example

```
import java.applet.*; import java.awt.*; import java.awt.event.*;
//<applet code="a6" width=400 height=400></applet>
public class a6 extends Applet implements KeyListener{
int x=30,y=30;String s="";
public void init()  {  addKeyListener(this);  }
public void keyTyped(KeyEvent ke)  {
    s+=ke.getKeyChar();
    repaint();
}
public void keyPressed(KeyEvent ke)  {
showStatus("key pressed");
}
public void keyReleased(KeyEvent ke)  {
showStatus("key Released");
}
public void paint(Graphics g)  {
g.drawString(s,x,y);
}
}
```

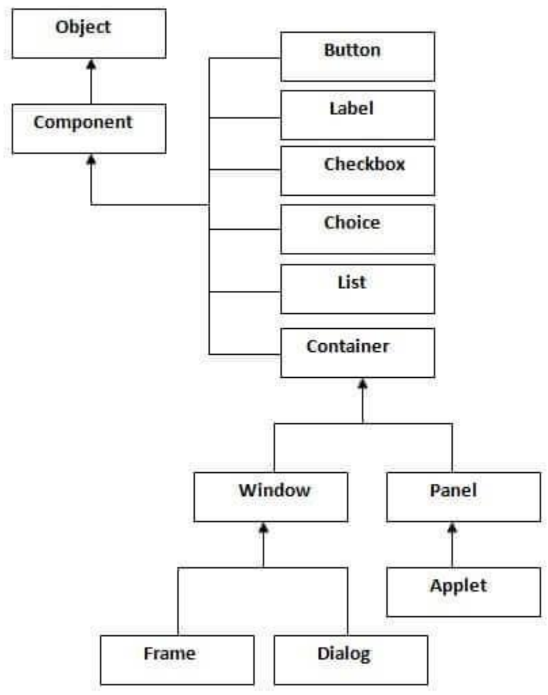
Introducing AWT:

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

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

AWT Classes Hierarchy:



Container: The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window: The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

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

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

Checkbox – Example

```
import java.applet.*; import java.awt.*;import java.awt.event.*;

//<applet code="a14" width=400 height=400></applet>

public class a14 extends Applet implements ItemListener{
```

```
Checkbox cb1,cb2;
String s1="win98      : ";
String s2="winNT      : ";
public void init()  {
cb1=new Checkbox("win98");    add(cb1);
cb1.addItemListener(this);
cb2=new Checkbox("winNT");    add(cb2);
cb2.addItemListener(this);
}
public void itemStateChanged(ItemEvent ie)  {
repaint();
}
public void paint(Graphics g)  {
g.drawString(s1+cb1.getState(),40,100);
g.drawString(s2+cb2.getState(),40,120);
}
}
```

<u>**Choice (Drop down List) – Example**</u>

```
import java.applet.*; import java.awt.*; import java.awt.event.*;
//<applet code="a16" width=400 height=400></applet>
public class a16 extends Applet implements ItemListener{
Choice ch1,ch2;
String msg1,msg2;
public void init()  {
ch1=new Choice();
ch1.add("rahul");
ch1.add("sachin");
ch1.add("saurav");
add(ch1);
ch1.addItemListener(this);
}
public void itemStateChanged(ItemEvent ie)  {
repaint();
}
public void paint(Graphics g)  {
msg1="Cricekter Name  :  ";
msg1+=ch1.getSelectedItem();
g.drawString(msg1,40,100);
}
}
```

Layout Managers

The Layout Managers are used to arrange components on the container in a particular manner.

1.BorderLayout: used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only.

Example:

Container_component.add(component,BorderLayout.NORTH);

2. GridLayout is used to arrange the components in rectangular grid.

Example:

Container_componet.setLayout(new GridLayout(3,3));

Container_componet.add(component_1);

3. FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

Example:

Container_componet.setLayout(new FlowLayout(FlowLayout.RIGHT));

Container_componet.add(component_1);

4. 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.

Example-Card Layout:

```
c=getContentPane();
card=new CardLayout(40,30);
//create CardLayout object with 40 hor space and 30 ver space
c.setLayout(card);
b1=new JButton("Apple");
b2=new JButton("Boy");
c.add("a",b1); c.add("b",b2);
```

Menus – Example:

```
class myframe extends Frame implements ActionListener{
String msg="";
myframe(String name)   {
super(name);
MenuBar mbar=new MenuBar();
setMenuBar(mbar);
Menu file=new Menu("File");
MenuItem new1,open,close;
file.add(new1= new MenuItem("New"));
file.add(open= new MenuItem("Open"));
file.add(close=new MenuItem("Close"));
mbar.add(file);
Menu edit=new Menu("Edit");
mbar.add(edit);
open.addActionListener(this);
close.addActionListener(this);
public void actionPerformed(ActionEvent ae)   {
msg=(String)ae.getActionCommand();
repaint();
}
public void paint(Graphics g)   {
g.drawString(msg,100,100);
}
}
```

Servlets

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

CGI: Before servlets CGI(Common Gateway Interface) technology was used to develop dynamic web pages. The CGI programs can be designed in the native OS and kept in particular directory. Web Servers communicates with these external program via an interface called CGI.

CGI Vs Servlet

BASIS FOR COMPARISON	CGI	SERVLET
Basic	Programs are written in the native OS.	Programs employed using Java.
Platform dependency	Platform dependent	Does not rely on the platform
Creation of process	Each client request creates its own process.	Processes are created depending on the type of the client request.
Conversion of the script	Present in the form of executables (native to the server OS).	Compiled to Java Bytecode.
Runs on	Separate process	JVM
Security	More vulnerable to attacks.	Can resist attacks.
Speed	Slower	Faster

BASIS FOR COMPARISON	CGI	SERVLET
Processing of script	Direct	Before running the scripts it is translated and compiled.
Portability	Can not be ported	Portable

Life Cycle of a Servlet:

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

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

3) **init()** method is invoked: The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet.

public void init(ServletConfig config) throws ServletException

4) **service()** method is invoked: The web container calls the service method each time when request for the servlet is received.

public void service(ServletRequest request, ServletResponse response)

5) **destroy()** method is invoked: The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc.

public void destroy()

Servlet API

Interfaces	Description
Servlet	Declares life cycle methods that all servlets must implement.
ServletConfig	Allows servlets to get initialization parameters
ServletContext	Allows servlets to communicate with its servlet container.
ServletRequest	Provides client request information to a servlet.
ServletResponse	Assist a servlet in sending a response to the client.

Classes	Description
GenericServlet	Provides a basic implementation of the Servlet interface for protocol independent servlets
ServletInputStream	Provides an input stream for reading binary data from a client request.
ServletOutputStream	Provides an output stream for sending binary data to the client.
ServletException	Defines a general exception, a servlet can throw when it encounters difficulty.

Interfaces and classes from java.servlet.http package

Impotant Interfaces in javax.servlet.http package

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession

Important Classes in javax.servlet.http package

1. HttpServlet
2. Cookie

important methods of HttpServletRequest

Methods	Description
Cookies getCookies()	returns an array containing all of the Cookie objects the client sent with this request
HttpSession getSession()	returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session
String getMethod()	Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
String getServletPath()	returns the part of this request's URL that calls the servlet

Some Important Methods of HttpServletResponse

Methods	**Description**
void addCookie(Cookie cookie)	adds the specified cookie to the response.
void sendRedirect(String location)	Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer
int getStatus()	gets the current status code of this response
String getHeader(String name)	gets the value of the response header with the given name.
void setHeader(String name, String value)	sets a response header with the given name and value

Handling Http Request and Http Response

```
// Simple Calculator Example

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

public class serv6 extends HttpServlet {

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

int x=Integer.parseInt(request.getParameter("t1"));

int y=Integer.parseInt(request.getParameter("t2"));

String opr=request.getParameter("s1");

opr=opr.trim();

int op=Integer.parseInt(opr);

int result=0;

switch(op)

{
```

```
case 1: result=x+y; break;
case 2: result=x-y; break;
case 3: result=x*y; break;
case 4: result=x/y; break;
}
PrintWriter pw=response.getWriter();
pw.println("Result = "+result);
}
}
```

Cookies:

Cookie is a bit of information created by server for the first client request and will sent back to the client browser.

For the subsequent client request, all the cookies that the web browser has will be automatically forwarded to the server along with the client request.

Size of the cookies are limited.

Generally cookies will expire when we close the browser window. If we want our cookies to persist for more time, we can set age for cookies.

Generally Cookies are used to exchange key,value pairs between web browser and web server.

Creating & sending cookies to client:

```
Cookie ck1=new Cookie("username","shiva");
response.addCookie(ck1);
```

Accessing all cookies coming from client:

```
PrintWriter out=response.getWriter();
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++)    {
out.print(ck[i].getName()+"     ");
out.println(ck[i].getValue());
}
}
```

Session Tracking:

Session simply means a particular interval of time.

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

Http protocol is a stateless so we need to maintain state using session tracking techniques.

Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of the user to recognize to particular user.

Creating an instance of current date and time and storing in session

```
Date d1=new Date();

HttpSession s1=request.getSession();

s1.setAttribute("mydate",d1);
```

getSession() in the above code returns a reference to the existing session if exists else create a new session and returns that session object.

setAttribute() helps use store our data(objects) with some name.

getSession() method also returns a cookies to the client browser with this new session id, so when the same client make the second request, server will compare the session id value inside the cookie to identify the client.

Accessing the data(date object) from session:

```
HttpSession s=request.getSession(false);

Date d=(Date)s.getAttribute("mydate");

response.getWriter().println(d);
```

JSP

Problems with Servlets

The Servlet itself contains the code for all aspects of the application like Request processing, Business logic & Presentation logic.

html code(presentation logic) is embedded inside java code(business logic).

1. Changes in the look and feel (presentation) of the application requires the Servlet to be updated and recompiled.

2. Developer needs proficiency in both web page designing and java.

3. Difficult to use development tools.

for example, if we use a tool to develop a user interface design, the generated html code has to be manually inserted with in the java code which is time consuming and error prone.

Benefits of JSP:

1. JSP Separates business logic and presentation logic.

i) Presentation logic(html code) can be written in a jsp file directly(as if you are writing in a HTML file).

ii) Business logic (java code) should be written inside JSP elements.

2. JSP has built objects, which helps in reducing development effort.

Anatomy of a JSP File

1. Template text:

it can be plane text, html code, XML code or any other markup language code.

2. JSP Elements:

i) Scripting elements (declarations, expressions, scriptlets)

ii) Directives (page, include & taglib)

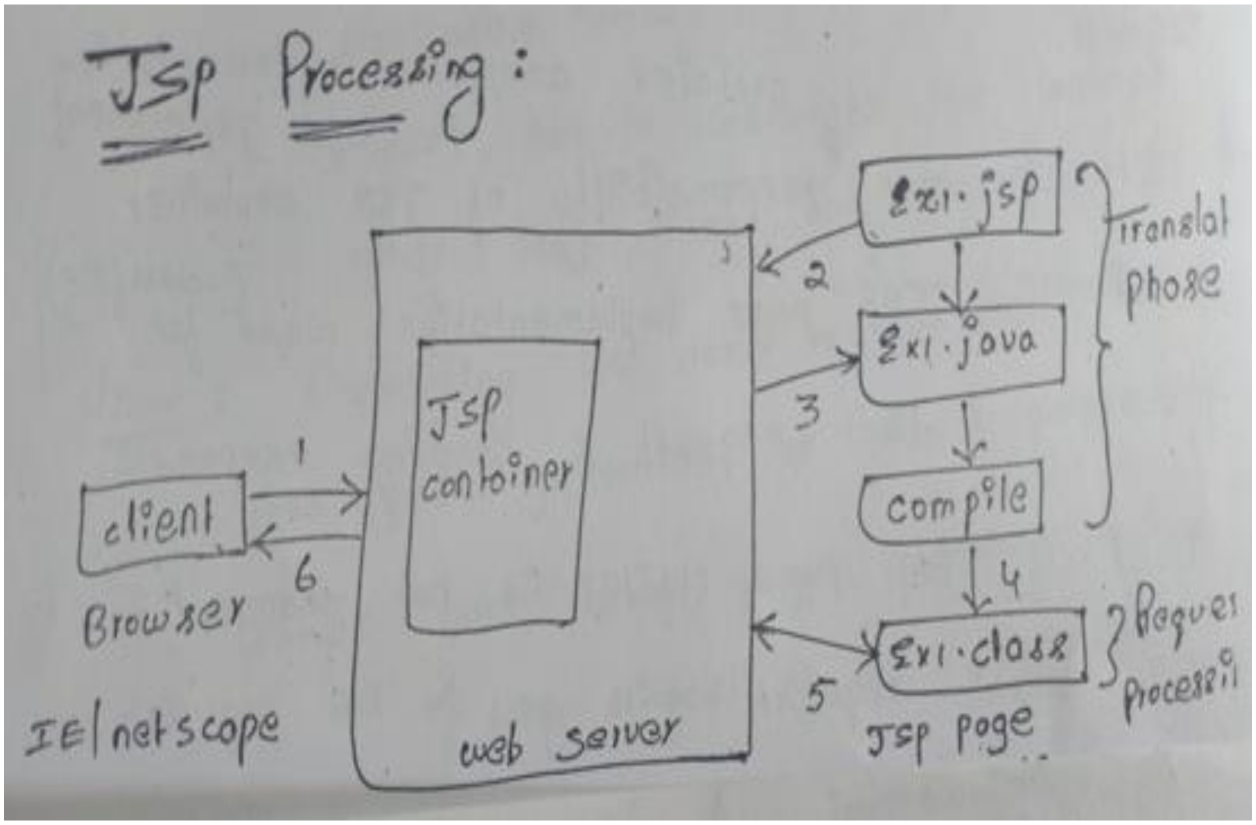
iii) Actions

iv) EL (Expression Language) expressions.

v) Comments.

When a JSP file is requested by the client, the template text is merged with dynamic content generated by JSP elements and sent as a response back to the client.

JSP Processing:



Translation phase:

When the first client sends a request for a JSP file, JSP file will be converted into a Servlet.

The “.jsp ” converted to “.java”

i) The html code present in the JSP file will be converted to the corresponding println() statements.

ii) The JSP elements will be converted to the java code that implements the corresponding dynamic behaviour.

The converted “.java” file is compiled and generates “.class” file.

This “.class” file is called as jsp page implementation class.

Then an instance for the jsp page implementation class is created.

Request processing phase:

The container invokes service() to process client request.

MVC Design Pattern

1. Model: Represents Business logic.

Responsible for generating the content(data) what the client has requested for.

Generally the java code that access the data from Database(if required) , process the data and generates the response requested by the client.

2. View: Represents Presentation logic.

Responsible for presenting the content(data) generated by the Model components.

We can have more than on View component for one Model component.

Technologies used are: i) HTML ii) JSP

3. Controller: Represents Request Processing.

The Controller is responsible for controlling the application logic and acts as the coordinator between the View and the Model.

The Controller receives an input from the users via the View, then processes the user's data with the help of Model and passes the results back to the View for displaying the results to client browser.

Technologies used are

i) Java Beans ii) Enterprise JavaBeans iii) POJO (Plane Old Java Objects)

JSP Elements

1.Directives: Directives are translation time instruction to the JSP container.

We have 3 type of directives : page, include & taglib.

example of page directive:

```
<%@   page   language="java"
        contentType="text/html”
        extends="demotest.DemoClass“
                    import="java.util.Date“
                    session=“true/false“
                    isThreadSafe="true/false“
                    isErrorPage=“true/false“
                    errorPage="errorHandler.jsp“%>
```

2. JSP Scripting Elements:

1.Declarations:

A declaration tag is a piece of Java code for declaring variables and methods.

If we declare a variable or method inside declaration tag it means that the declaration is made inside the Servlet class but outside the service() method.

Example

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

JSP Elements

2. Scripting Elements

Scriptlets:

1. Allows you to write Java code in JSP file.
2. JSP container moves statements in _jspservice() method while generating Servlet from JSP.

Example:

```
<%

int  num1=10;

int  num2=40;

int  num3 = num1+num2;

out.println("Scriplet Number is " +num3);

%>
```

3.Actions

We can dynamically insert a file, reuse the bean components, forward user to another page, etc. through JSP Actions Unlike directives, actions are re-evaluated each time the page is accessed.

Example

```
<jsp:useBean    id="name“  class="demotest.DemoClass">

<jsp:include     page="date.jsp“ />

<jsp:forward    page="jsp_action_42.jsp" />
```

JSP Built in Objects – Servlet Classes

request(HttpServletRequest)

response(HttpServletResponse)

out(JSPWriter)

config(ServletConfig)

exception(Throwable)

session(HttpSession)

application(ServletContext)

page

pageContext