Unit-IV

APPLETS:

Java applets- Life cycle of an applet – Adding images to an applet – Adding sound to an applet. Passing parameters to an applet. Event Handling. Introducing AWT: Working with Windows Graphics and Text. Using AWT Controls, Layout Managers and Menus. Servlet – life cycle of a servlet. The Servlet API, Handling HTTP Request and Response, using Cookies, Session Tracking. Introduction to JSP.

Java Applets

when a HTML page wants to communicate with the user on internet, It can use a special program called '**Applet**' to communicate and respond to the user. The user can interact by typing some details or by clicking the components available in the application. The program then process and displays the results. We can understand an applet as a java bye code embedded in a HTML page, generally for the purpose of achieving communication with the user. We can think of an applet as:

Applet=java Byte code+ HTML page

What is an Applet?

An applet represents java byte code embedded in a web page.

(or)

Applet:- Applet are small applications that are accessed on an internet server, transported over the internet, automatically installed, and run as part of a web document.

Creating an applet

To create an applet, we need to write a java program and compile it to byte code. Then we should embed it into HTML page on particular location wherever we want it to be displayed. This page is then stored in the web server. A client machine communicates with the web server, the server then sends the HTML page that contains the applet. This page is transmitted to the client where the applet is executed on the client browser. Thus applets are executed at client side by the web browser.

To create an applet, we have **Applet** class of **java.applet** package. These classes use the following methods, which are automatically run by any applet program. So these methods should be overridden in the applet program.

1. **init()**

This method is the first method to be called by the browser and it is executed only once. So the programmer can use this method to initialize any variables, creating components and creating threads etc. when this method is completed browser looks for the next method **start().**

2. **start()**

This method is called after init() method and each time the applet is revisited by the user. For example the user has minimized the web page that contains the applet and moved to another page then this method's execution is stopped. When the user come back to view the web page again, start() method execution will resume. Any calculations and processing of data should be done in this method and results are also displayed.

3. **stop()**

This method is called by the browser when the applet is to be stopped. If the user minimizes the web page, then this is called and when the user again comes back to this page, then start() method is called. In this way start() and stop () methods can be called repeatedly.

4. **destroy()**

This method is called when the applet is being terminated from memory. The stop() method will always be called before destroyed. The code related to releasing the memory allocated to the applet and stopping any running threads should be written in this method().

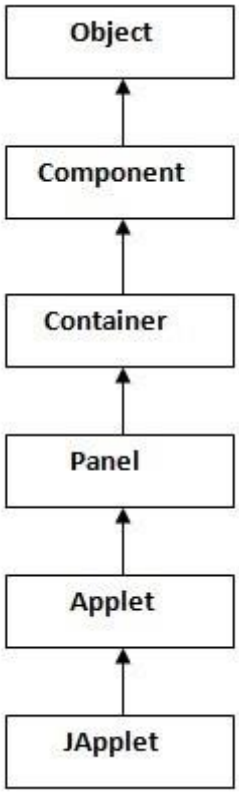
Executing init(), start(), stop() and destroy methods in that sequence is called **"life cycle of an applet".**

5. **paint(Graphics g)**

This method is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc. This method is Invoked immediately after the start()

method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt. package.

Hierarchy of Applet



What is applet life cycle?

An applet is born with init() method and starts functioning with start() method. To stop the applet the stop() method is called and to terminate completely from the memory ,the destroy() method is called. Once the applet is terminated, we should reload the HTML page to get the applet start once again from init() method. This cyclic way of executing the methods is called applet life cycle.

Uses of Applet

Applets can be used for multiple purposes. Some of the applications are:

1. Applets are used for creating dynamic web pages. For example a student can type his Hall Ticket Number in a text filed and click the retrieve button to get back his results from his University server. Applets are useful to provide such interaction with the user at runtime.

2. Another use of applets is for creating animation and games where the images can be displayed or moved giving a visual impression that they are alive

How to run an Applet

To run an applet we can use one of the methods below.

1. **By AppletViewer**:-This provided by sun along with JDK. The appletviewer takes the name of the HTML file containing the applet tag.
2. **By html file:-** Embed the <applet> in the html file and execute in web Browser.

How to use <Applet> Tag

<applet> tag is useful to embed an aplet into an HTML page. It has the following form

< APPLET [**CODEBASE** = *codebaseURL*] **CODE** = *appletFile*
[**ALT** = *alternateText*] [**NAME** = *appletInstanceName*]
WIDTH = *pixels* **HEIGHT** = *pixels*
[**ALIGN**=*align*][**VSPACE**=*pixels*] [**HSPACE**= *pixels*]
[< **PARAM NAME** = *appletParameter1* **VALUE** = *value* >]
[< **PARAM NAME** = *appletParameter2* **VALUE** = *value*>]
. . .
</APPLET>

Where

CODEBASE = Path of the applet class file
CODE = *name of the applet class file*
ALT = *alternate text to be displayed*
NAME = *applet Instance Name*
WIDTH = maximum width of applet in *pixels*
HEIGHT = maximum height of applet in *pixels*
ALIGN = *alignment(*left, right, top, middle, bottom).
VSPACE = *pixels*
HSPACE = *pixels*
<PARAM NAME = *appletParameter1* **VALUE** = *value* >

<PARAM> tags are the only way to specify applet-specific parameters. Applets read user-specified values for parameters with the getParameter() method.

/* Demonstarion of life cycle method of Applet . Remember all methods of the applet and applet class itself should be declared 'public'. Otherwise they are not available to the browser to execute.*/

```
import  java.applet.*;

public  class Applet1 extends  Applet
{
public void init()
{
System.out.println("iam in init()");
}

public void start()
{
System.out.println("iam in start()");
}
public void stop()
{
System.out.println("iam in stop()");
}

public void destroy()
{
System.out.println("iam in destroy()");
}}
```

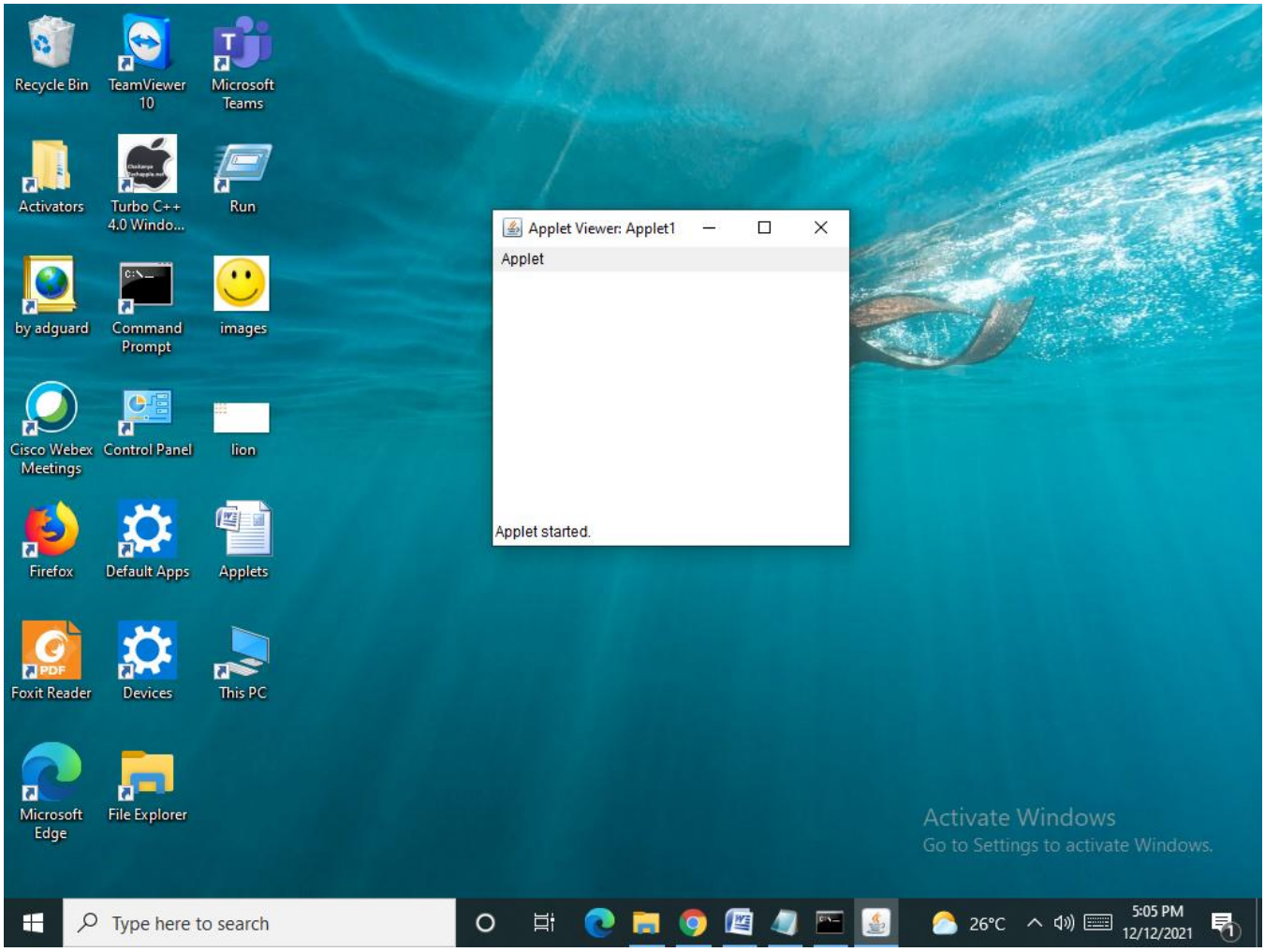
/* <applet code=Applet1 width=200 height=200>
</applet>
***/**

Compile:-

```
E:\applets>javac  Applet1.java
```

Execution:

```
E:\applets>Appletviewer  Applet1.java
```



/* demonstrating of paint method */

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

public class paint extends Applet
{
public void paint(Graphics g)
{
g.drawRect(10,20,30,40);
g.drawOval(50,80,100,150);
g.drawString("This is my first applet application",100,200);
}
}
```

/*

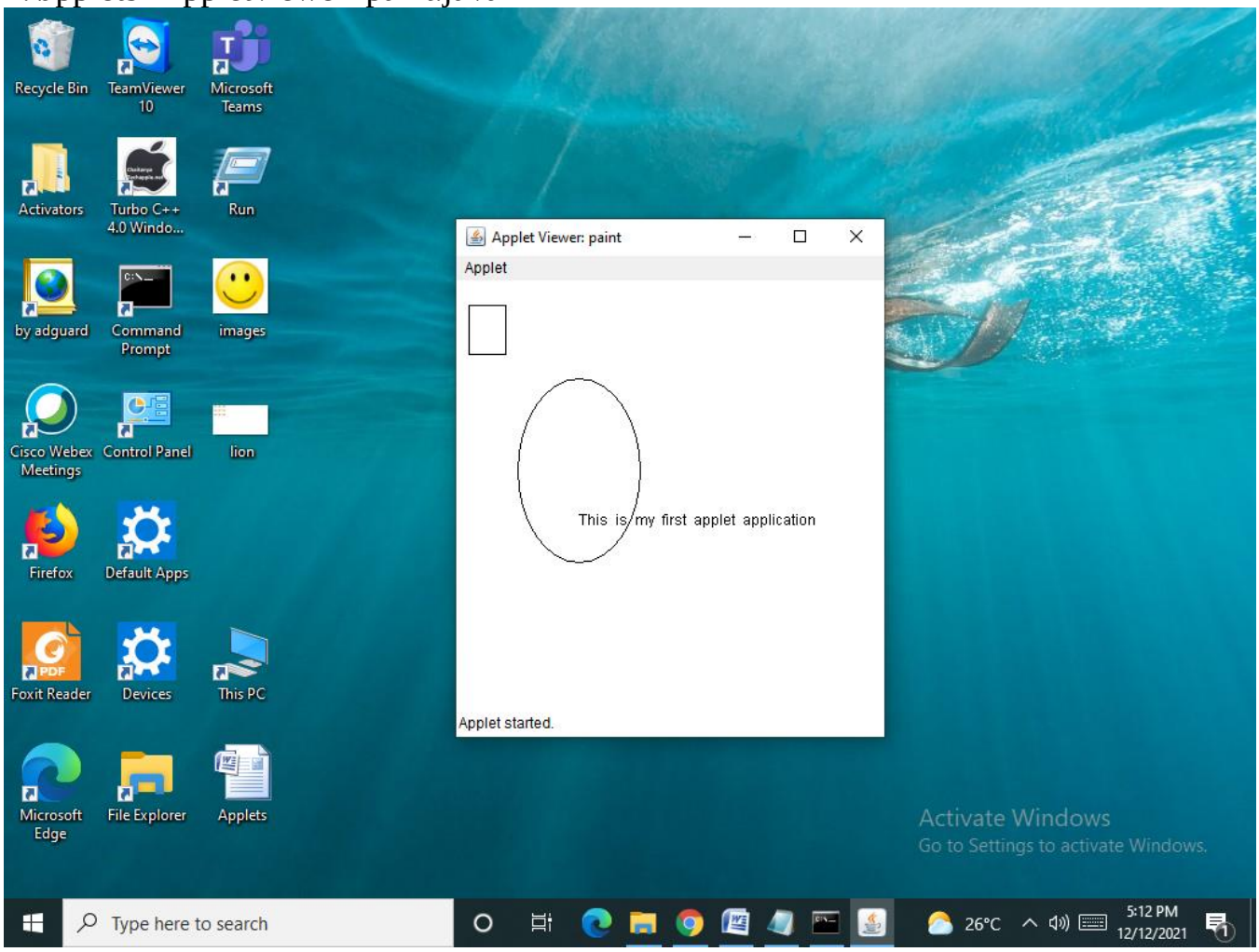
<applet code=paint width=350 height=350>
</applet>
***/**

Compile:-

E:\applets>javac paint.java

Execution:

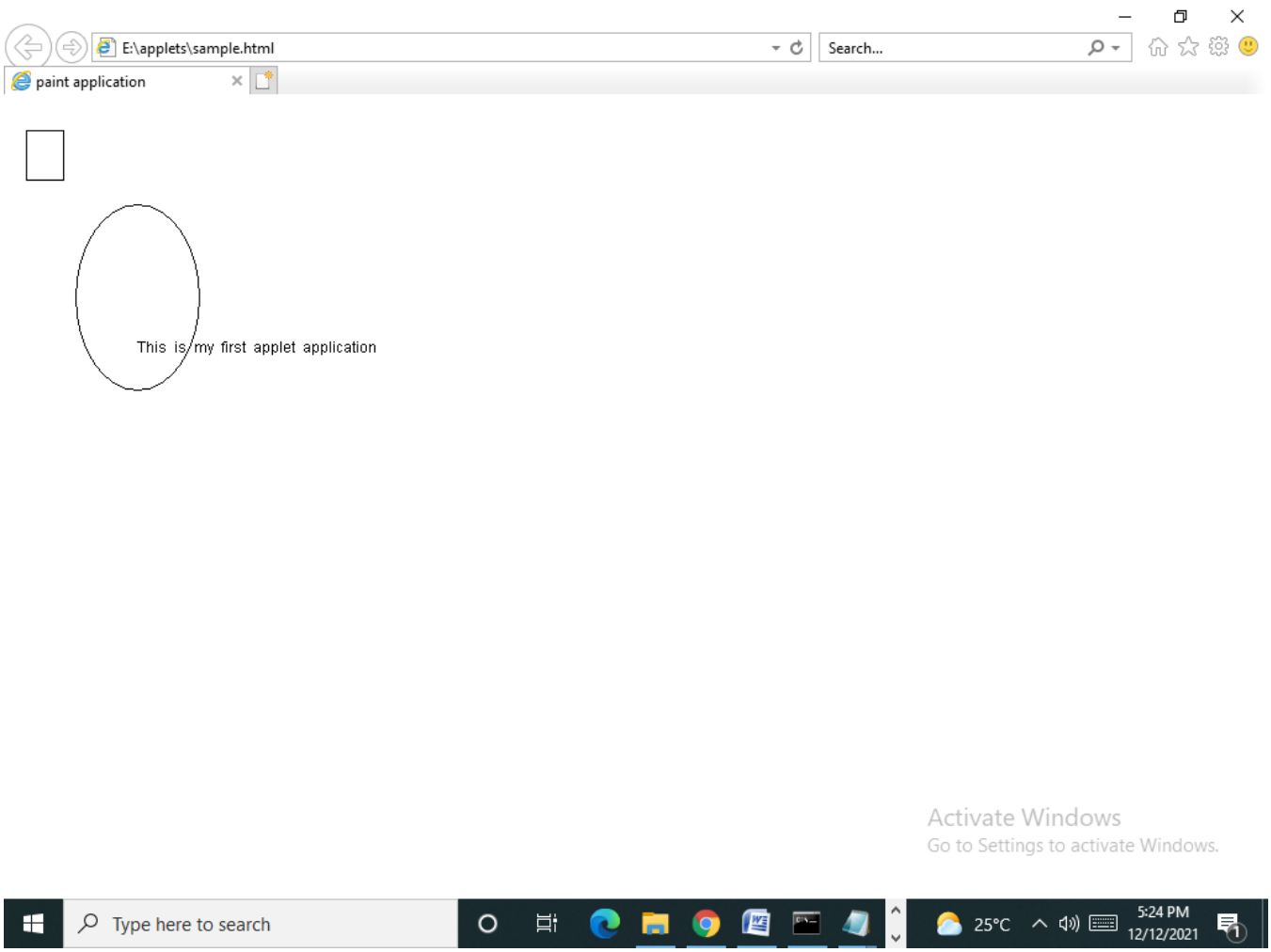
E:\applets>Appletviewer paint.java



Alternative :2 to execute application using HTML file

<html>
<head>
<title>paint application</title>
</head>
<body>
<applet code=paint width=500 height=500>
</applet>
</body></html>

Open the browser and type the name of the html file to execute



Displaying Image in Applet

Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image.

Syntax of drawImage() method:

1. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.

How to get the object of Image:

The java.applet.Applet class provides getImage() method that returns the object of Image.

Syntax:

public Image getImage(URL u, String image){}

Other required methods of Applet class to display image:

1. **public URL getDocumentBase():** is used to return the URL of the document in which applet is embedded.
2. **public URL getCodeBase():** is used to return the base URL.

/* program to demonstate adding image to applet */

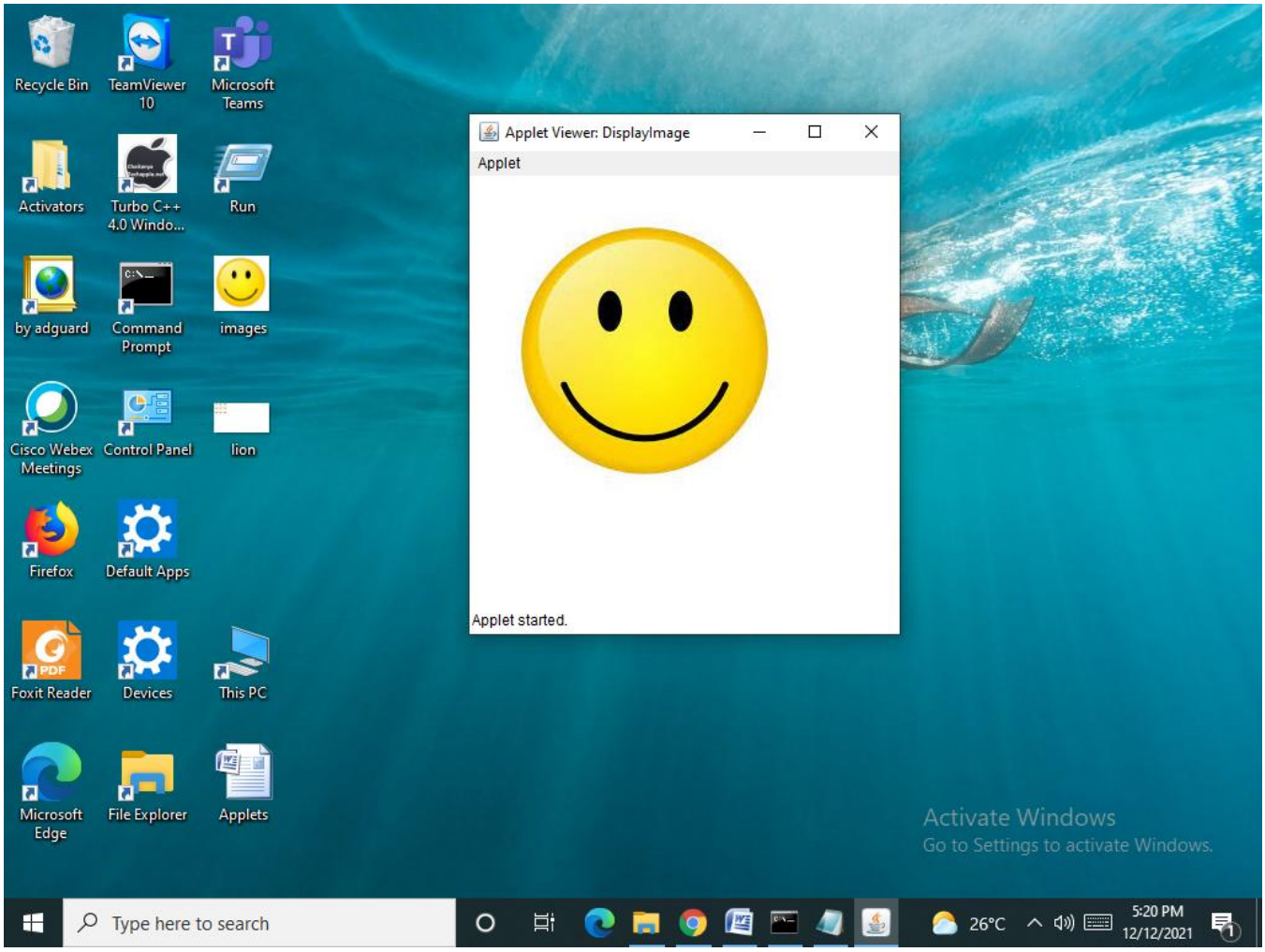
```
import java.awt.*;
import java.applet.*;

public class DisplayImage extends Applet {

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

Compile:-

E:\applets>javac DisplayImage .java

Execution:

E:\applets>Appletviewer DisplayImage.java



Adding sound to an Applet

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

public class PlaySoundApplet extends Applet implements ActionListener
 {
   Button play,stop;
   AudioClip audioClip;

   public void init()
 {
      play = new Button(" Play in Loop ");
      add(play);
      play.addActionListener(this);
      stop = new Button("Stop");
```

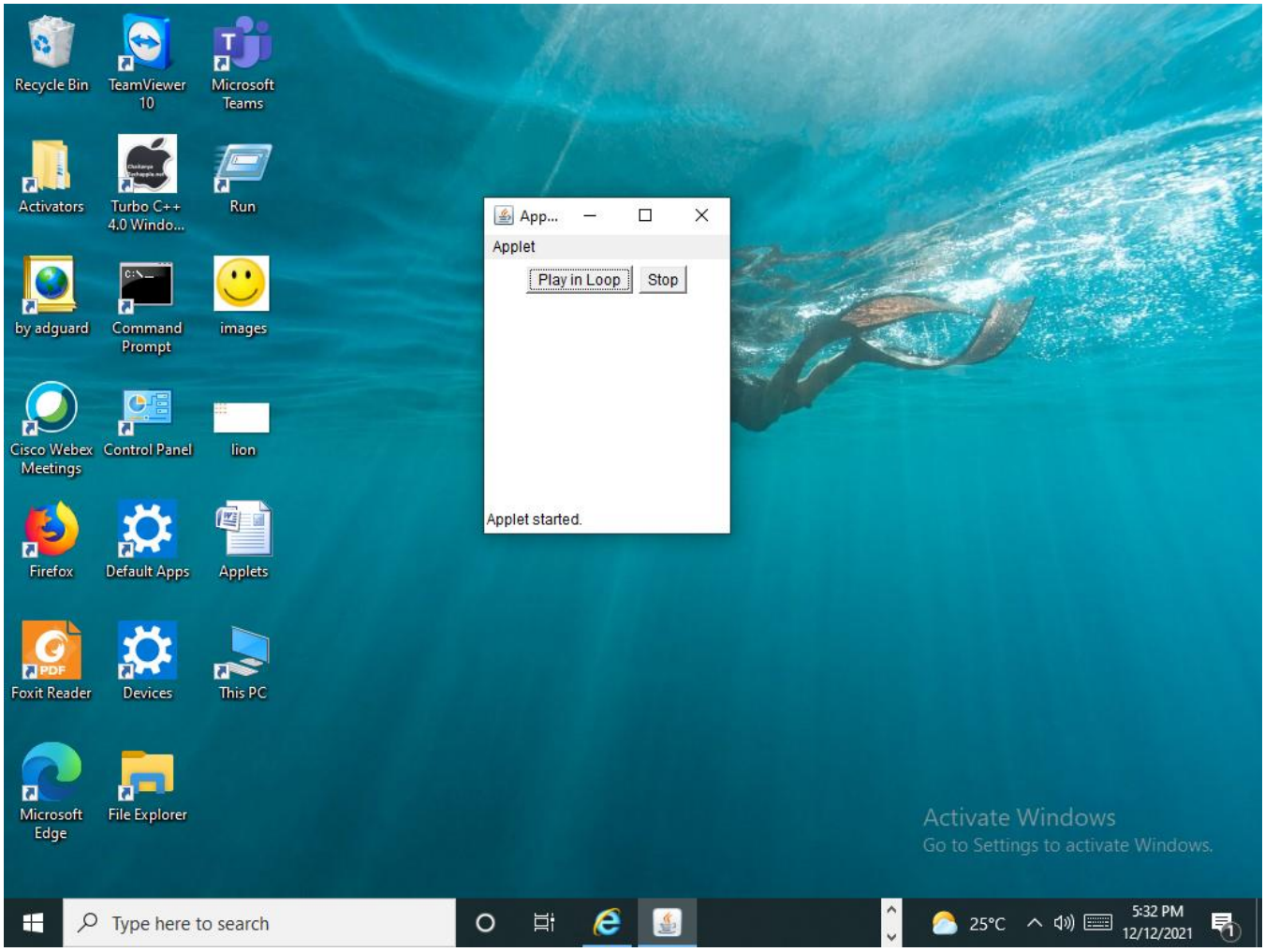
```
    add(stop);
    play.addActionListener(this);
    stop.addActionListener(this);
    audioClip = getAudioClip(getCodeBase(), "start.wav");
   System.out.println(getCodeBase());
  }
  public void actionPerformed(ActionEvent ae) {
    Button source = (Button)ae.getSource();
    if (source.getLabel() == " Play in Loop ") {
      audioClip.loop();
    } else if(source.getLabel() == "Stop"){
      audioClip.stop();
    }
  }
}

/*
<applet  code="PlaySoundApplet"  width=200  height=200>
</applet>
*/
```

Compile:-

E:\applets>javac PlaySoundApplet.java

Execution:

E:\applets>Appletviewer PlaySoundApplet.java



Restrictions on Applet

Applets can show images and animation, play Sounds, take user input and send it to the server etc. Applets cannot interact and soil resources of the client system and hence they are harmless. This restricted environment where the applets are executed is called sandbox. The sandbox provides an applet with some amount of memory to get executed and does not allow the applet to do any illegal operation on the client system. Some Restrictions on applet are

1. Applets can never run executable program in the client.
2. Applets cannot communicate with any host other than the server from which they were downloaded.
3. Applets cannot read from a file or write into a file that belongs to the client machine.
4. Applets cannot load programs stored on the user's system.

Passing parameters to Applet

Along with applet tag, We can use <param> tag to pass information to applets. For example to pass name and salary of an employee to an applet from HTML page, we can write the applet and param tags as:

```
<applet code=tax width=300 height=300>
<param name=t1 value="SriRam">
<param name=t2 value="15000">
</applet>
```

<param> tag has two attributes name and value represents the name of the parameter and value indicates the value. For example in the above code we have two parameters t1 and t2 and their values "SriRam" and "15000" respectively.
To retrieve the values of the parameters, an applet uses getParameter() method. This method takes the parameter name and return its value as a string.

```
name=getParameter("t1");
str=getParameter("t2");
```

where name and str are String type variables. getParameter() method should be used inside the init() method of applet.
<param> tag is useful for passing information to the applets. This eliminates the need for modifying the applet even if we want to pass different data.

/* program to pass parameters to applet*/

```
import java.applet.*;
import java.awt.*;
public class Tax extends Applet
{
String name,str;
float sal;
float tax;
public void init()
{
name=getParameter("t1");
str=getParameter("t2");
sal=Float.parseFloat(str);
calculate(sal);
}
public void calculate(float sal)
{
if(sal<10000)
```

```
tax=0.0f;
else if(sal<=200000)
tax=sal*0.1f;
else
tax=sal*0.2f;
}
public  void  paint(Graphics  g)
{
g.drawString("Hello "+name,20,100);
g.drawString("Your salary "+sal,20,120);
g.drawString("Pay  tax "+tax,20,140);
}
}
```

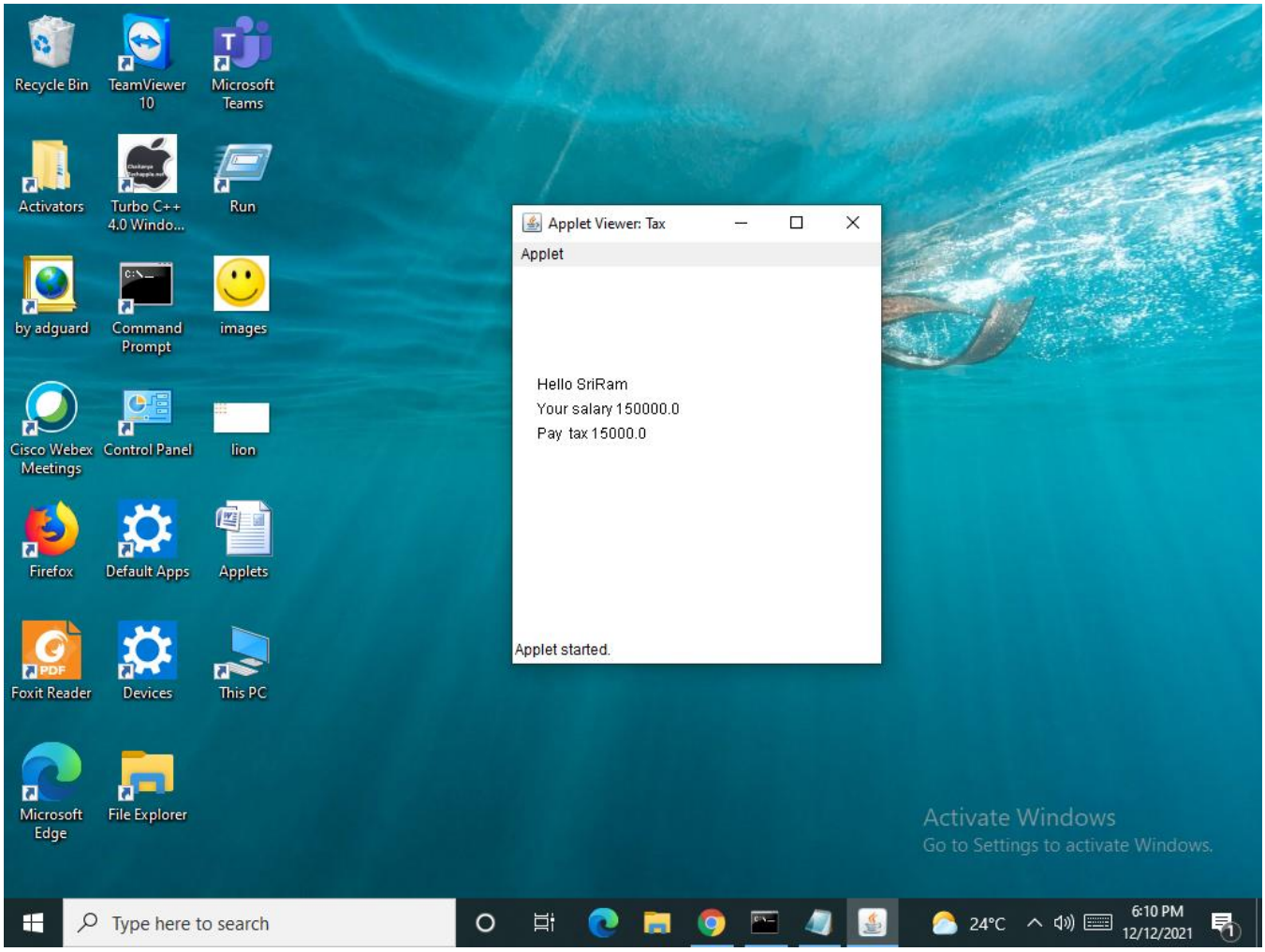
```
/* <applet code=Tax width=300 height=300>
<param name=t1 value=SriRam>
<param name=t2 value=150000>
</applet>
*/
```

Compile:-

```
E:\applets>javac  Tax.java
```

Execution:

```
E:\applets>Appletviewer  Tax.java
```



AWT(Abstract Window Toolkit)

Java AWT (Abstract Window Toolkit) is *an API to develop Graphical User Interface (GUI) or windows-based applications* in Java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

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

The AWT tutorial will help the user to understand Java GUI programming in simple and easy steps.

Why AWT is platform independent?

Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, ChechBox, button, etc.

For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.

In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.

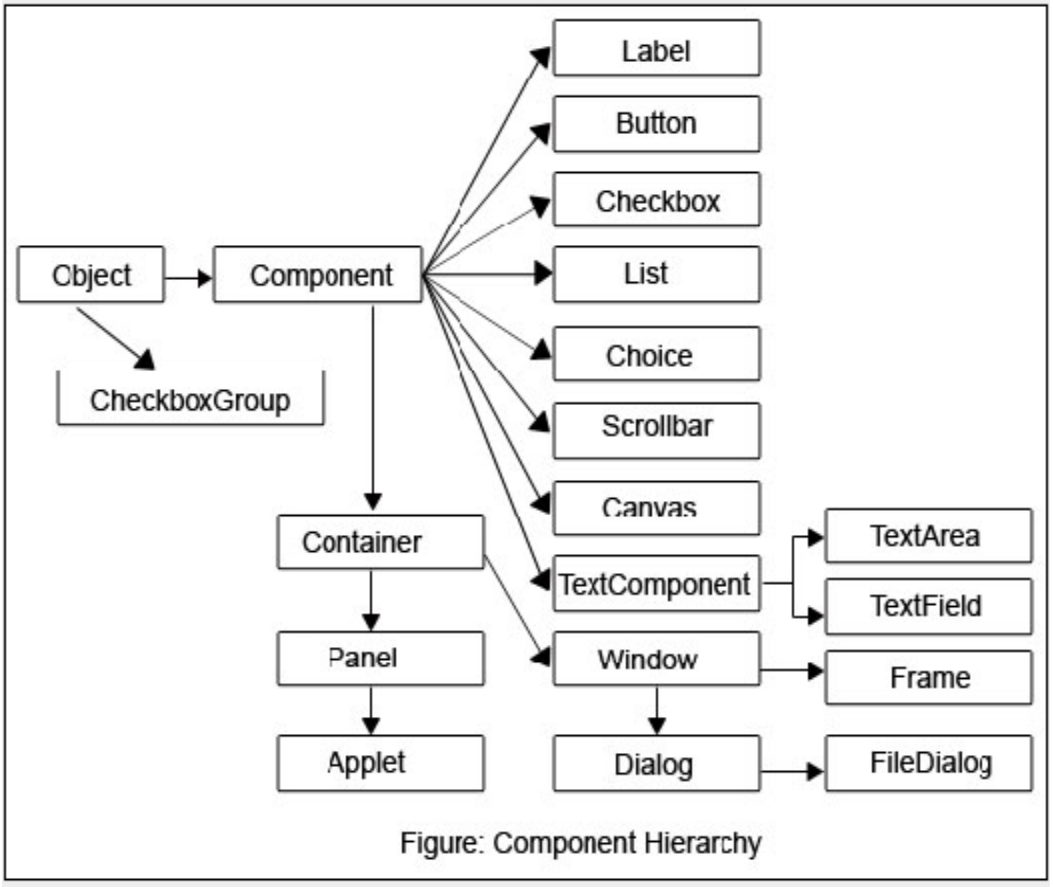


Figure: Component Hierarchy

Components

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

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**.

It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

There are four types of containers in Java AWT:

1. Window
2. Panel
3. Frame
4. Dialog

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. We need to create an instance of Window class to create this container.

Panel

The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

Frame

The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

Some of the classes available in AWT package

a) Frame (Container class)

Constructors

1. **Frame() // creates a Frame without title**
2. **Frame(String) // Creates a frame with title.**

Methods

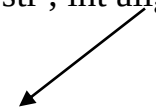
1. setVisible(boolean)// makes the frame to be visible
2. setSize(int,int) // sets the size of the frame
3. setLocation(int,int) // display the frame at the specified position on the screen.
4. setResizable(boolean); // make the frame resizable by passing true value else no resizable

b) Label

Constructors

1. Label(); //default constructor
2. Label(String); //

3. Label(String str , int alignment)



    1. Label.LEFT
    2. Label.Right
    3. Label.CENTER

Methods

1. void setText(String);
2. string getText();
3. void setAlignment(int);
4. int getAlignment();

c) Button

Constructors

1. Button(); //creates Button with no caption on it.
2. Button(String);// creates Button with caption as given string

Methods

1. void setLabel(String);
2. String getLabel();

d) Check boxes

Constructors

1. Checkbox();//creates check box without any caption or Label
2. Checkbox(String);//create checkbox with the caption as string parameter
3. Checkbox(String, boolean); //creates check box with caption as String parameter and Boolean parameter tells whether it is checked or unchecked based on Boolean value
4. Checkbox(String, boolean, CheckboxGroup);//creates check box in mutual exclusive manner

Methods

1. booleangetState();
2. void setState(Boolean);
3. String getLabel();
4. Void setLabel(String)

e) CheckboxGroup

Constructors

1. checkboxGroup();

methods

1. Checkbox getSelectedCheckbox();

2. Void setSelectedCheckbox(Checkbox);

f) List

Constructors

1. List();
2. List(int);
3. List(int,boolean);

Methods

1. void add(String);
2. void add(String, int);
3. String getSelectedItem();
4. intgetSelectedItem();
5. String [] getSelectedItems();
6. Int[] getSelctedIndexes();
7. intgetItemCount();
8. void select(int index);
9. String getItem(int);

g) TextField

Constructors

1. TextField();
2. TextField(int);
3. TextField(String);
4. TextField(String, int);

Methods

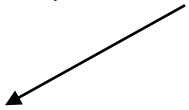
1. String getText();
2. void setText(String);
3. String getSelectedText();
4. void select(intstartindex, intendindex);
5. Boolean isEditable();
6. Void setEditable(bollean);

7. Void setEchoChar(char);
8. Boolean echoCharisSet();
9. Char getEchoChar();

h) TextArea

Constructors

1. TextArea();
2. TextArea(String);
3. TextArea(introws,int cols);
4. TextArea(String,introws,int cols);
5. TextArea(String,introws,intcols,int scrollbars);



   a. SCROLLBARS_BOTH
   b. SCROLLBARS_HORIZONTAL_ONLY
   c. SCROLLBARS_NONE
   d. SCROLLBARS_VERTICAL_ONLY.

Event Delegation Model

When we create a component, generally the component is displayed on the screen but is not capable of performing any actions. For example, we created a button which can be displayed but cannot perform any action, even when someone click on it. But expectation will be different. A user wants the button to perform some action.

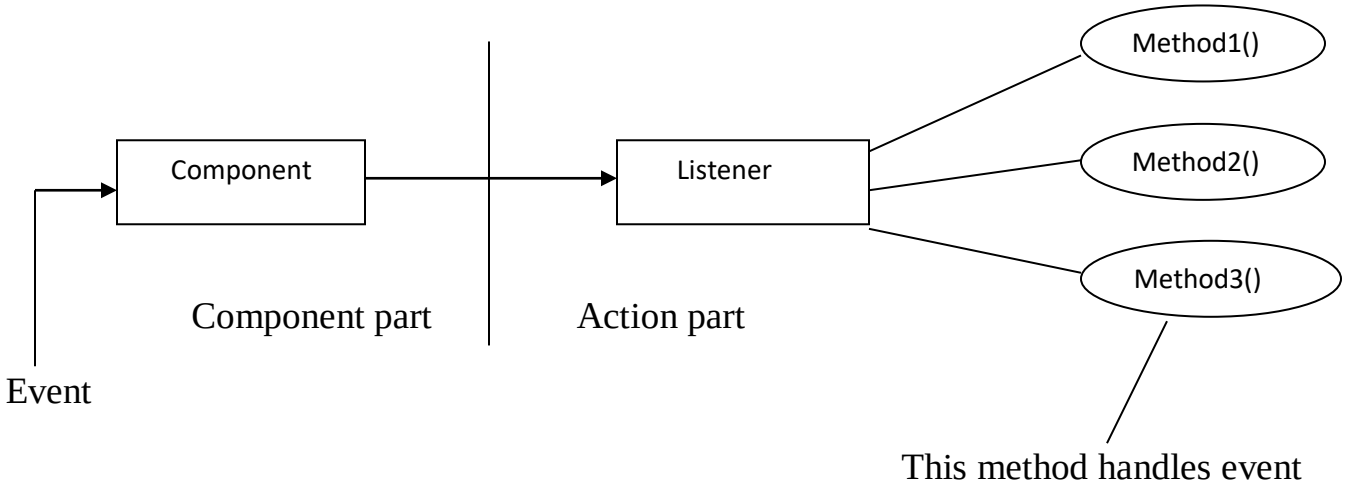
Event:-An event is an object that describes a state change in source.

Clicking like this is called event. An event represents a specific action done on a component. Clicking, double clicking, typing data inside component, mouse over etc are all examples of events.

When an event is generated on the component, the component will not know about it because it cannot listen to the event. To let the component understand that an event is generated on it, we should add some listener to the components. A listener is an interface

which listens to an event coming from a component. A listener will have some abstract methods which need to be implemented by the programmer.

When an event is generated by the user on the component, the event is not handled by the component. On the otherhand, the component sends (delegates) that event to listener that attached to it. The listener will not handle the event. It hands over (delegates) the event to an appropriate method. Finally the method is executed and the event is handled. This is called "**event-delegation model**".



In this model the component is separated from the event is handled.

Advantage

1. The component and the action part can be developed in two separate environments.
2. We can modify the code for creating the component without modifying the code for action part of the component. Similarly , we can modify the action part without modifying the code for the component. Thus, we can modify one part without effecting any modification to other part. This makes debugging and maintenance of code very easy.

Event sources :A source is an object that generates an event.

A source must register listeners in order for the listener to receive notifications about a specific type of event. Each type of event has its own registration method.

General form Public void addTypeListener (TypeListener el).

Sources from which we can generate events are given below

1. **Button:** it generates action event when the user presses a button.
2. **Checkbox :** it generates item event when the checkbox is selected or deselected.
3. **List :** it generates item event when the user selects or deselect an item from Listbox. it also generates action event when an item is double clicked.
4. **MenuItem:** like List, it generates both action and item event. ActionEvent is generates when an menu item is selected and an item event is generated when a checkable menu item is selected/deselected.

Event Listeners

A listener is an object that is notified when an event occurs. It has two major requirements

1. It must have been registered with one or more sources to receive notification about specific types of events.
2. It must implement methods to receive and process these notifications.

Event classes

The classes that represent events are at the core of java's event handling mechanism. At the root of the java event classes hierarchy is **EventObject,** which is in java.util. It is the super class for all events. Its one of the constructor is

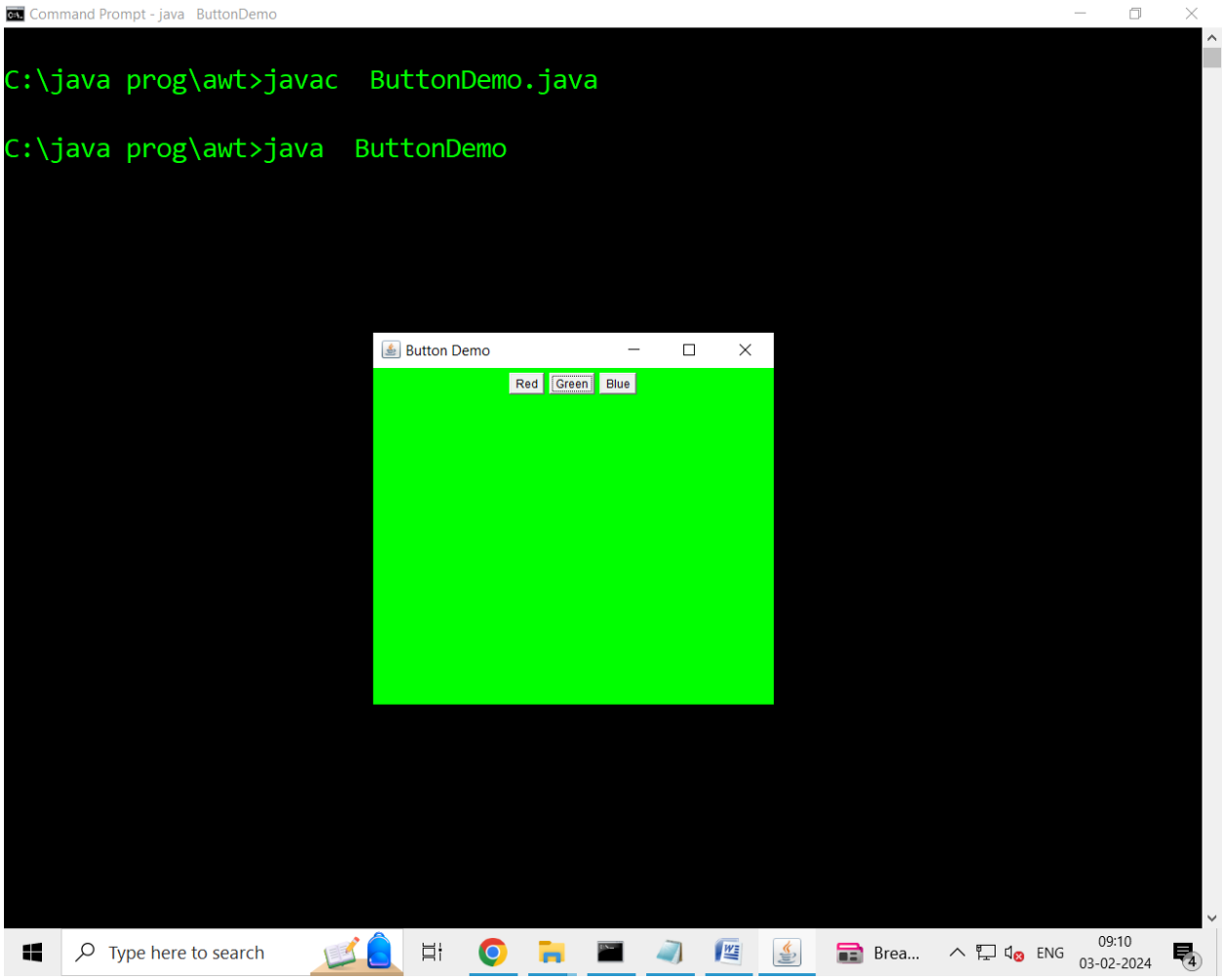
EventObject(Object src) here src is the object that generates this event.

The class **AWTEvent,** defined within the java.awt package is a subclass of **EventObject.** It is super class of all AWT based events used by the delegation event model.

/* Demo program on Action Event*/

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

public class ButtonDemo extends Frame implements ActionListener
{
Button b1,b2,b3;

ButtonDemo()
{
```

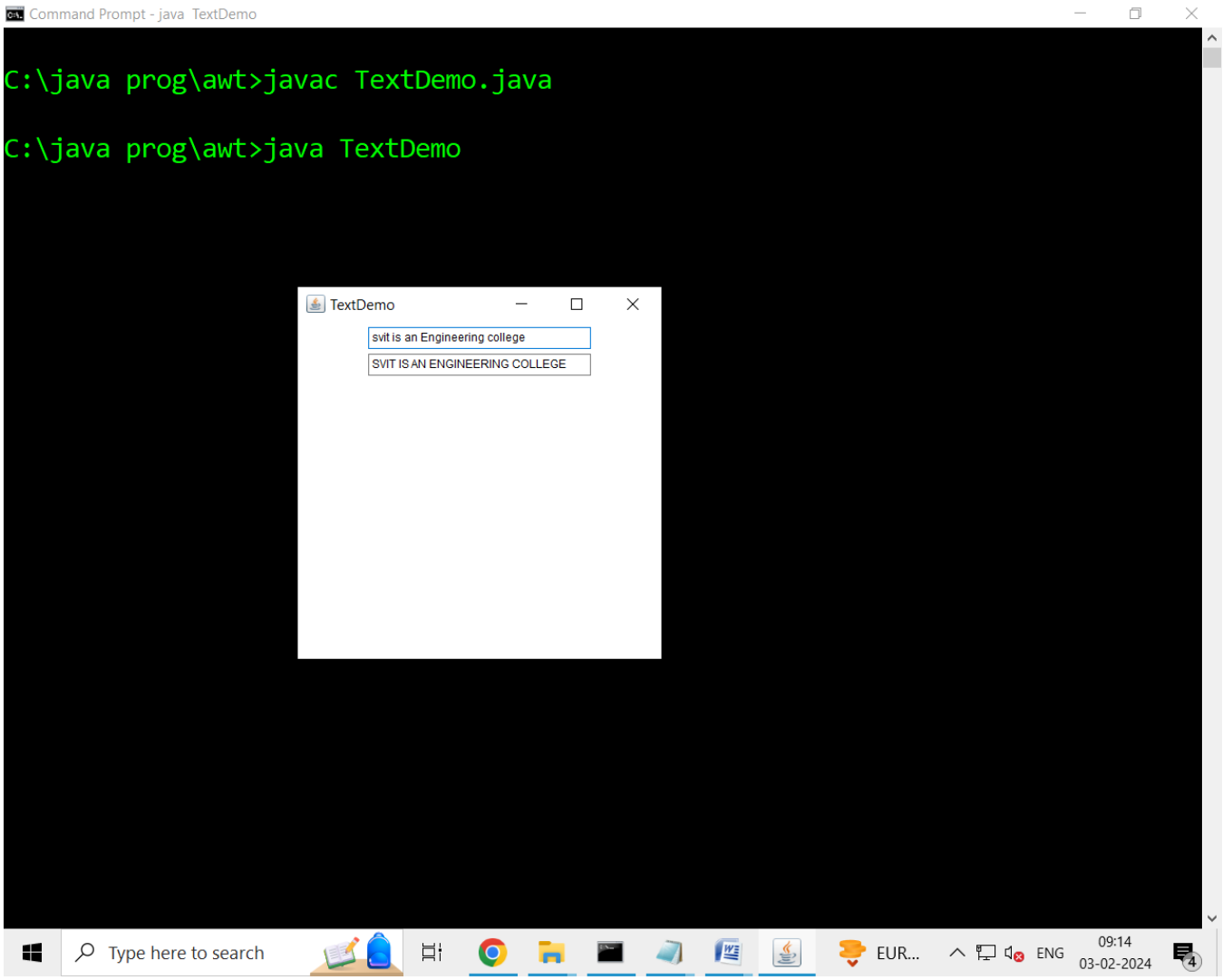
```
super("Button Demo");
setSize(400,400);
setVisible(true);
b1=new  Button("Red");
b2=new  Button("Green");
b3=new  Button("Blue");
setLayout(new  FlowLayout());
add(b1);
add(b2);
add(b3);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
addWindowListener(new WindowAdapter()
{
public  void windowClosing(WindowEvent we)
{
System.exit(0);
}});
}
public  void  actionPerformed(ActionEvent  ae)
{
Color  c=null;
if(ae.getSource()==b1)
setBackground(Color.red);
else if(ae.getSource()==b2)
setBackground(Color.green);
else
setBackground(Color.blue);
}
public  static  void main(String  args[])
{
new ButtonDemo();
}}
```



/* Demo program on TextEvent*/

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

public class TextDemo extends Frame implements TextListener
{
TextField t1,t2;

TextDemo()
{
super("TextDemo");
setVisible(true);
setSize(400,400);
setLocation(300,300);
t1=new TextField(30);
```

```
t2=new  TextField(30);
setLayout(new  FlowLayout());
add(t1);
add(t2);
t1.addTextListener(this);
addWindowListener(new  WindowAdapter()
{
public  void windowClosing(WindowEvent  we)
{
System.exit(0);
}});}
public  void textValueChanged(TextEvent  te)
{
String  str=t1.getText();
t2.setText(str.toUpperCase());
}
public  static  void main(String  args[])
{
new  TextDemo();
}
}
```

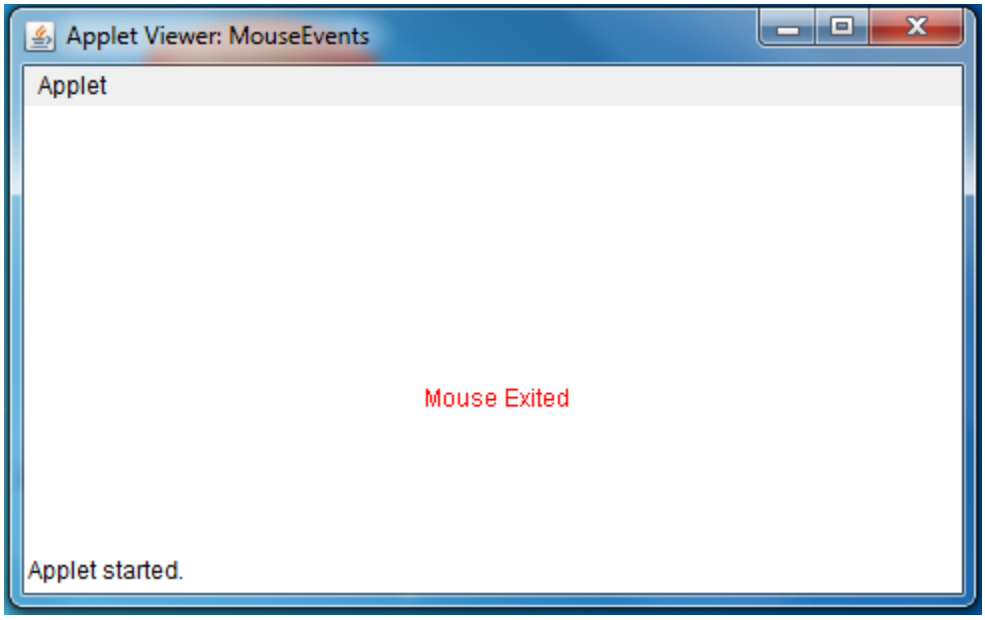


/* Demo program on MouseEvent*/

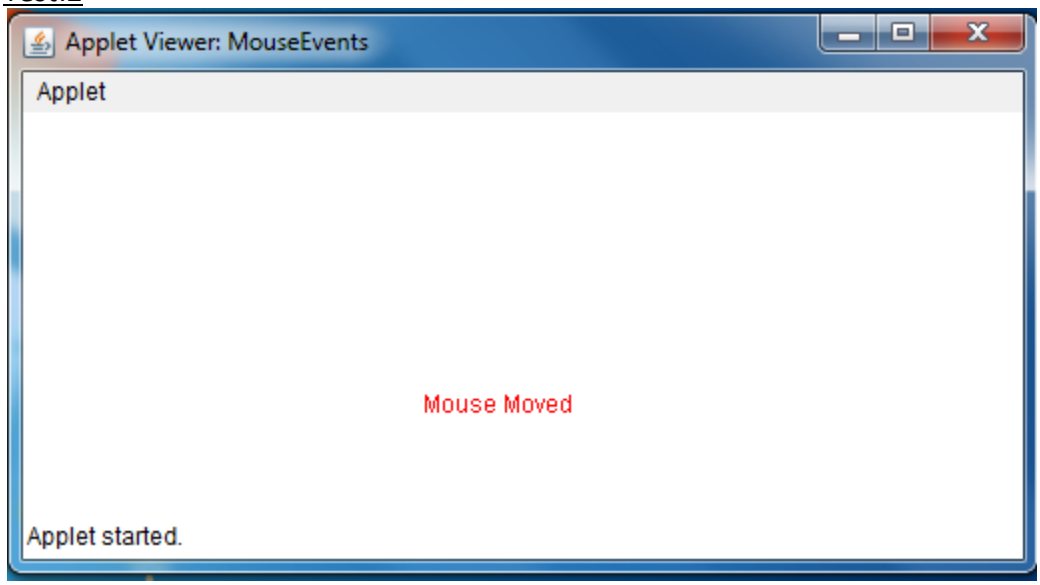
```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/*<applet code="MouseEvents" width=600 height=300></applet>*/

public class MouseEvents extends Applet
{
String str;
public void init()
{
setForeground(Color.RED);
str="Welcome To Mouse Events";
```

```
addMouseListener(new MouseAdapter()
{
public  void mouseEntered(MouseEvent me)
{
str="Mouse  Just Entered applet Window";
repaint();
}
public  void mouseClicked(MouseEvent me)
{
str="Mouse Clicked";
repaint();
}
public  void mouseExited(MouseEvent me)
{
str="Mouse Exited";
repaint();
}
public  void mousePressed(MouseEvent me)
{
str="Mouse Pressed";
repaint();
}
public  void mouseReleased(MouseEvent me)
{
str="Mouse Released";
repaint();
}
});
addMouseMotionListener(new MouseMotionAdapter()
{
public void mouseDragged(MouseEvent me)
{
str="Mouse Dragged";
repaint();
}
public void mouseMoved(MouseEvent me)
{
str="Mouse Moved";
repaint();
}
});
}
```

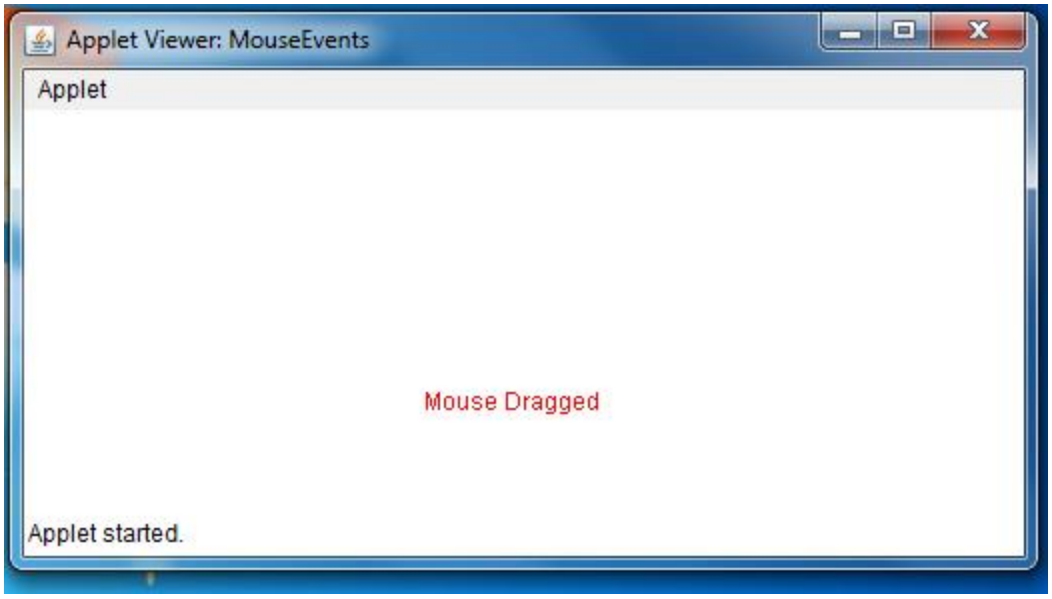
```
public void paint(Graphics  g)
{
g.drawString(str,200,150);
}}
```

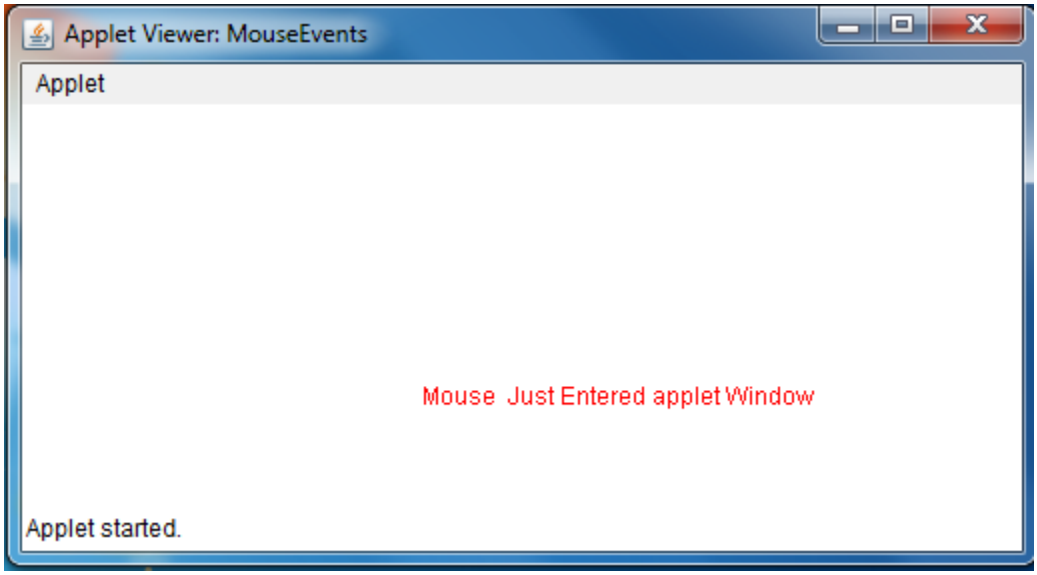


Test:2



Test 3:





/* Demo program on KeyEvent */

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="impkey" width=200 height=300></applet>*/

public class impkey extends Applet implements KeyListener
{
TextField t;
TextArea ta;
public void init()
```

```
{
t=new TextField(20);
ta=new TextArea();
add(t);
add(ta);
t.addKeyListener(this);
}
public void keyPressed(KeyEvent ke)
{
if(ke.getSource()==t)
ta.setText("key pressed");
}

public void keyReleased(KeyEvent ke)
{
if(ke.getSource()==t)
ta.setText("Key Released");
}
public void keyTyped(KeyEvent ke)
{
if(ke.getSource()==t)
{
char c=ke.getKeyChar();
ta.setText("KeyTyped"+c);
}}}
```

Event Adapters

When an event listener interface defines more than one method, it is often accompanied by an eventAdapter class that provides empty implementations for each of the methods. For example, the MouseListener interface defines five different methods. If your program is interested only in the mouseClicked() method, it may be easier for you to subclass the MouseAdapter class and override mouseClicked() than to implement all five methods of the MouseListener interface directly.

/* demo window Adapter*/

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

public  class  windowadapterDemo  extends  Frame
{
public  windowadapterDemo()
```

```
{
super("Window Adapter Demo");
setSize(200,200);
setVisible(true);
addWindowListener(new callme());
}
public static void main(String args[])
{
new windowadapterDemo();
}}
class callme extends WindowAdapter
{
public void windowClosing(WindowEvent we)
{
System.exit(0);
}}
```

List of Adapter classes in java.awt.event Adapter classes

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

Creating Menus in AWT

A top-level window can have a menu bar associated with it. A menu bar displays a list of top-level menu choice is associated with a drop-down menu. This concept is implemented in the AWT by the following classes **MenuBar, Menu** and **MenuItem.** In general, a menu bar contains one or more Menu Objects. Each Menu object contains a list of MenuItem objects. Each MenuItem object represents something that can be selected by the user. It is also possible to include Checkable menuitems. These are menu options of type **CheckboxMenuItem** and will have a check marks next to them when they are selected.

To create a menu bar first create an instance of MenuBar. This class only defines the default constructor. Next, create instances of Menu that will define the selections displayed on the bar.

1. **MenuBar(class)**

   constructor

   MenuBar()

   Methods

   1. Menu add(Menu)

2. **Menu(class)**

   Constructors

   1. Menu()
   2. Menu(String optionName)
   3. Menu(String ,Boolean removable);

3. **MenuItem(class)**

   Constructors

   1. MenuItem()
   2. MenuItem(String itemName)
   3. MenuItem(String, MenuShortcut)

   Methods

   1. void setEnabled(boolean)
   2. boolean isEnabled();
   3. Void setLabel(String)
   4. String getLabel();

5. MenuItem add(MenuItem)

4. CheckboxMenuItem

Creates a checkable menu item by using subclass of MenuItem

Constructors

1. CheckboxMenuItem()
2. CheckboxMenuItem(String itemName)
3. CheckboxMenuItem(String itemName,booelan)

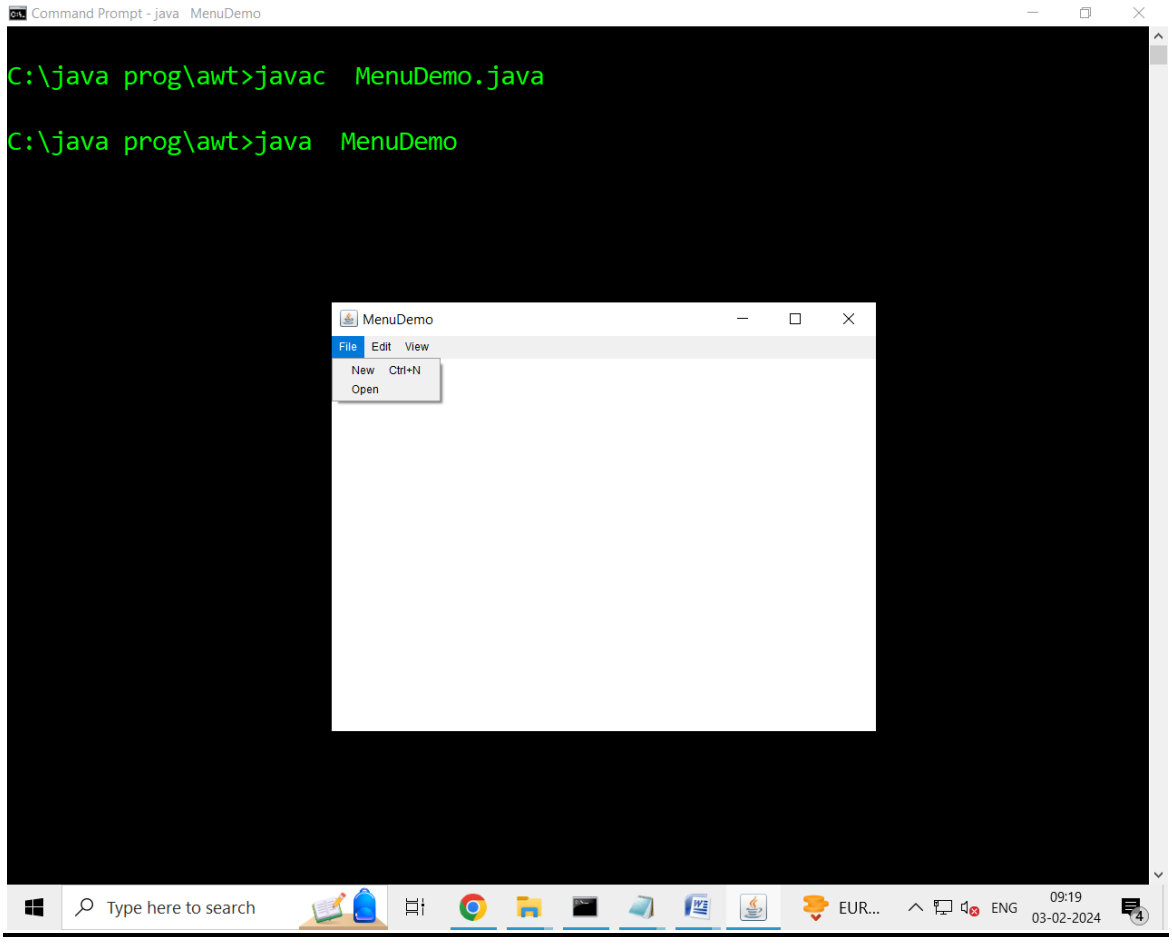
Methods

1. boolean getState();
2. void setState(boolean);

/* Demo program on MenuEvent*/

```
import java.awt.*;
public class MenuDemo extends Frame
{
MenuBar mb;
Menu m1,m2,m3;
MenuItem i1,i2,i3,i4,i5;
MenuDemo()
{
super("MenuDemo");
setSize(300,300);
setVisible(true);
mb=new MenuBar();
setMenuBar(mb);
CheckboxMenuItem mm=new CheckboxMenuItem("display options",true);
MenuShortcut ms=new MenuShortcut('N');
m1=new Menu("File");
m2=new Menu("Edit");
m3=new Menu("View");
i1=new MenuItem("New",ms);
i2=new MenuItem("Open");
i3=new MenuItem("Save");
```

```
i4=new  MenuItem("Cut");
i5=new  MenuItem("Copy");

m1.add(i1);
m1.add(i2);

m2.add(i3);
m2.addSeparator();
m2.add(i4);

m3.add(i5);
m3.add(mm);
mb.add(m1);
mb.add(m2);
mb.add(m3);
}
public static  void  main(String  args[])
{
new  MenuDemo();
}}
```



Anonymous class

An Anonymous class is one that is not assigned a name. we can use anonymous inner class can be used in writing the event handlers.

```
import java.awt.*;
import java.awt.event.*;
class annon extends Frame
{
        annon()
        {
        setSize(300,300);
        setVisible(true);
        addWindowListener(new WindowAdapter()
                                {
                        public void windowClosing(WindowEvent we)
                                {
                                System.exit(0);
                                }
                                });
         }
public static void main(String args[])
        {
        new annon();
        }

}
```

The syntax new MouseAdapter(){..} indicates to compiler that the code between the braces defines an anonymous inner class. Furthermore the class extends MouseAdapter. This new class not named, but it is automatically instantiated when this expression is executed.

Layout Managers

Layout Manager:-A Layout manager is a class that is useful to arrange components in a particular manner in a frame or container.

The following classes represent the layout managers in java.

a. FlowLayout.
b. BorderLayout.
c. CardLayout.

d. GridLayout

FlowLayout

FlowLayout is useful to arrange the components in a line one after the other. When a line is filled with components, they are automatically placed in the next line. This is the default layout in applets and panels.

To create FlowLayout, we can the following ways.

1. FlowLayout f=new FlowLayout();

   This creates FlowLayout. By default, the gap between components will be 5 pixels and the components are centered in the first line.

2. FlowLayout f=new FlowLayout(int alignment);

   Here, the alignment of components can be specified. To arrange the components starting from left to right, we can use ***FlowLayout.LEFT*** . to adjust the components towards right, we can use ***FlowLayout.RIGHT*** and for center alignment, we can use ***FlowLayout.CENTER* .**

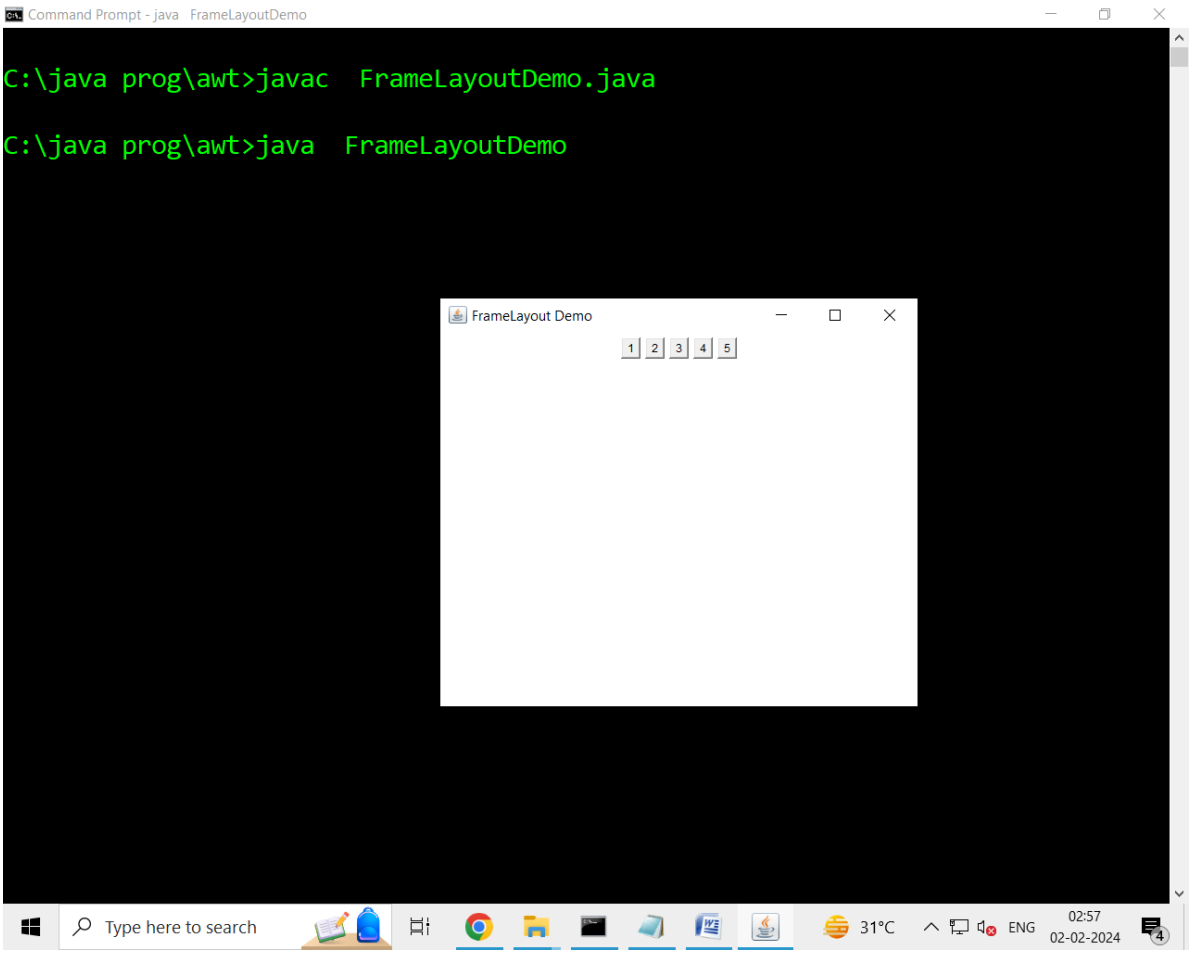
3. FlowLayout f=new FlowLayout(int alignment, int hgap, int vgap);

   Here, the hgap specify the space between components. hgap represents horizontal gap and vgap represents vertical gap in pixels.

/* Demo on FlowLayout*/

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

class FrameLayoutDemo extends Frame
{
Button b1,b2,b3,b4,b5;
FrameLayoutDemo()
{
setTitle("FrameLayout Demo");
setSize(300,300);
setVisible(true);
b1=new Button("1");
b2=new Button("2");
b3=new Button("3");
b4=new Button("4");
```

```
b5=new  Button("5");
FlowLayout f= new FlowLayout();
setLayout(f);
add(b1);
add(b2);
add(b3);
add(b4);
add(b5);
}
public  static  void main(String  args[])
{
new FrameLayoutDemo();
}
}
```



BorderLayout

BorderLayout is useful to arrange the components in the 4 borders of the frame as well as in the center. The border are identical with the names of directions. The top border is

specified as ‘North’, the right side border ‘East’, the bottom one as ‘South’, and the left one as ‘West’. The center is represented as ’Center’.

To create the BorderLayout we can use the following ways.

1. BorderLayout b=new BorderLayout();

   This creates a BorderLayout without ant gaps between the components.

2. BorderLayout b=new BorderLayout(int hgap, int vgap);

   Here, the hgap specify the space between components. hgap represents horizontal gap and vgap represents vertical gap in pixels.

While adding the components to the container the direction should be specified as

C.add(“North”,component); //C is container.

Here the component is added in the container in North direction.

We can also add the componenet in North direction as shown here

C.add(component, BorderLayout.NORTH);

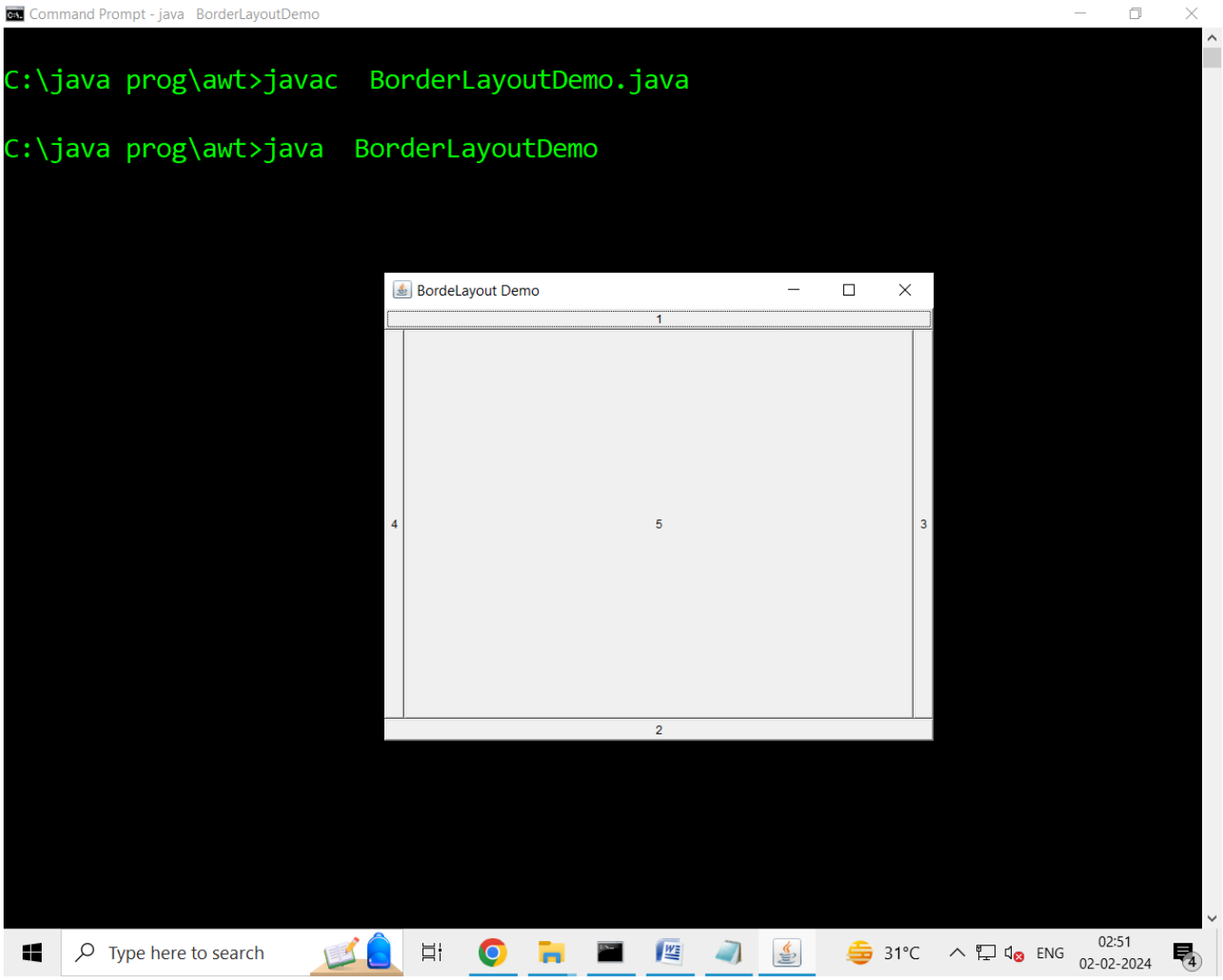
	NORTH	
WEST	**CENTER**	**EAST**
	SOUTH	

Figure 1: Direction of components in BorderLayout.

/* Demo program on BorderLayout*/

```
import java.awt.*;
import java.awt.event.*;
class BorderLayoutDemo extends Frame
{
```

```
Button  b1,b2,b3,b4,b5;
BorderLayoutDemo()
{
setTitle("BordeLayout Demo");
setSize(300,300);
setVisible(true);
b1=new  Button("1");
b2=new  Button("2");
b3=new  Button("3");
b4=new  Button("4");
b5=new  Button("5");
//FlowLayout f= new FlowLayout();
//setLayout(f);
add(b1, BorderLayout.NORTH);
add(b2, BorderLayout.SOUTH);
add(b3, BorderLayout.EAST);
add(b4, BorderLayout.WEST);
add(b5, BorderLayout.CENTER);

}
public  static  void main(String  args[])
{
new BorderLayoutDemo();
}
}
```



CardLayout

A CardLayout object is a layout manager which treats each component as a card. Only one card is visible at a time, and the container acts as a stack of cards. The first component added to a CardLayout object is the visible component when the container is first displayed.

To create CardLayout object, we can use the following ways

1. CardLayout c=new CardLayout();

Here, the CardLayout object is created without any gaps between the components.

2. CardLayout c=new CardLayout(int hgap, int vgap);

The proceeding statement creates a card layout with the specified horizontal and vertical gaps between the components.

While adding the components to the container, we can use add() method as:

Eg: C.add("card_name", component);

To retrieve the cards one by one , the following methods can be used.

a. void first(container); to retrieve the first card.
b. void last(container); to retrieve the last card.
c. void next(container); to retrieve the next card.
d. void previous(container); to retrieve the previous card.
e. void show(container,"card_name"); to see a particular card with name specified.

GridLayout

GridLayout is useful to divide the container into a two-dimensional grid form that contains several rows and columns. The container is divided into equal-sized rectangles, and one component is placed in each rectangle.

To create GridLayout

Object we can write as:

1. **GridLayout g=new GridLayout();**

This creates a Grid Layout with a default of one column per component, in a single row.

2. **GridLayout g=new new GridLayout (int rows, int cols);**

This creates a Grid Layout with specified number of rows and columns.

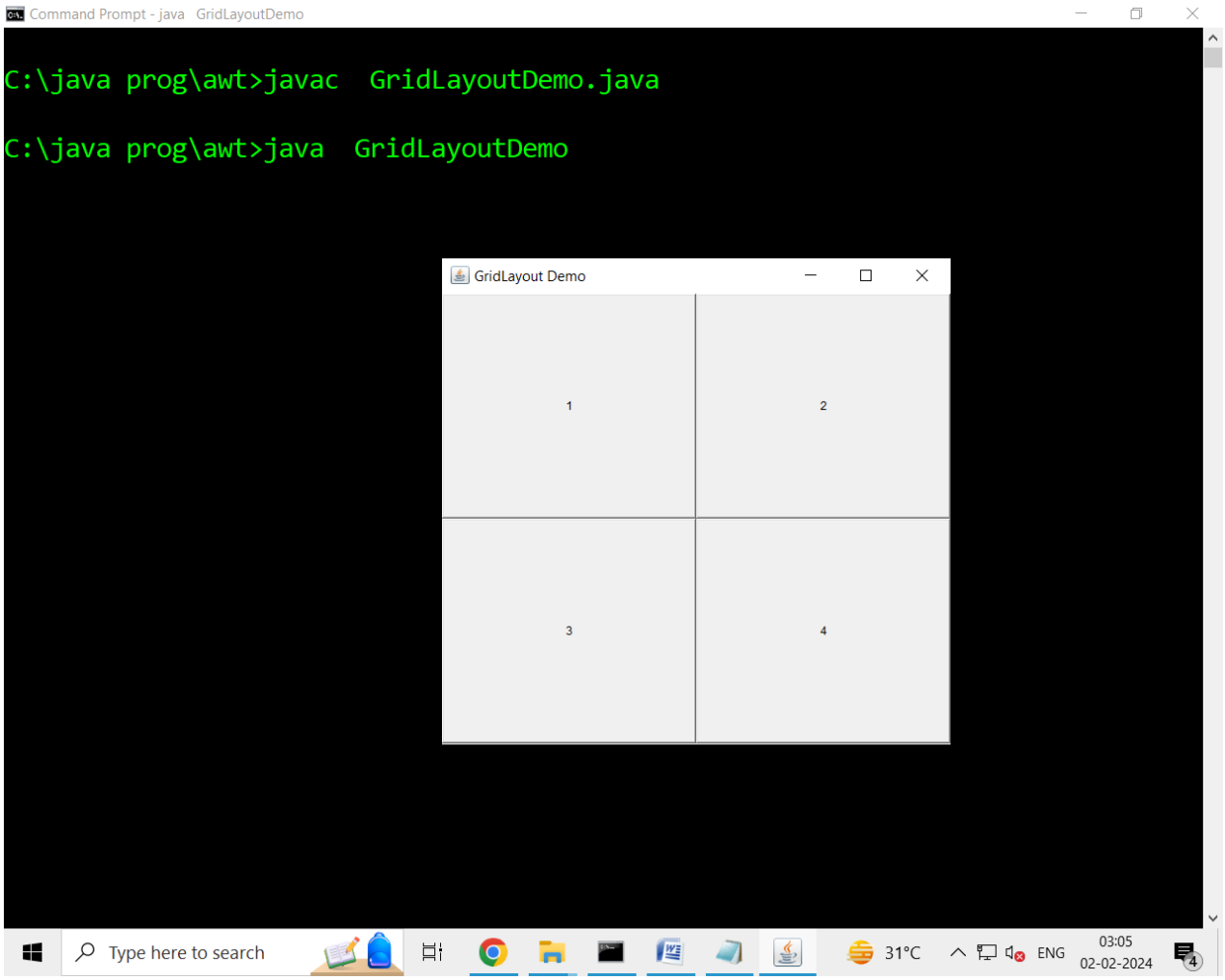
3. **GridLayout g=new GridLayout (int rows, int cols, int hgap, int vgap);**

Here, the hgap represents horizontal gap between component and vgap represents vertical gap between components.

/* Demo on GridLayout*/

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

```
class  GridLayoutDemo  extends Frame
{
Button b1,b2,b3,b4;
GridLayoutDemo()
{
setSize(300,300);
setVisible(true);
setTitle("GridLayout Demo");
GridLayout  g1=new  GridLayout(2,2);
setLayout(g1);
b1=new  Button("1");
b2=new  Button("2");
b3=new  Button("3");
b4=new  Button("4");
add(b1);
add(b2);
add(b3);
add(b4);
}
public  static  void  main(String  args[])
{
new  GridLayoutDemo();
}
}
```



Introduction to servlets

Web Based Programming can be classified into two categories

1. Client side programming.
2. Sever-side programming.

Client-side programming:-in client side programming paradigm, programs scripts are downloaded, interpreted, and executes by the browser. The author of the program does not have any idea about type and version of the browse used to execute them.

Applets are client-side programming technology, where special java programs are embedded into the web pages with help of **<applet>** tag. When a browser loads such a web page, the applet byte code is downloaded and executed on the client side. If the browser uses an old **J**ava **R**untime **E**nvironment (JRE), applet byte code cannot be executed properly,

Hence this technology tends to be not acceptable. These issues have enforced business to use Sever Side Programming.

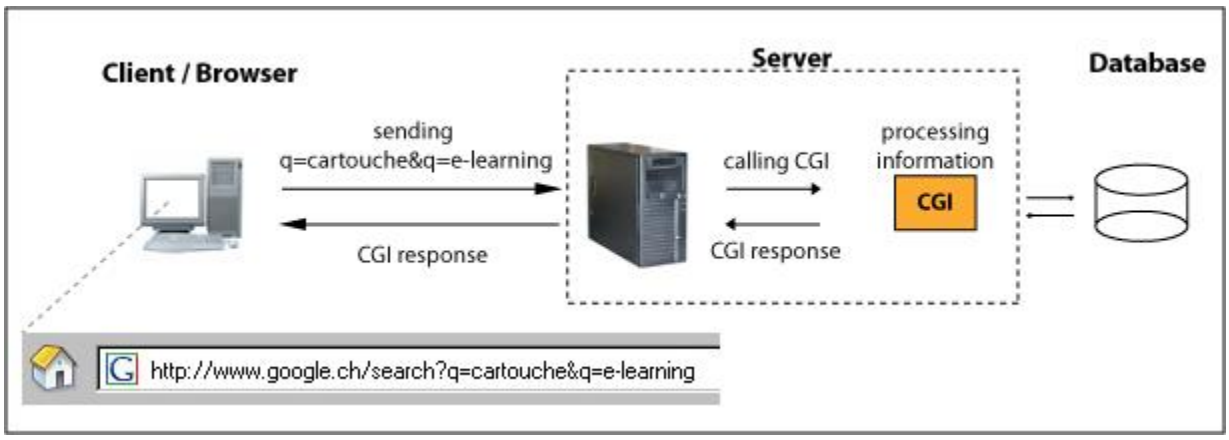
Server-side programming:- Server-side programming solves the above said problems. The basic idea of this paradigm is that programs are executed in the web server. Hence there is no issue of browser incompatibility or long downloaded time. The web server sends web pages that even old browser can understand. Example of server-side programming languages are **CGI, Servlets, JSPs, PHP, PERL(Practical Extraction Reporting Language).**

What is CGI?

CGI (Common Gateway Interface) is server side technology, it is designed on the basis of C language and scripting languages. Basically 'C' programming Language. is process based programming language. So CGI is also Process based Programming language.

Working of CGI application

When a CGI enables web server receives a request for a CGI program, it does not send a file as it is. Instead the web server executes the program at the server end and sends the output back to the client, which is then displayed in the browser's window. Moreover, the CGI specification is independent of any programming language. It defines how information is transferred from a web server to an external application and from external application to web server, so these external application may be developed using a programming language that fits the application.



Disadvantages of CGI

There are many problems in CGI technology:

1. If we deploy CGI application at server then container will create a separate process for each and every request coming from client.
2. Basically, process is heavy weight, it will take more memory and more execution time, it will reduce application performance

3. It uses platform dependent language e.g. C, C++, perl.

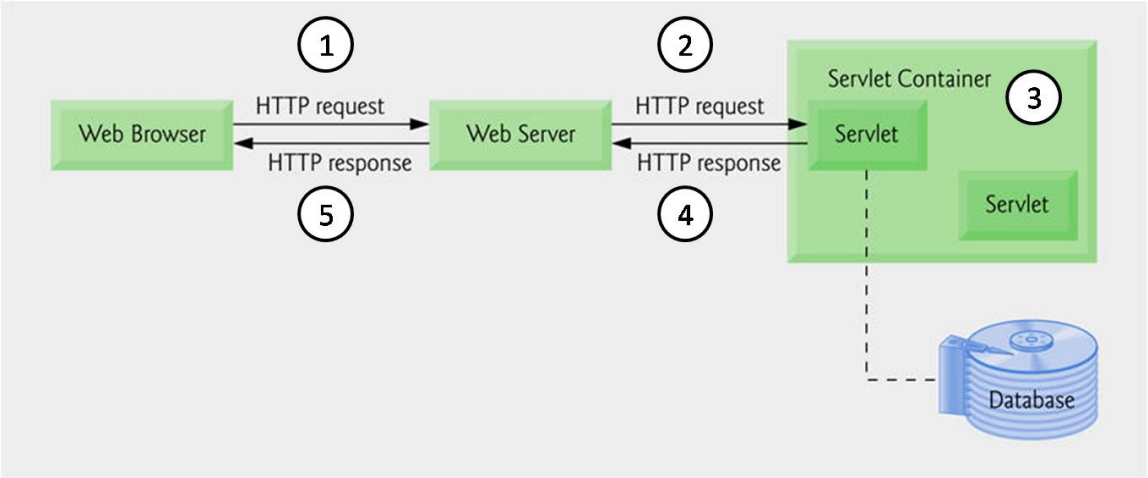
Servlets

Servlets is a java technology for server-side programming. It is java module that runs inside a java-enabled web server and services requests obtained from the web server. Servlets are not tied to a specific client-server protocol, but are most commonly used with HTTP.

(Or)

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

Execution of a Servlets consists of five steps



1. The client sends a request to the web server.
2. The web server interprets it and forwards it to the corresponding Servlets.
3. The Servlets process the request, generates the output .
4. The generated output is sends it back to the web server.
5. The web sever sends the response back to the client. The browser then displays it on the screen.

It runs entirely within Java Virtual Machine(JVM). Since it runs on the server and generates simple output, which even old versions of browsers can interpret, there is no incompatibility.

Difference between CGI and Servlets

CGI	Servlet
It is a process based. i.e., for every request, a new process will be created and that process is responsible to generate required response.	It is thread based. i.e., for every request new thread will be created and that thread is responsible to generate required response.
Creation and destruction of new process for every request is costly, if the number of requests increases then the performance of the system goes down. Hence CGI technology fails to deliver scalable applications.	Creation and destruction of a new thread for every request is not costly, hence if the number of requests increases there is no change in response time. Due to this, servlet technology succeeds to deliver scalable applications.
Two processes never share common address space. Hence there is no chance of occurring concurrence problems in CGI	All the threads shares the same address space, Hence concurrence problem is very common in servlets.
We can write CGI program in variety of languages, But most popular language is perl.	We can write servlets only in java.
Most of the CGI languages are not object oriented. Hence we miss the benefits of oops.	Java language itself is object oriented. Hence we can get all the key benefits of oops.
Most of CGI languages are platform dependent.	Java language is platform independent.

Servlet API

1. javax.servlet.*;
2. javax.servlet.http.*;

a) javax.servlet.* package

it consists of 6 interfaces and 5 classes
List of interfaces

1. Servlet

1.void init(ServletConfig)
2.void service(SrervletRequest,ServletResponse);
3.void destroy()
4. ServletConfig getServletConfig();
5.String getServletInfo();

2. **ServletRequest**

1. String getParameter(String);
2. Enumeration getParameterNames();
3. String[] getparameterValues();
4. String getContentType();
5. int getContentLength();
6. String getProtocol();
7. String getServerPort();
8. ServletInputStream getInputStream();

3. **ServletResponse**

1.PrintWriter getWriter();
2. ServletOutputStream getOutputStream();
3. void setContentType(String);
4. void setContentLength(int);

4. **ServletContext**

1. void setAttribute(String,Object);
2. Object getAttribute(String);
3. Enumeration getAttributeNames();
4. String getInitParameter(String);
5. Enumeration getInitParameterNames();
6. String getServerinfo();

5. **ServletConfig(4 abstract methods)**

1. String getInitParameter(String);
2. Enumeration getInitParameterNames();
3. ServletContext getServletContext();
4. String gerServletName();

6. **SingleThreadModel(Marker interface)**

classes in javax.servlet package

1. Genericservlet

1. void destroy();
2.String getInitParameter(String);
3. Enumeration getInitParameterNames();
4. ServletConfig getServletContext();
5. String getServletInfo();
6. String getServletName();
7. void init();

A convenience method which can be overridden so that there's no need to call super.init(config).

ServletRequest Interface

An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc.

Methods of ServletRequest interface

There are many methods defined in the ServletRequest interface. Some of them are as follows:

Method	Description
public String getParameter(String name)	is used to obtain the value of a parameter by name.
public String[] getParameterValues(String name)	returns an array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box.
java.util.Enumeration getParameterNames()	returns an enumeration of all of the request parameter names.
public int getContentLength()	Returns the size of the request entity data, or -1 if not known.

public String getContentType()	Returns the Internet Media Type of the request entity data, or null if not known.
public abstract String getServerName()	Returns the host name of the server that received the request.
public int getServerPort()	Returns the port number on which this request was received.

Servlet Interface

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

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

Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

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

public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

ServletConfig Interface

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

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

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **public ServletContext getServletContext():**Returns an object of ServletContext.

ServletResponse interface

Defines an object to assist a servlet in sending a response to the client. The servlet container creates a ServletResponse object and passes it as an argument to the servlet's service method.

Method Summary	
java.io.PrintWriter	**getWriter**() Returns a PrintWriter object that can send character text to the client.
void	**setContentLength**(int len) Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header.
void	**setContentType**(java.lang.String type)

	Sets the content type of the response being sent to the client.
void	**setLocale**(java.util.Locale loc) Sets the locale of the response, setting the headers (including the Content-Type's charset) as appropriate.

GenericServlet class

GenericServlet class implements **Servlet**, **ServletConfig** and **Serializable** interfaces.It provides the implementation of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet class

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.
5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
6. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. **public ServletContext getServletContext()** returns the object of ServletContext.
8. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
9. **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
10. **public String getServletName()** returns the name of the servlet object.

11. **public void log(String msg)** writes the given message in the servlet log file.
12. **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

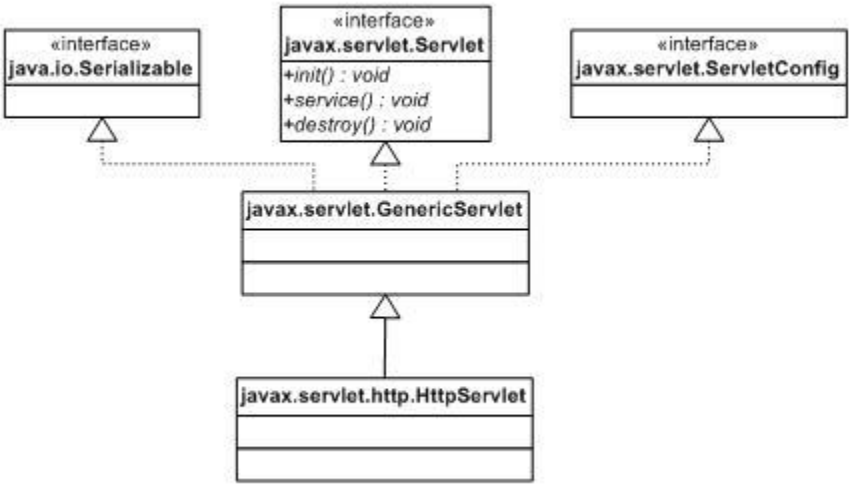
b) javax.servlet.HttpServlet.*;

The javax.servlet.http package contains a number of classes and interfaces that 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.

Interface Summary	
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.

Class Summary	
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.

HTTPServlet class



Public abstract class HttpServlet extends GenericServlet implements Serializable

The **HttpServlet** class extends **GenericServlet** and provides an HTTP-specific implementation of the Servlet interface. This class specific methods are

1. **protected void doGet(HttpServletRequest req, HttpServletResponse resp)**
2. **protected void doHead(HttpServletRequest req, HttpServletResponse resp)**
3. **protected void doPost(HttpServletRequest req, HttpServletResponse resp)**
4. **protected void doPut(HttpServletRequest req, HttpServletResponse resp)**
5. **protected void doDelete(HttpServletRequest req, HttpServletResponse resp)**
6. **protected void doTrace(HttpServletRequest req, HttpServletResponse resp)**
7. **protected void doOptions(HttpServletRequest req, HttpServletResponse resp)**
8. **protected void service(HttpServletRequest req, HttpServletResponse resp)**
9. **public void service(ServletRequest req, ServletResponse res)**

Service () methods

The HTTPServlet has two variants of this method

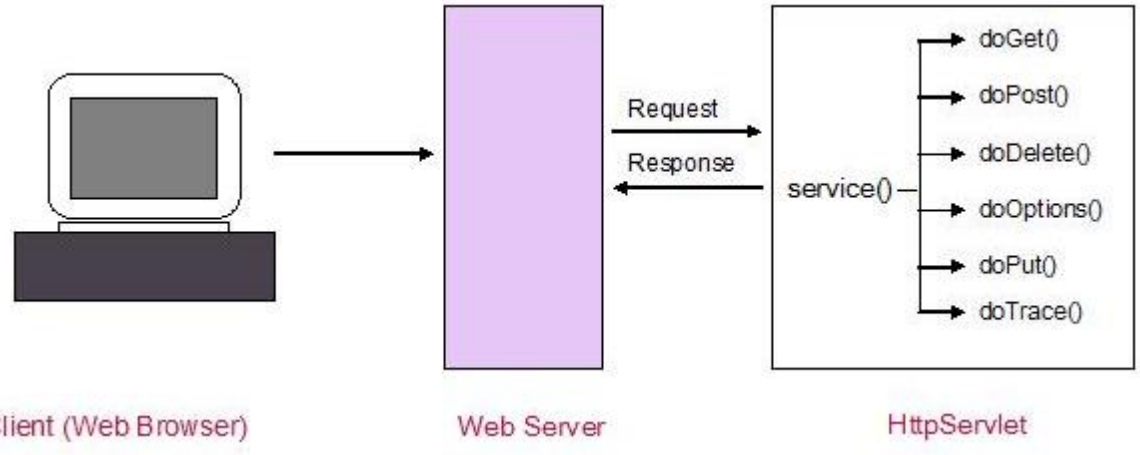
1. **public void service(ServletRequest request, ServletResponse response)** throws ServletException, IOException

   This is an implementation of the service() method in the GenericServlet. This method casts request and response objects to HttpServletRequest and HttpServletResponse, and calls the following overloaded service() method. Therefore we should not override the above method

2. **protected void service(HttpServletRequest req, HttpServletResponse resp)** throws ServletException,IOException

This overloaded method takes HTTP-specific request and response objects and is invoked by the first method above. HttpServlet implements this method to be a dispatcher of HTTP requests.

When this ***service()*** method is invoked, it reads the method type stored in the ***request*** and determines which method to invoke based upon this value. These are the methods that we will want to override. If the method type is **GET**, it will call ***doGet()***. If the method is **POST**, it will call ***doPost()***. There are five other method types. All these methods have the same parameter list as the **service()** method.



HttpServletRequest interface

public interface **HttpServletRequest** extends ServletRequest

HttpServletRequest is **an interface and extends the ServletRequest interface**. By extending the ServletRequest this interface is able to allow request information for HTTP Servlets. Object of the HttpServletRequest is created by the Servlet container and, then, it is passed to the service method (doGet(), doPost(), etc.)

Method Summary

java.lang.String	getContextPath() Returns the portion of the request URI that indicates the context of the request.
Cookie[]	getCookies() Returns an array containing all of the Cookie objects the client sent with this request.
java.lang.String	getQueryString() Returns the query string that is contained in the request URL after the path.
java.lang.String	getRequestedSessionId() Returns the session ID specified by the client.
java.lang.StringBuffer	getRequestURL() Reconstructs the URL the client used to make the request.
java.lang.String	getServletPath() Returns the part of this request's URL that calls the servlet.
HttpSession	getSession() Returns the current session associated with this request, or if the request does not have a session, creates one.
HttpSession	getSession(boolean create) Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.
boolean	isRequestedSessionIdFromCookie() Checks whether the requested session ID came in as a cookie.

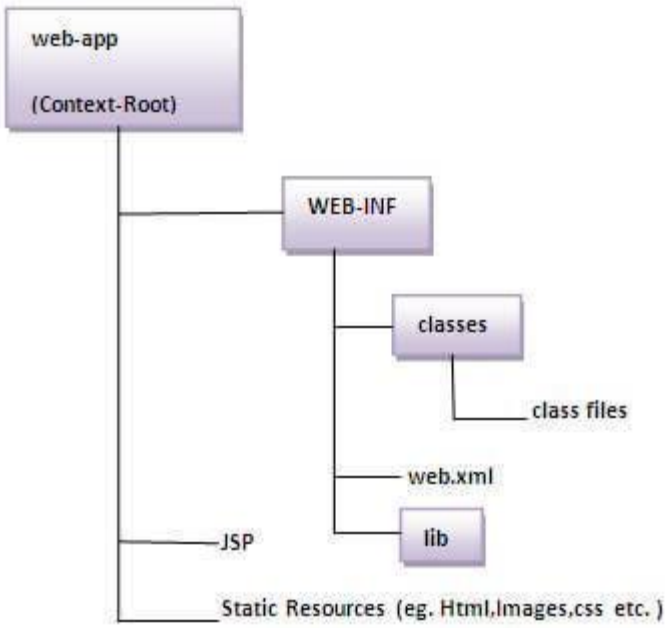
HttpServletResponse

public abstract interface **HttpServletResponse** extends ServletResponse

HttpServletResponse is **a predefined interface present in javax. servlet. http package**. It can be said that it is a mirror image of request object. The response object is where the servlet can write information about the data it will send back.

Method Summary	
void	addCookie(Cookie cookie) Adds the specified cookie to the response.
void	sendError(int sc) Sends an error response to the client using the specified status code and clearing the buffer.
void	sendError(int sc, java.lang.String msg) Sends an error response to the client using the specified status clearing the buffer.
void	sendRedirect(java.lang.String location) Sends a temporary redirect response to the client using the specified redirect location URL.
void	setStatus(int sc) Sets the status code for this response.
void	setStatus(int sc, java.lang.String sm) **Deprecated.** *As of version 2.1, due to ambiguous meaning of the message parameter. To set a status code use setStatus(int), to send an error with a description use sendError(int, String). Sets the status code and message for this response.*

/*Directory Structure for Web Application*/



/* program to demonstrate HttpServlet to print Welcome to Servlets*/

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet{
public void service(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");//setting the content type
PrintWriter pw=res.getWriter();//get the stream to write the data

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

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

Web.xml

```
<web-app>
<servlet>
<servlet-name>hello</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>hello</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
```

Get vs. Post

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

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

GenericServlet vs HttpServlet

GenericServlet allows all types of protocols and has only one abstract method which is service() method and other four methods are concrete method whereas HttpServlet is a class which is specific and allows only http protocol and other protocol like FTP doesn't allow SMTP.

#	GenericServlet	HttpServlet
1.	GenericServlet defines a generic and protocol-independent servlet.	But HttpServlet defines a HTTP protocol specific servlet.
2.	In GenericServlet, the **service()** method is abstract.	But **service()** method in HttpServlet is non-abstract.
3.	GenericServlet extends Object class and implements Servlet, ServletConfig, and Serializable interfaces.	But HttpServlet extends GenericServlet and implements a Serializable interface.
4.	GenericServlet supports only service() method which does not contain doGet() and doPost() methods.	HttpServlet also supports doGet(), doPost(), doHead(), doPut(), doOptions(), doDelete(), doTrace() methods.
5.	GenericServlet can process multiple clients request from a single form hence programmer doesn't prefer this servlet.	HttpServlet can process multiple clients requesting from multiple HTML forms hence programmer prefer this servlet.
6.	GenericServlet is Stateless and is not commonly used servlet.	But HttpServlet is Stateful and it is most popular servlet because it is commonly used by the developer.

Servlet Life Cycle

The Servlet container of web Server supervises and controls the life cycle of a servlet. When a container receives a request from a client and determines that the request should be handled by a servlet, it performs the following steps.

1. If an instance of target servlet does not exist, the container does the following
    a. Finds and loads the servlet class.
    b. Creates an instance of the servlet class.
    c. Call the **init()** method on this servlet instance to initialize it.
2. Otherwise (i.e if the target servlet exists). It invokes the **service()** method on it, passing a **ServletRequest** type object and a **ServletResponse** type Object.
3. If the container decides that the servlet is no longer needed. It removes and finalizes the servlet by calling the servlet's destroy() method.

1. init()

A servlet's life begins here. This method is called only once by the web container. Just after it instantiates and loads the servlet. So, it is a good idea to read the persistent configurations data. Initialize resources that will be used during the rest of the time, and perform any other one-time activity by overriding the init() method. Once the servlet is initialized, it becomes ready to handle the client's request. The prototype of the init() method is as follows

public void init(ServletConfig)

The web container passes a ServletConfig object, which contains the startup configuration fir his servlet such as initialization parameters.

2. Service()

This is the most important method, whose signature is as follows

void service(ServletRequest request, ServletResponse response)

This method gets called each time the web container receives a request intended for this servlet. The web container calls the **service()** method on a servlet with two objects: a **ServletRequest** type object request and a **ServletResponse** type object response. The Request object encapsulates the communication from the client to the server. While the response object encapsulates the communication from the servlet back to the client.

The **ServletRequest** interface provides methods to retrieve information sent by the class. Similarly, the **ServletResponse** interface provides methods to send data to the clients.

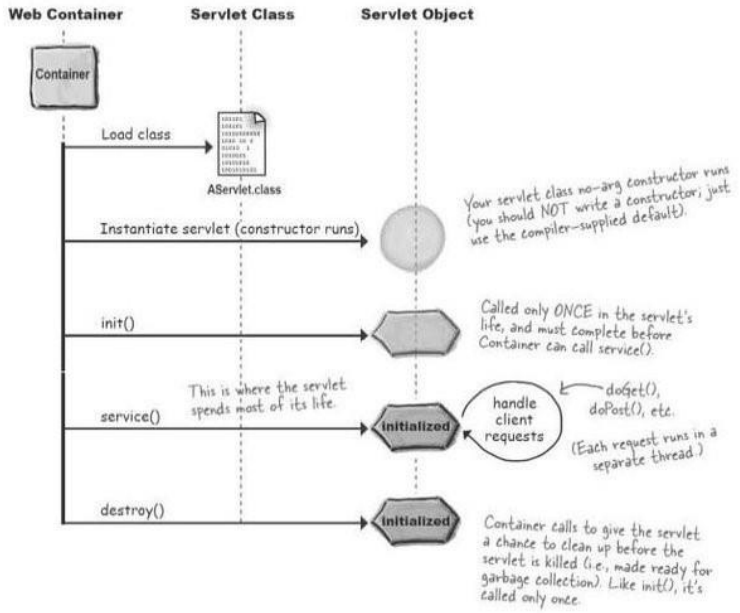
3. destroy()

The method signature is as follows

void destroy()

This method gets called when the web container uninstalls servlet. This method is overridden to clean up the resources that were allocated to this servlet. It also makes sure that any persistent state is synchronized with the servlet's current in-memory state.

Life cycle of Servlet



Session Tracking in Servlets

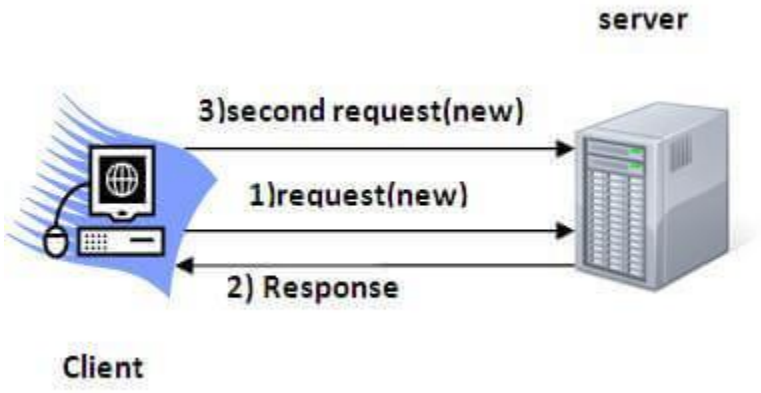
Session Tracking is a mechanism that servlets use to maintain state (data) about a series of requests from same user(that is request originating from the same browser) across some period of time.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless[1] that means each request is considered as the new request. It is shown in the figure given below:

[1] HTTP is stateless protocol. A client open connection and requests some resource or information. The server responds with the requested resource(if available) or sends an HTTP error status. After closing the connections, the server does not remember any information about the client. So, the server consider the next request from the same client as a fresh request, with no relation to the previous request. This is what makes HTTP a stateless protocol.

A protocol is stateful if the response to a given request may depend not only on the current. But also on the outcome of previous requests.



Session Tracking Techniques

There are four techniques used in Session tracking:

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

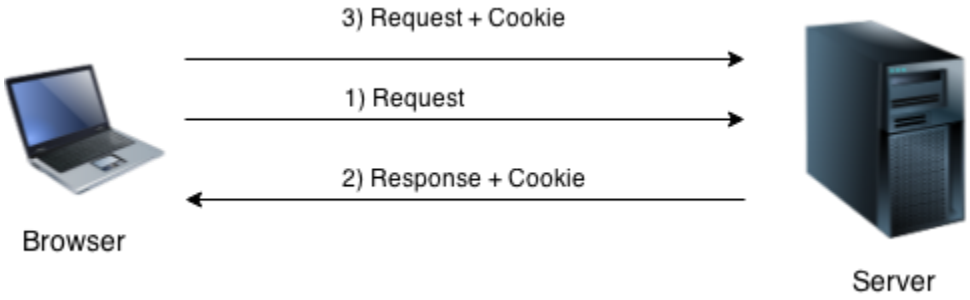
Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user. A cookie contains one or more name-value pairs with certain additional attributes, which are exchanges in the response and request headers



Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

Method	Description

public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.

How to create Cookie?

Let's see the simple code to create cookie.

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

How to delete Cookie?

Let's see the simple code to delete cookie.

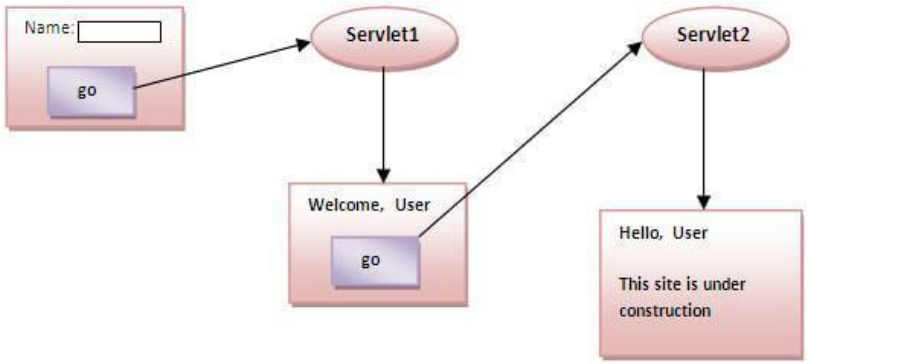
```
Cookie ck=new Cookie("user","");//deleting value of cookie
ck.setMaxAge(0);//changing the maximum age to 0 seconds
response.addCookie(ck);//adding cookie in the response
```

How to get Cookies?

Let's see the simple code to get all the cookies.

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
 out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of coo
kie
}
```

Example



index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

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


public class FirstServlet extends HttpServlet {

  public void doPost(HttpServletRequest request, HttpServletResponse response){
    try{

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");
    out.print("Welcome "+n);

    Cookie ck=new Cookie("uname",n);//creating cookie object
    response.addCookie(ck);//adding cookie in the response

    //creating submit button
    out.print("<form action='servlet2'>");
    out.print("<input type='submit' value='go'>");
    out.print("</form>");

    out.close();

        }catch(Exception e){System.out.println(e);}
  }
}
```

SecondServlet.java

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

public class SecondServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response){
   try{
```

```
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        Cookie ck[]=request.getCookies();
        out.print("Hello "+ck[0].getValue());

        out.close();

            }catch(Exception e){System.out.println(e);}
        }
    }
```

web.xml

```
    <web-app>

    <servlet>
    <servlet-name>s1</servlet-name>
    <servlet-class>FirstServlet</servlet-class>
    </servlet>

    <servlet-mapping>
    <servlet-name>s1</servlet-name>
    <url-pattern>/servlet1</url-pattern>
    </servlet-mapping>

    <servlet>
    <servlet-name>s2</servlet-name>
    <servlet-class>SecondServlet</servlet-class>
    </servlet>

    <servlet-mapping>
    <servlet-name>s2</servlet-name>
    <url-pattern>/servlet2</url-pattern>
    </servlet-mapping>

    </web-app>
```

2) Hidden Form Field

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

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

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

<input type="hidden" name="uname" value="SVIT">

Advantage of Hidden Form Field

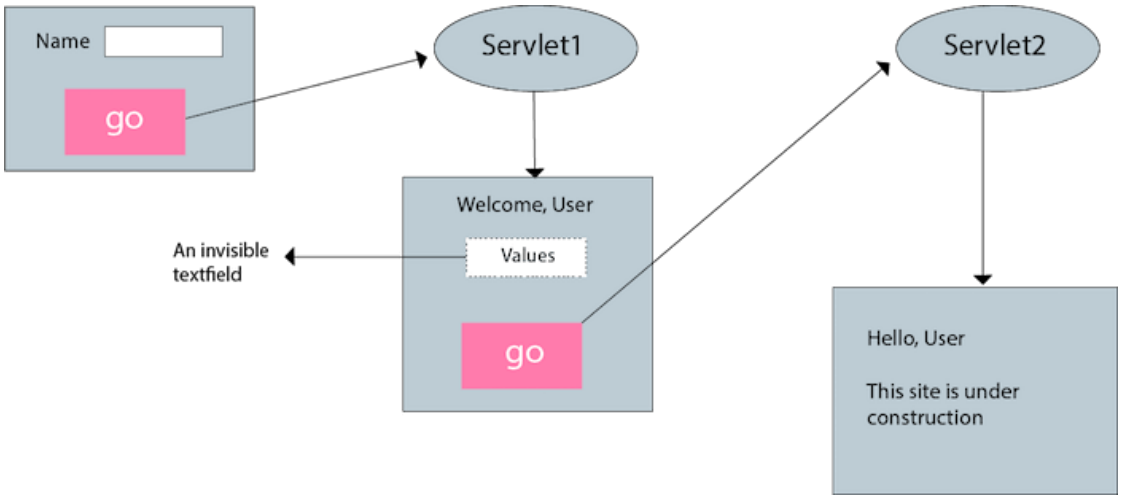
1. It will always work whether cookie is disabled or not.

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.

Example of using Hidden Form Field

In this example, we are storing the name of the user in a hidden textfield and getting that value from another servlet.



index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

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

public class FirstServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response){
        try{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String n=request.getParameter("userName");
        out.print("Welcome "+n);

        //creating form that have invisible textfield
        out.print("<form action='servlet2'>");
        out.print("<input type='hidden' name='uname' value='"+n+"'>");
        out.print("<input type='submit' value='go'>");
        out.print("</form>");
        out.close();

                }catch(Exception e){System.out.println(e);}
    }

}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
        try{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        //Getting the value from the hidden field
        String n=request.getParameter("uname");
        out.print("Hello "+n);

        out.close();
                }catch(Exception e){System.out.println(e);}
```

}}

web.xml

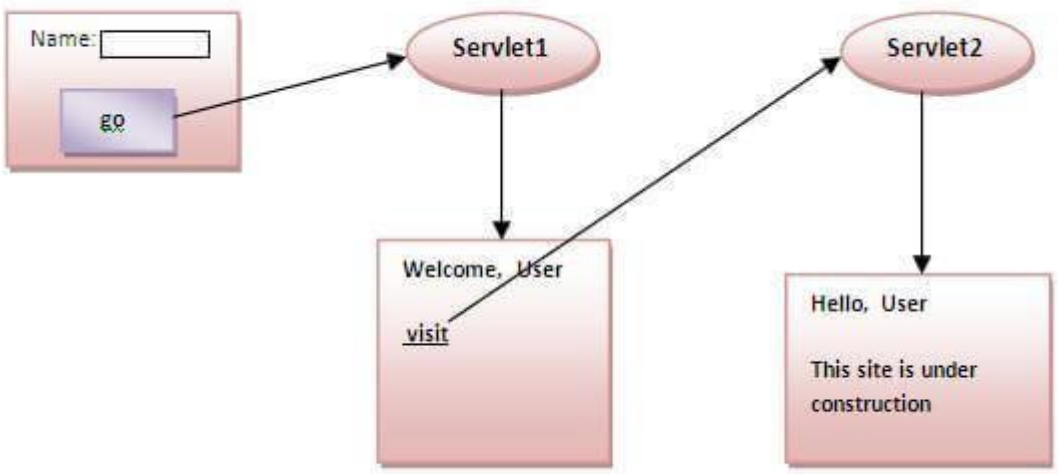
```
<web-app>
  <servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

3)URL Rewriting

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

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

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



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.

Example of using URL Rewriting

In this example, we are maintaning the state of the user using link. For this purpose, we are appending the name of the user in the query string and getting the value from the query string in another page.

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

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

```
public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
      try{

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String n=request.getParameter("userName");
      out.print("Welcome "+n);

      //appending the username in the query string
      out.print("<a href='servlet2?uname="+n+"'>visit</a>");

      out.close();

            }catch(Exception e){System.out.println(e);}
   }

}
```

SecondServlet.java

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

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
      try{

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      //getting value from the query string
      String n=request.getParameter("uname");
      out.print("Hello "+n);

      out.close();

            }catch(Exception e){System.out.println(e);}
```

```
} }
```

web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

4) HttpSession interface

The HttpSession object is used for session management. 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. This unique ID can be stored into a cookie or in a request parameter.

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.

This is how you create a HttpSession object.

How to get the HttpSession object ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

public String getId(): Returns a string containing the unique identifier value.

public long getCreationTime(): Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

public long getLastAccessedTime(): Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.

public void invalidate(): Invalidates this session then unbinds any objects bound to it.

public void setAttribute(String name, Object value): Binds the object with a name and stores the name/value pair as an attribute of the HttpSession object. If an attribute already exists, then this method replaces the existing attributes.

public Object getAttribute(String name): Returns the String object specified in the parameter, from the session object. If no object is found for the specified attribute, then the getAttribute() method returns null.

public Enumeration getAttributeNames(): Returns an Enumeration that contains the name of all the objects that are bound as attributes to the session object.

public void removeAttribute(String name): Removes the given attribute from session.

setMaxInactiveInterval(int interval): Sets the session inactivity time in seconds. This is the time in seconds that specifies how long a sessions remains active since last request received from client.

Session Example

index.html

```
<form action="login">
  User Name:<input type="text" name="userName"/><br/>
  Password:<input type="password" name="userPassword"/><br/>
  <input type="submit" value="submit"/>
</form>
```

MyServlet1.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MyServlet1 extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response){
     try{
      response.setContentType("text/html");
      PrintWriter pwriter = response.getWriter();

      String name = request.getParameter("userName");
      String password = request.getParameter("userPassword");
      pwriter.print("Hello "+name);
      pwriter.print("Your Password is: "+password);
      HttpSession session=request.getSession();
      session.setAttribute("uname",name);
      session.setAttribute("upass",password);
      pwriter.print("<a href='welcome'>view details</a>");
      pwriter.close();
    }catch(Exception exp){
       System.out.println(exp);
     }
  }
}
```

MyServlet2.java

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

```
public class MyServlet2 extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response){
  try{
      response.setContentType("text/html");
      PrintWriter pwriter = response.getWriter();
      HttpSession session=request.getSession(false);
      String myName=(String)session.getAttribute("uname");
      String myPass=(String)session.getAttribute("upass");
      pwriter.print("Name: "+myName+" Pass: "+myPass);
      pwriter.close();
  }catch(Exception exp){
      System.out.println(exp);
   }
  }
}
```

web.xml

```
<web-app>
<servlet>
   <servlet-name>Servlet1</servlet-name>
   <servlet-class>MyServlet1</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>Servlet1</servlet-name>
   <url-pattern>/login</url-pattern>
</servlet-mapping>
<servlet>
   <servlet-name>Servlet2</servlet-name>
   <servlet-class>MyServlet2</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>Servlet2</servlet-name>
   <url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
```

Output:

First Screen:

After clicking Submit:

After clicking view details:

JSP Introduction

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) 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.

3) 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.

4) Less code than Servlet

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.