

UNIT - V

Applets

There are two types of Java Programs.

- 1) Standalone application programs
- 2) Applet programs

Standalone application program is a program that is run on our system using java runtime environment. Applet is a small java program that can be executed by a java compatible web browser or applet viewer. Applet is a graphical user interface used to execute forms.

Applet is a special type of program that is embedded into a webpage to generate the dynamic content. It runs inside the web browser and works at client side. Applet is embedded in a HTML page using the APPLET or OBJECT tag and hosted on a web server.

All Applets are sub-classes (either directly or indirectly) of *java.applet.Applet* class. Applets are not stand-alone programs. Instead, they run within either a web browser or an appletviewer. JDK provides a standard applet viewer tool called appletviewer.

In general, execution of an applet does not begin at main() method. Output on applet window is not performed by *System.out.println()*. Rather it is handled with various AWT(Abstract Window Toolkit) methods, such as *drawString()*.

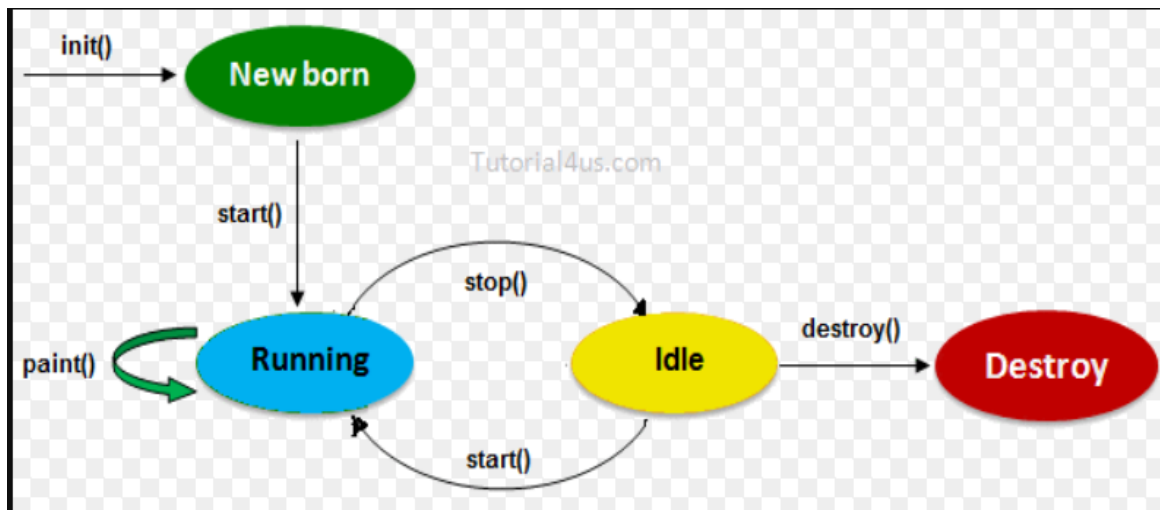
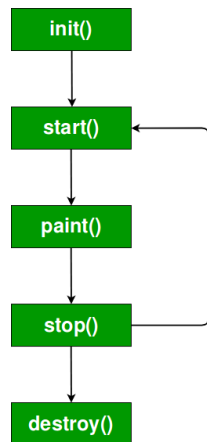
Differences between Application programs and Applet programs:

Application Programs	Applet Programs
Standalone application program is a program that is run on our system using java runtime environment.	Applet is a small java program that can be executed by a java compatible web browser or appletviewer.
It requires a main() method for its execution.	It does not require a main() method for its execution.
Created by writing the program inside the main() method.	Created by extending the Applet class.
Executed using Java Runtime Environment	Executed either by a java compatible web browser or appletviewer tool.

Advantages of Applet:

- Applets are used to make the web site more dynamic.
- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many platforms, including Linux, Windows, MacOS etc.

Java Applet Life Cycle:



Note: init() and destroy() methods are executed only once for the entire life cycle.

Lifecycle methods for Applet:

The `java.applet.Applet` class provides 5 lifecycle methods.

java.applet.Applet class

For creating any applet `java.applet.Applet` class must be imported. It provides 5 life cycle methods of applet.

1. **public void init():** The `init()` method is the first method to be called and is used to initialize the Applet. It is invoked only once.
2. **public void start():** is invoked after the `init()` method or browser is maximized. It is used to start the Applet. This is running state of Applet.

Note: `init()` is called once i.e. when the first time an applet is loaded whereas `start()` is called each time an applet's HTML document is displayed on screen. So, if a user leaves a web page and comes back, the applet resumes execution at `start()`.

3. **public void paint(Graphics g):** is used to paint the Applet. It provides `Graphics` class object that can be used for drawing oval, rectangle, arc etc.
The `paint()` method is called each time an AWT-based applet's output must be redrawn. `paint()` is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, `paint()` is called. The `paint()` method has one parameter of type `Graphics`. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running.

Note: `Graphics` class is present in `java.awt` package. So import `java.awt.*`;

4. **public void stop():** is used to stop the Applet. It is invoked when Applet is stopped or browser is minimized. The `stop()` method is called when a web browser leaves the HTML document containing the applet.
5. **public void destroy():** is used to destroy the Applet. It is invoked only once. The `destroy()` method is called when the environment determines that your applet needs to be removed completely from memory.

Applet is a predefined class and it has the following lifecycle methods Class

```
Applet{
    public void init(){
        showStatus("applet initialized");
    }
    public void start(){
        showStatus("applet started");
    }

    public void stop(){
        showStatus("applet stopped");
    }
    public void destroy(){
        showStatus("applet destroyed");
    }
    public void paint(Graphics g){
        //null implementation;
    }
}
```

There are **two** standard ways in which you can run an applet:

1. Executing the applet within a Java-compatible web browser.
2. Using applet viewer, such as the standard tool, appletviewer. An applet viewer executes your applet in a window. This is generally the fastest and easiest way to test your applet.

There are two ways to run an applet through appletviewer

- (i) By appletviewer tool (along with java program).
- (ii) By html file (java program + html file)

(i) Byappletviewertool **Program**

1

```
//AppletEx2.java

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

/*<appletcode="AppletEx2.class"width="500"height="500">
</applet>*/

public class AppletEx2 extends Applet {

    //Overriding paint() method public

    void paint(Graphics g) {

        g.drawString("This is my first applet", 100, 150);

    }

}
```

Note: Comments can be written anywhere in the program. But best practice is to write before class

Compile Program: javac AppletEx2.java

Execute Program: appletviewer AppletEx2.java

(xii) Byhtml file(javaprogram+htmlfile separately)

Program 2

```
//AppletEx1.java

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

public class AppletEx1 extends Applet {

    //Overriding paint() method

    public void paint(Graphics g) {
```

Object Oriented Programming Through Java(25CS303)

```
        g.drawString("Thisismyfirst applet",100,150);  
  
    }  
  
}
```

Compileprogram:javacAppletEx1.java

//myapplet.html

```
<html>  
<body>  
<appletcode="AppletEx1.class"width="300"height="300">  
</applet>  
</body>  
</html>
```

Executeprogram:appletviewermyapplet.html



Program3Appletprogramtodemonstrateappletlifecyclemethods

```
import java.awt.*;

import java.applet.*;

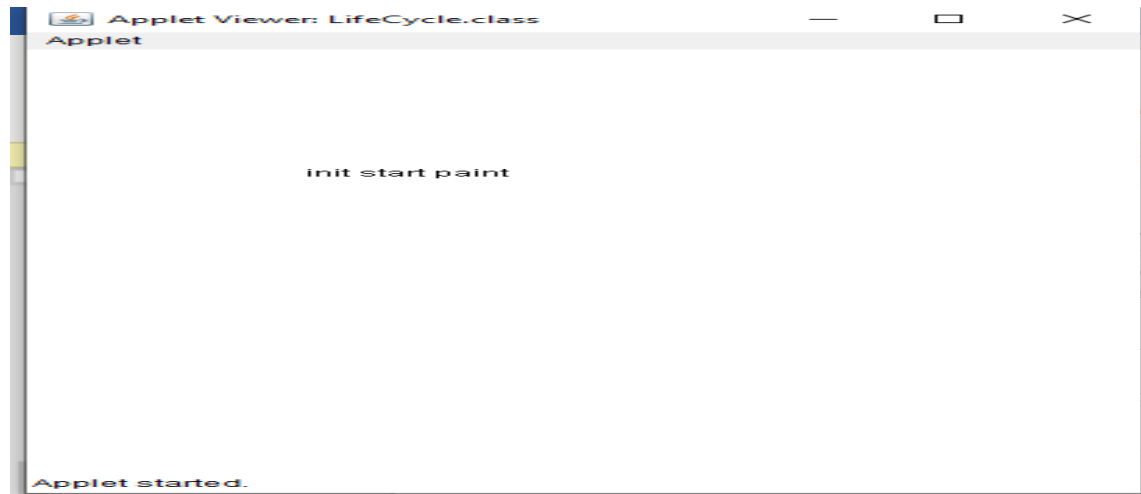
/*<appletcode="LifeCycle.class"height=500width=400>

</applet>*/

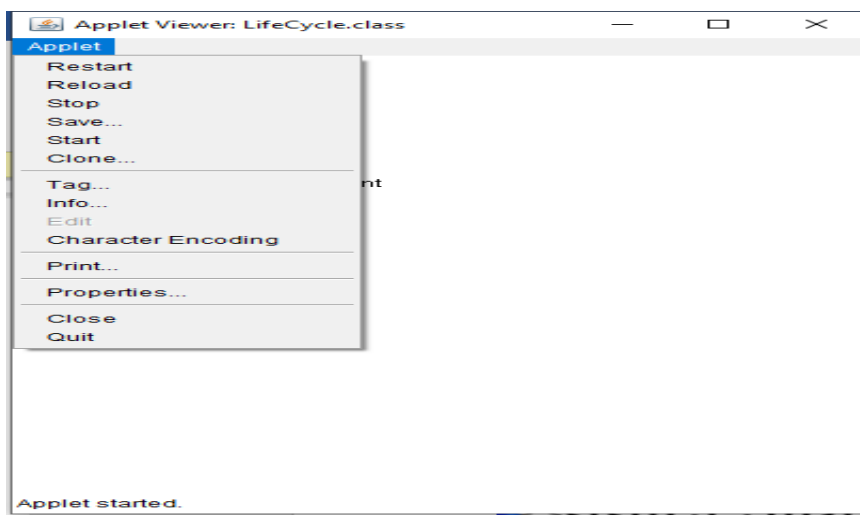
public class LifeCycle extends Applet{

    String S="";
    //Overriding all methods of Applet life cycle
    public void init(){
        S=S+"init";
    }
    public void start(){
        S=S+"start";
    }
    public void paint(Graphics g){
        S=S+"paint ";
        g.drawString(S,100,150);
    }
    public void stop(){
        S=S+"stop";
    }
    public void destroy(){
        S=S+"destroy";
    }
}
```

Object Oriented Programming Through Java(25CS303)



Note: Click on Applet-on-Appletwindow

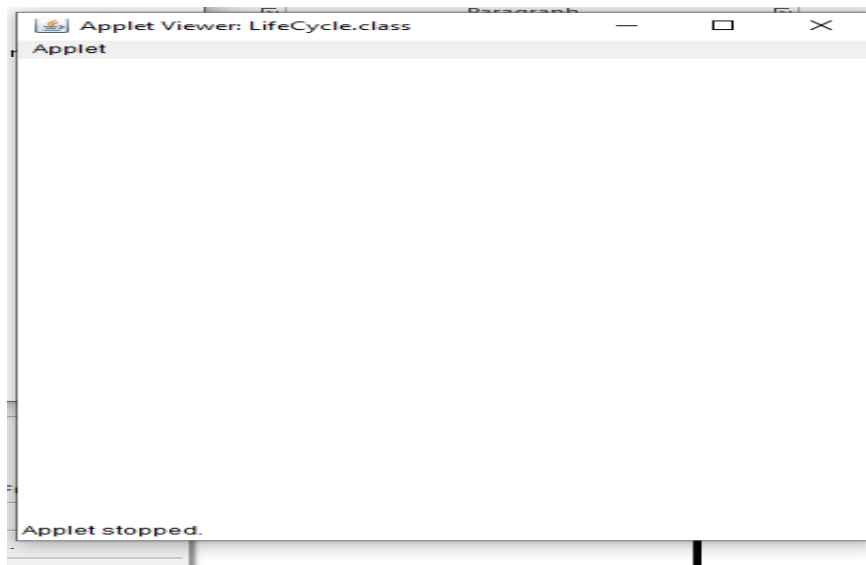


Object Oriented Programming Through Java(25CS303)

Note:Afterclickingon Restart weget thefollowingoutput

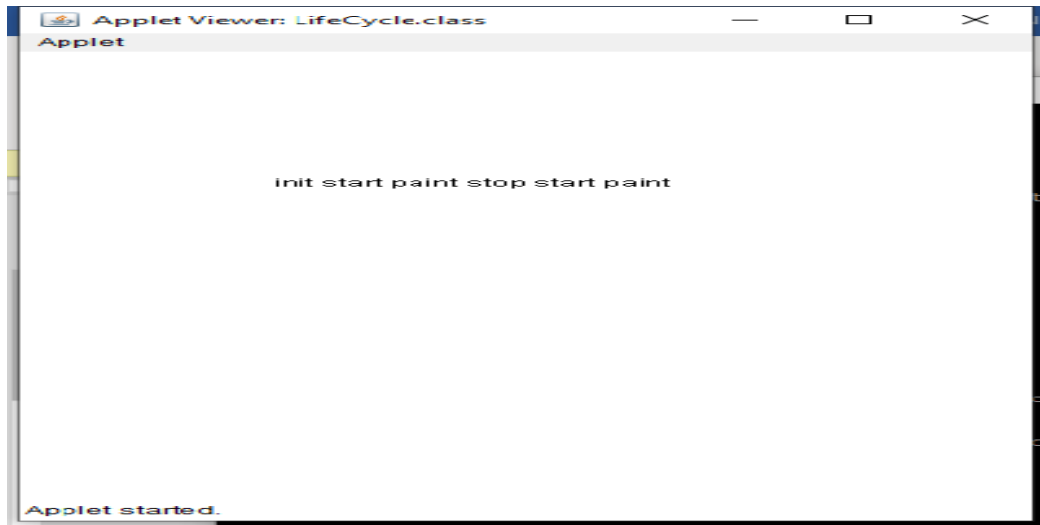


Note:Afterclickingon Stopweget thefollowingoutput.Theapplet entersidlestate.



Object Oriented Programming Through Java(25CS303)

Note: To enter into the running state from the idle state, click on the start button. The following applet window is created.



Graphics class methods:-

Graphics is a predefined class which is available in the `awt` package. Methods of graphics are

- 1) **void** `drawString(String str, int x, int y)`
- 2) **void** `drawLine(int start_x, int start_y, int end_x, int end_y)`
- 3) **void** `drawRect(int start_x, int start_y, int width, int height)`
- 4) **void** `fillRect(int start_x, int start_y, int width, int height)`
- 5) **void** `drawRoundRect(int start_x, int start_y, int width, int height, int x_Diameter, int y_Diameter)`
- 6) **void** `fillRoundRect(int start_x, int start_y, int width, int height, int x_Diameter, int y_Diameter)`
- 7) **void** `drawOval(int start_x, int start_y, int width, int height)`
- 8) **void** `fillOval(int start_x, int start_y, int width, int height)`
- 9) **void** `drawArc(int start_x, int start_y, int width, int height, int startAngle, int sweepAngle)`
- 9) **void** `fillArc(int start_x, int start_y, int width, int height, int startAngle, int sweepAngle)`
- 10) **void** `drawPolygon(int x[], int y[], int no_of_points)`

Note: In `drawPolygon`, `x` and `y` are integer arrays which hold (x,y) points. The first point and the last point should be same. For example to draw a square we need four points but while using `drawPolygon` method we require 5 points. As the starting point and ending point are same.

Program4:

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

/*<appletcode="AppletDrawing.class"height=500width=400>
</applet>*/

public class AppletDrawing extends Applet{

    public void paint(Graphicsg){

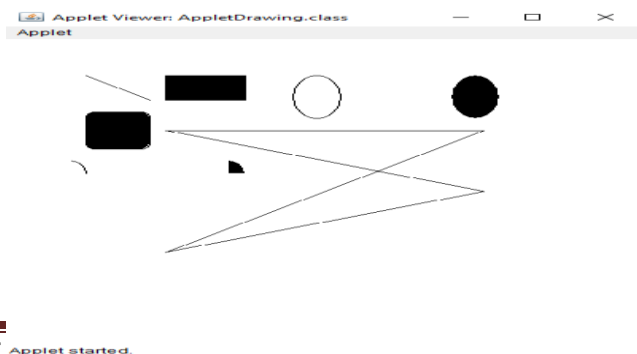
        g.drawLine(50,60,90,100);
        g.drawRect(100,60,50,40);
        g.fillRect(100,60,50,40);
        g.drawRoundRect(50,120,40,60,10,20);
        g.fillRoundRect(50,120,40,60,10,20);
        g.drawOval(180,60,30,70);
        g.fillOval(280,60,30,70);
        g.drawArc(30,200,20,40,0,90);
        g.fillArc(130,200,20,40,0,90);

        intx[]={ 100,300,100,300,100};
        inty[]={ 150,150,350,250,150};

        g.drawPolygon(x,y,5);

    }

}
```



Passing parameters to applets:

We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named `getParameter()`.

Syntax: **public** String `getParameter`(String parameterName)

Program 5:

```
//Passing parameter to Applets

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

/*<applet code="AppletParam.class" width="300" height="300">
  <param name="msg" value="Welcome to applet">
</applet>   */

public class AppletParam extends Applet {

    public void paint(Graphics g) {

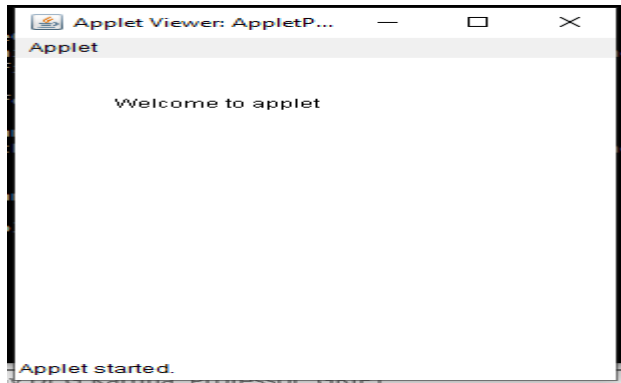
        String str = getParameter("msg");

        g.drawString(str, 50, 50);

    }

}
```

Object Oriented Programming Through Java(25CS303)



Program6:

```
//PassingparameterstoApplets (FactorialProgram)

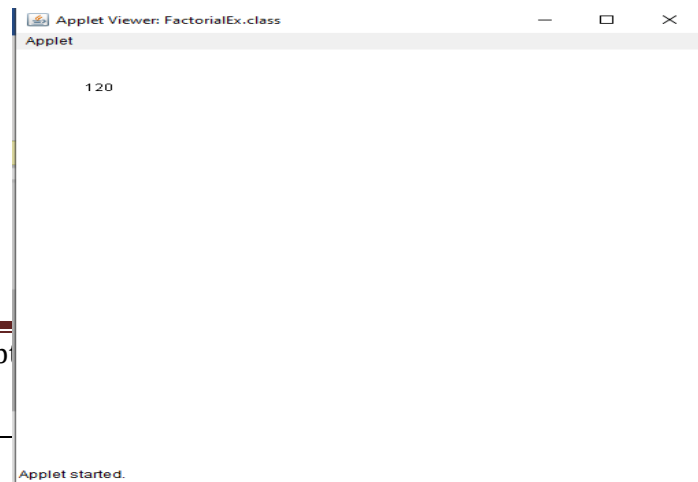
import java.applet.*;
import java.awt.*;

/*<appletcode="FactorialEx6.class"width="500"height="500">
<param name="x"value=5></param>
</applet>    */

public class FactorialEx6 extends Applet{

    public void paint(Graphicsg){

        Stringstr=getParameter("x");
        int y=Integer.parseInt(str);
        int fact=1;
        for(inti=1;i<=y;i++)
            fact=fact*i; g.drawString(fact+"",50,50);
    }
}
```

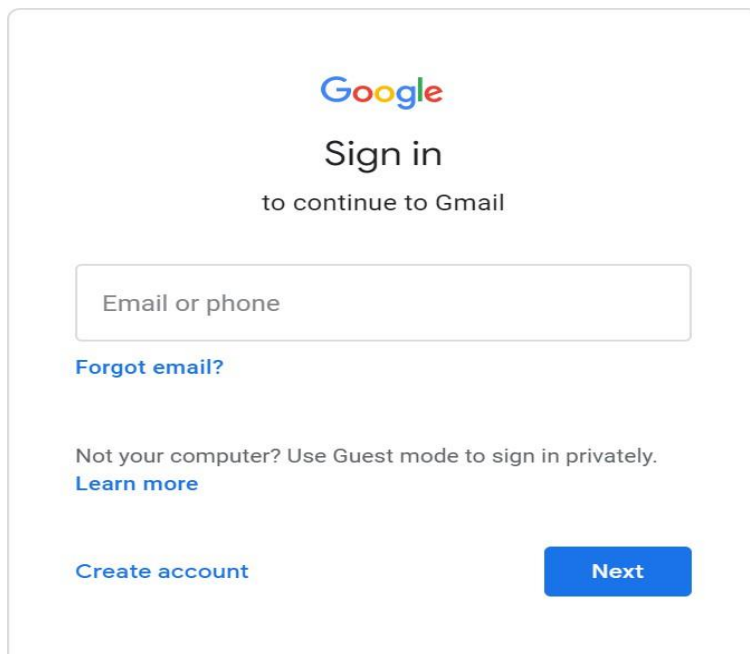


EventHandling

What is Event Handling?(Delegation Event Model)

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. The **delegation event model** defines standard and consistent mechanisms to generate and process events. Its concept is quite simple: **a source generates an event and sends it to one or more listeners**. In this scheme, **the listener simply waits until it receives an event. Once received, the listener processes the event and then returns.**

Note: In the following example, the button **Next** is the **Event source**. When we click the button, an **Event** is generated. The **Event** is listened by the **listener** to perform the next action going to the next page.



The Delegation Event Model has the following key participants namely:

- **Event** - In the delegation model, **an event is an object that describes a state change in a source**. It can be generated as a consequence of a person interacting with the elements in a graphical user interface. Some of the activities that cause events to be generated are **pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse.**

Object Oriented Programming Through Java(25CS303)

- **EventSource**-A source is an object that generates an event. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method.

Here is the general form: `public void addTypeListener(TypeListener t)` Note: `TypeListener` is changed depending on the Event.

- **EventListener**-A listener is an object that is notified when an event occurs. It has two major requirements. First, it must have been registered with one or more sources to receive notifications about specific types of events. Second, it must implement methods to receive and process these notifications.

Event Delegation Methods

(i) Event

☐ predefined class

(ii) Event Source

☐ AWT controls (text area, text fields, scroll bars, list, button, labels, choice)

(iii) EventListener

☐ predefined interface

- When an action is performed on an event source, an event is generated.
- The event listeners will wait until an event is generated.
- Once an event is performed, one or more listeners will listen to the events and parse the corresponding methods.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener

Object Oriented Programming Through Java(25CS303)

WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Note: In event handling programs, we may override `init()` and `paint()` method.

Init() method is overridden when we need to do following:

```
public void init(){
    //1.Create objects for class using constructor
        (for event sources which are awt controls)
    //2.Registration of listener to source (source can be applet window/awt controls)
        (public void addTypeListener(TypeListener t))
    //3.Adding our own awt controls to applet.
        (add() method is used)
}
```

paint() method is overridden when we want to display something on the applet window.

Handling mouse and keyboard Events

1) Mouse events:-

This event indicates a mouse action occurred in a component. This low-level event is generated by a component object for Mouse Events and Mouse Motion events.

MouseEvent:-(class) //class name must be the argument of the methods of interface

```
int getX()
int getY()
int getClickCounts()
```

MouseListener:-(Interface)

```
void mouseClicked(MouseEvent me)
void mousePressed(MouseEvent me)
void mouseReleased(MouseEvent me)
void mouseEntered(MouseEvent me)
void mouseExited(MouseEvent me)
```

MouseMotionListener:-(interface)

```
void mouseDragged(MouseEvent me)
```

Object Oriented Programming Through Java(25CS303)

voidmouseMoved(MouseEvent me)

//Note:For MouseEvent and Key board Eventprogramsawt controlsarenot required. For these two programs the source is applet window.

Program:Handling MouseEvent

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*<appletcode="MouEve.class"width=300 height=400></applet>*/

public class MouEve extends Applet implements MouseListener,
MouseMotionListener{
    String msg="";
    public void init(){
        //overridinginit()
        addMouseListener(this);           //toregisterourlistenerto
        applet window addMouseMotionListener(this);
    }

    public void mouseClicked(MouseEvent me){
        msg="Mouseclicked"; repaint();
    }

    public void paint(Graphics g){
        g.drawString(msg,50,60);
    }

    public void mouseEntered(MouseEvent me){
        setBackground(Color.red);
        repaint();
    }

    public void mouseExited(MouseEvent me){
        setBackground(Color.yellow);
        msg="mouse exited";
        repaint();
    }

    public void mouseReleased(MouseEvent me){
        setBackground(Color.green);
        repaint();
    }

    public void mousePressed(MouseEvent me){
        setBackground(Color.blue);
        repaint();
    }

    public void mouseDragged(MouseEvent me){
        msg="mousedragged";
```

```
repaint();
}

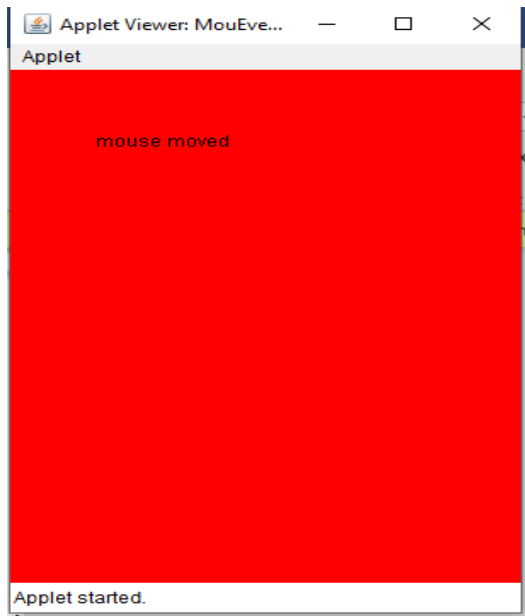
public void mouseMoved(MouseEvent me){
    msg="mousemoved";
    repaint();
}
}
```

Note: Output of paint() method when the applet is run

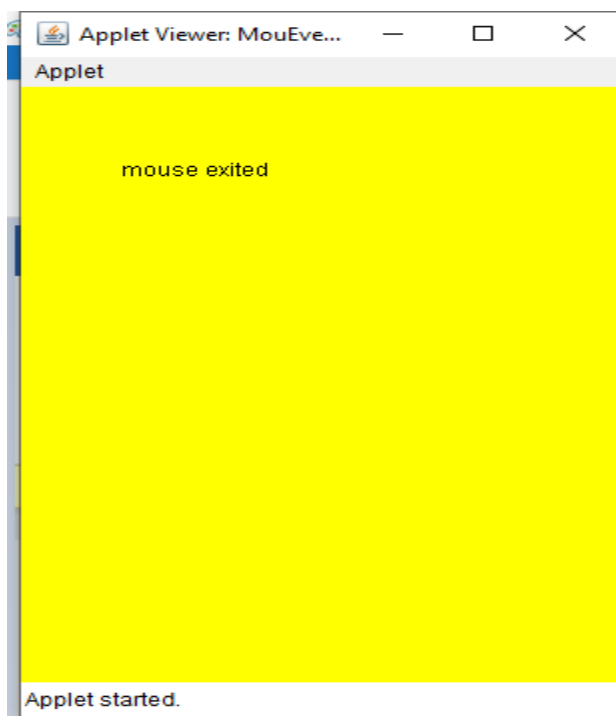


Note: output after moving the mouse pointer into the applet window. When we move the mouse pointer into the applet window two methods are performed. mouseMoved() and mouseEntered() methods. So we are able to see the message "mouse moved" present in the mouseMoved() method and the applet window color is changed to red as mouseEntered() method is also executed.

Object Oriented Programming Through Java(25CS303)

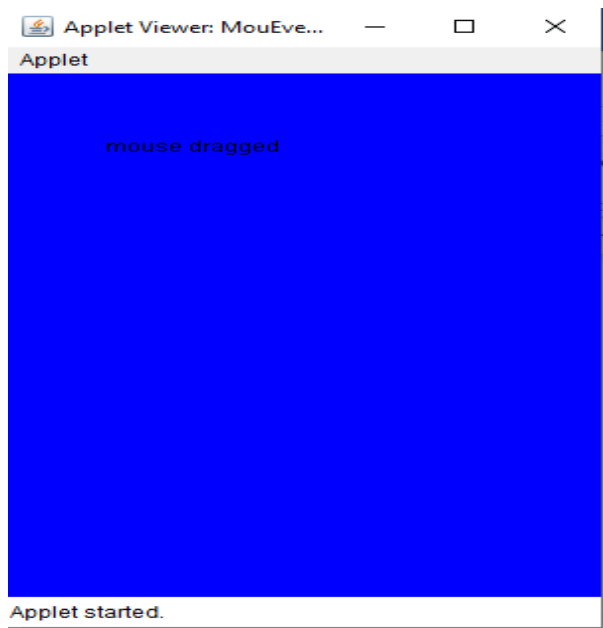


Note: output after moving the mouse pointer out of the applet window. MouseExited method is executed.

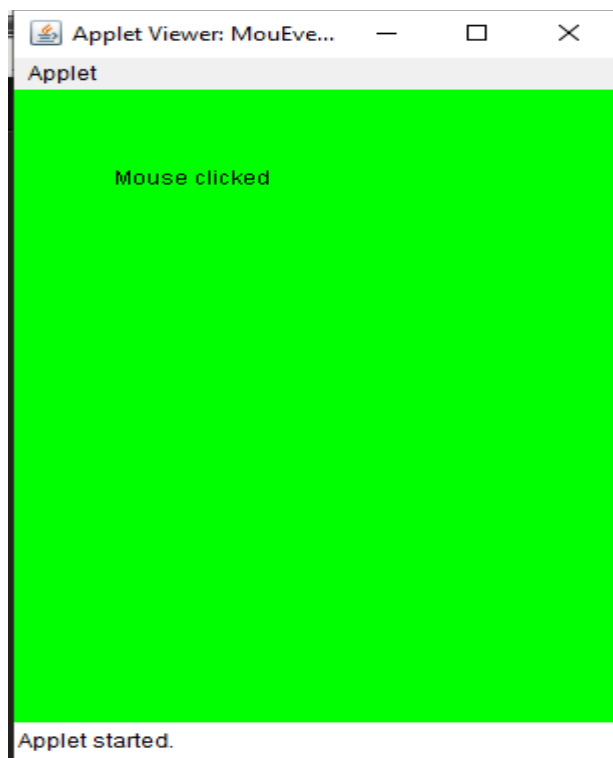


Note: For dragging the mouse we have to press the mouse button and drag so the two methods are performed `mousePressed()` and `mouseDragged()`. So from the `mousePressed()` method the color of the applet window is changed and from the `mouseDragged()` method the message "mouse dragged" is displayed on the applet window

Object Oriented Programming Through Java(25CS303)



Note: When we click the mouse three methods are executed `mousePressed()`, `mouseReleased()` and `mouseClicked()`. From the `mouseClicked()` method the message "mouse clicked" is displayed and from the `mouseReleased()` method the color of the applet is changed to green.



repaint()method

The paint() method supports painting via a Graphics object. The repaint() method is used to cause paint() to be invoked by the AWT painting thread. It's not necessary to call repaint unless you need to render something specific onto a component. As the applet needs to update information displayed in its window (i.e. redraw the window), each time the mouse is being clicked so this is possible with the use of repaint() method. To sum up, the repaint() method is invoked to refresh the viewing area.

2) Key events:- This event indicates how the keys in keyboard are repressed/clicked.

KeyEvent(class)

- (i) char getKeyChar()
- (ii) int getKeyCode()

KeyListener(interface)

- (i) void keyPressed(KeyEvent ke)
- (ii) void keyTyped(KeyEvent ke)
- (iii) void keyReleased(KeyEvent ke)

// Key events program

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

/*<appletcode="KeyEve.class"width=200height=300>
</applet>*/

public class KeyEve extends Applet implements KeyListener{

    Char ch=' ';

    public void init(){
        addKeyListener(this);          //to register our listeners to applet
        requestFocus(); //to move the focus of keyword to applet window
    }

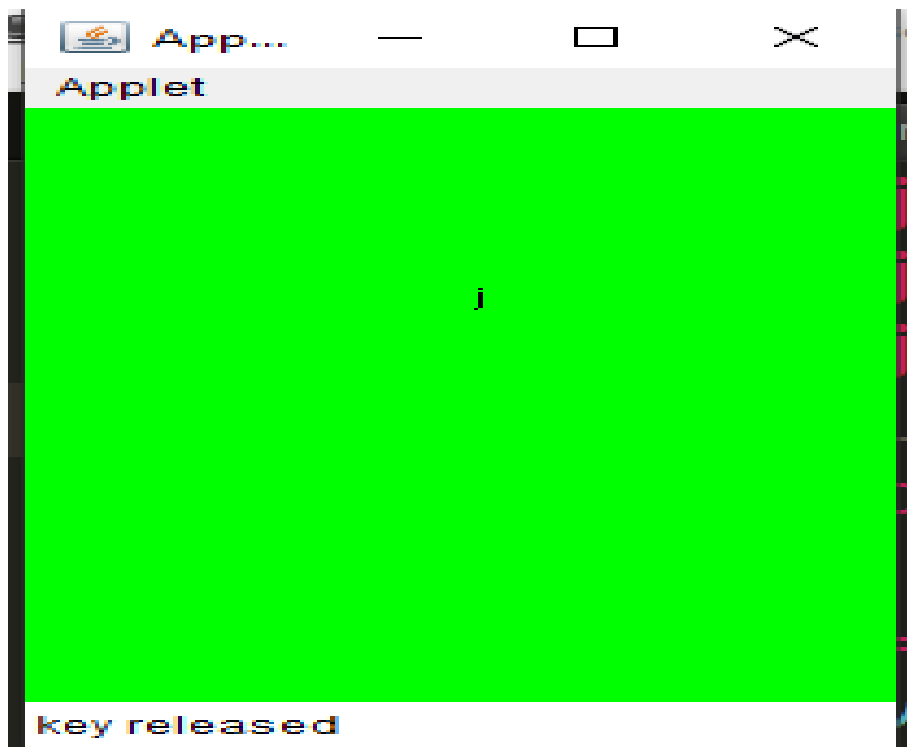
    public void keyPressed(KeyEvent ke){
        setBackground(Color.green);
    }

    public void keyReleased(KeyEvent ke){
        showStatus("key released");
    }
}
```

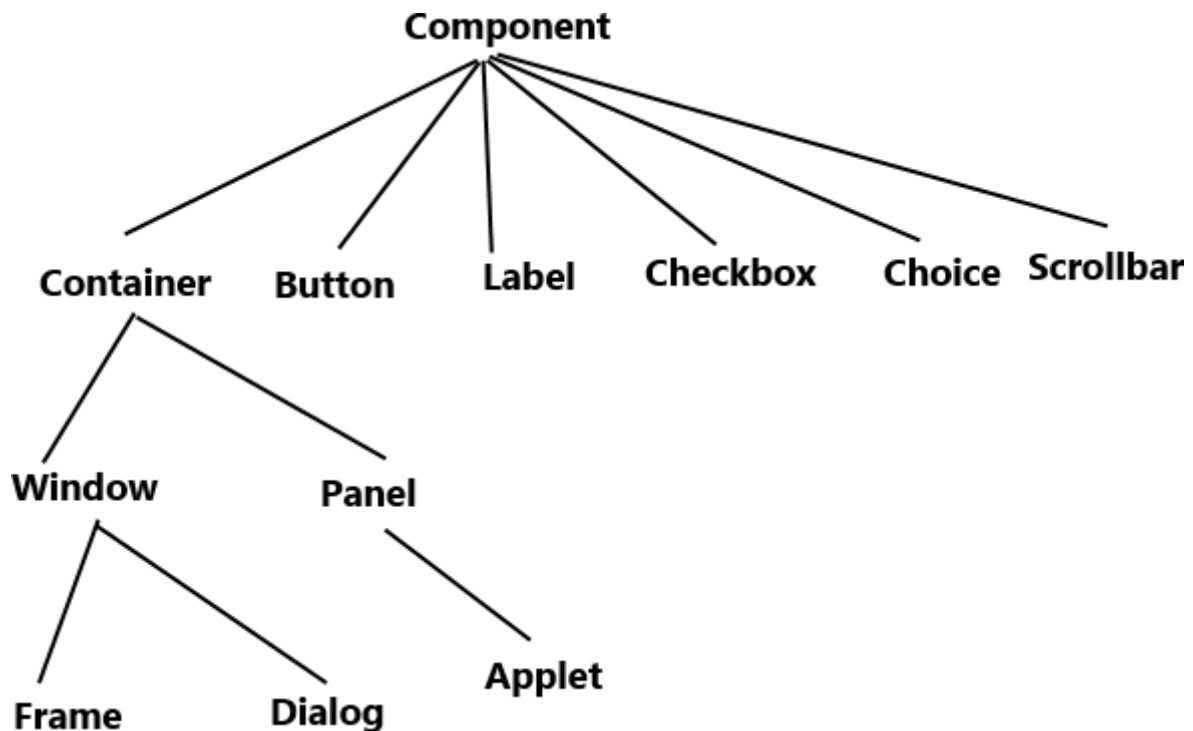
Object Oriented Programming Through Java(25CS303)

```
public void keyTyped(KeyEvent ke){  
    ch=ke.getKeyChar();  
    repaint();  
}  
  
public void paint(Graphics g){  
    g.drawString(""+ch,100,100);  
}  
}
```

Note: When the keyTyped keyPressed(), keyReleased() and keyTyped() methods are executed.



AWTClassHierarchy:-



AwtComponents

Label, TextField, Button, Choice, List, Checkbox, CheckboxGroup, TextArea, Scrollbars

1. Label :-

Label is a class.

The object of Label class is a component for placing text in a container. It is used to display a single line of read-only text. The text can be changed by an application but a user cannot edit it directly.

Constructors:

- (i) Label()
 - (ii) Label(String str)
 - (iii) Label(String str, int how)
- how = Label.LEFT, Label.RIGHT, Label.CENTER

Methods in Label:

```
void setText(String str)
String getText()
void setAlignment(int how)
int getAlignment()
```

2. Text Fields:-

TextField-is a class.

The [object](#) of a TextField class is a text component that allows the editing of a single line text. It inherits TextComponent class.

- (i) TextField()
- (ii) TextField(int numChar) // for visibility
- (iii) TextField(String str) // default String
- (iv) TextField(String str, int numChar)

Methods in TextField:

```
String getText()
void setText(String str)
void getSelectedText() // select using the cursor
void select(int start index, int end index)
boolean isEditable() // default true
void setEchoChar(char ch) // for passwords
boolean echoCharIsSet()
char getEchoChar()
```

3. Buttons:- Button-is a class

The button class is used to create a labeled button that has platform independent implementation. The application results in some action when the button is pushed.

- (i) Button()
- (ii) Button(String str)

Methods in Button Class:

```
void setLabel(String str)
String getLabel()
```

Action Events:-

ActionEvent-is a class

Methods in ActionEvent class:

```
Object getSource() // returns the reference variable of the Button
String getActionCommand() // it returns the label of Button
```

ActionListener-is an Interface

Object Oriented Programming Through Java(25CS303)

The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in `java.awt.event` package. It has only one method: `actionPerformed()`.

Method in ActionListener:

public void actionPerformed(ActionEvent ae)

// Program on Labels and Textfields

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

/*<applet code="LabelsTextEx9.class" width="600" height="600">
</applet>*/

public class LabelsTextEx9 extends Applet implements ActionListener {

    Label l1, l2;

    TextField t1, t2;

    Button b1;

    // First init() and then paint() method will be executed and initial output is printed
    // on the applet window.

    // whenever we click the Button then actionPerformed() method is executed

    public void init() {

        l1 = new Label("username"); // creating objects
        l2 = new Label("password");

        t1 = new TextField(" ");
        t2 = new TextField(" ");
        t2.setEchoChar('*');
        b1 = new Button("signin");
        b1.addActionListener(this); // register the listener to the Button
        add(l1); // adding the objects to applet window
        add(t1);
        add(l2);
        add(t2);
        add(b1);
    }
}
```

```
}

//when we click the Button actionPerformed is invoked by the ActionListener

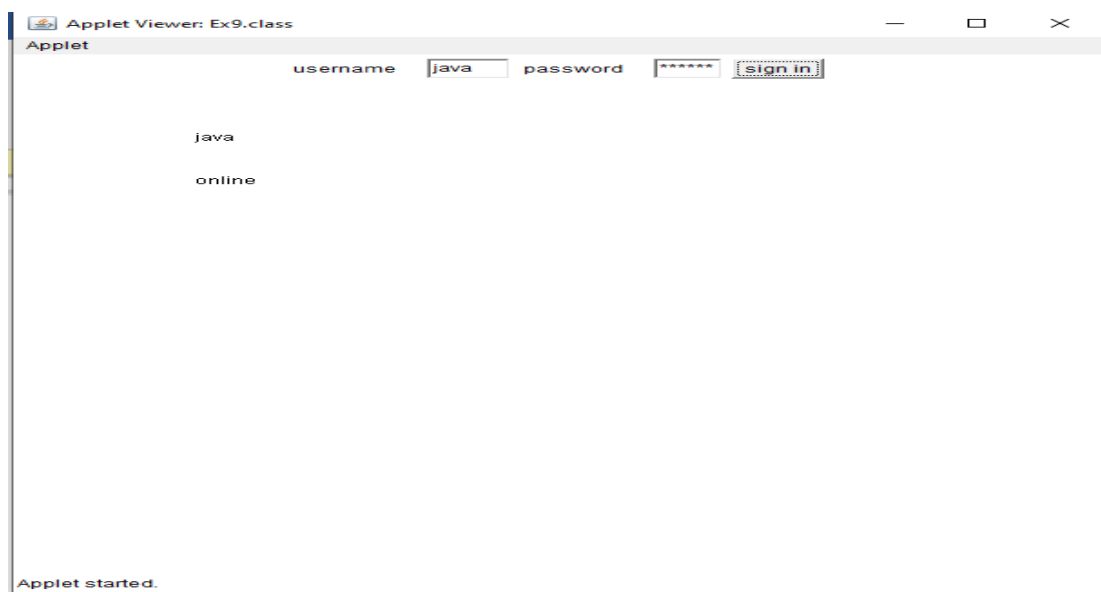
public void actionPerformed(ActionEvent ae){

    repaint();
}

public void paint(Graphics g){

    g.drawString(t1.getText(),100,100);
    g.drawString(t2.getText(),100,150);
}

}
```



//java Program on Buttons

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

/*<appletcode="ButtonsEx10.class"width=500
height=400></applet>*/

public class ButtonsEx10 extends Applet implements ActionListener{

    Button b1,b2,b3;

    //Firstinit()andthenpaint()methodwill beexecutedand
    //initial outputisprintedontheappletwindow.
    //whenevertheButtonisclickedactionPerformed()methodisexecuted

    public void init(){

        b1=new Button("Red");
        b2=newButton("Yellow");
        b3=newButton("Green");

        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
        add(b1);
        add(b2);
        add(b3);
    }

    public void actionPerformed(ActionEvent ae){

        if(ae.getSource()==b1)

            //getSource()methodreturnstheobjectreferenceofButton

            setBackground(Color.red);

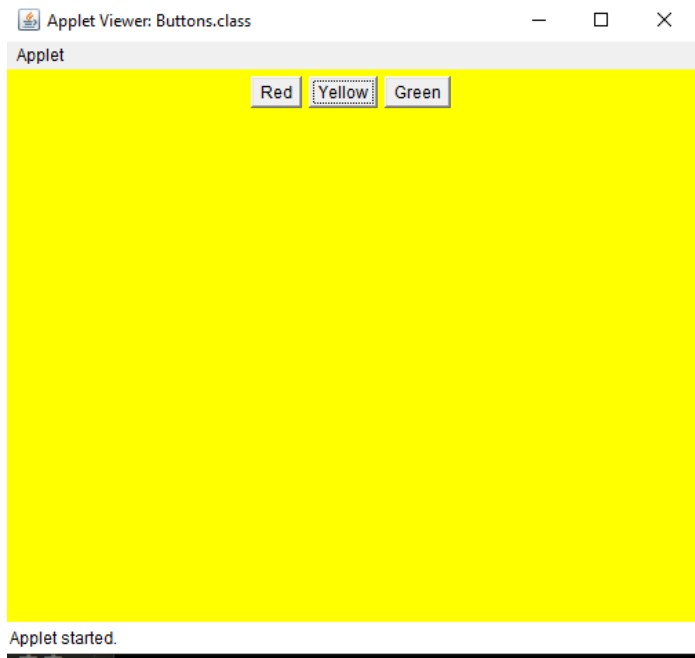
        else if(ae.getSource()==b2)

            setBackground(Color.yellow);

        else

            setBackground(Color.green);
```

```
}  
  
}
```



4. Choice

The object of **Choice** class is used to show popup menu of choices. Choices selected by user is shown on the top of a menu. It inherits Component class. Only one can be selected, visibility and selection may or may not be same.

Constructor:

Choice()

Methods in Choice class:-

void add(String name)

int getSelectedIndex() // returns the index of the selected item String

getSelectedItem() // returns the item which is selected int

getItemCount() // Total number of items in Choice

void select(int index) // default selection

void select(String name) // selecting a String from the

items String getItem(int index) // returns the item in the particular index

Item Events:-

ItemEvent is a class

Object getItem()

ItemListener is an interface

void itemStateChanged(ItemEvent e)

// the item state will change from either selection to non-selection or non-selection to selection

//Program on Choice component

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

/*<appletcode="ChoiceEx12.class"height=400width=300></applet>*/

public class ChoiceEx12 extends Applet implements ItemListener{

    Choice c1,c2,c3;

    //First init() and then paint() method will be executed and initial output is
    //printed on the applet window. whenever the state of any Choice is
    //changed it goes to itemStateChanged method

    public void init(){

        //creating objects of Choice and adding the items to the choice

        c1=new Choice();

        for(int i=1;i<=30;i++){
            c1.add(i+"");
        }

        c2=new Choice();

        c2.add("January");
        c2.add("February");
        c2.add("March");
        c2.add("April");
        c2.add("May");
        c2.add("June");
        c2.add("July");
        c2.add("August");
        c2.add("September");

        c2.add("October");
        c2.add("November");
        c2.add("December");

        c3=new Choice();
        for(int i=2000;i<=2015;i++)
```

```
c3.add(i+""");

add(c1);
add(c2);
add(c3);

c1.addItemListener(this);
c2.addItemListener(this);
c3.addItemListener(this);
}

public void itemStateChanged(ItemEvent ie){

    repaint();

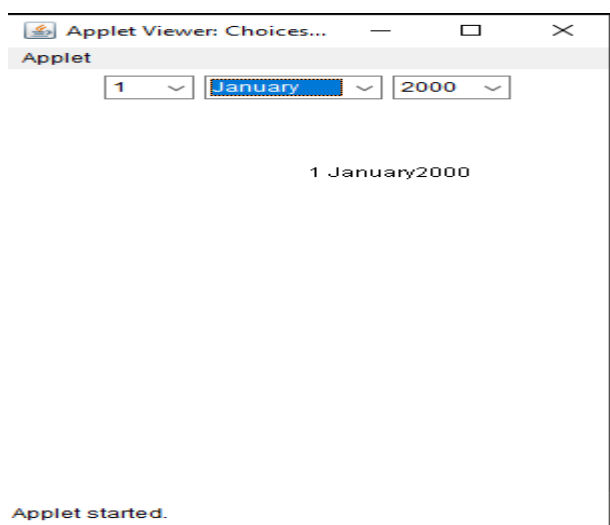
}

public void paint(Graphics g){

    String str1=c1.getSelectedItem()+c2.getSelectedItem()+c3.getSelectedItem();
    g.drawString(str1,150,100);

}

}
```



Object Oriented Programming Through Java(25CS303)

The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

List is a class

Used for multiple selection on items

Constructors:

List() // only single selection and visibility is one item

List(int numRows) // only single selection and visibility is numRows
List(int numRows, Boolean multipleSelect)

// if multipleSelect = true then only multiple selection is possible otherwise single selection

Methods in List:

void add(String name)

void add(String name, int index)

String getSelectedItem()

String[] getSelectedItems()

int getSelectedIndex()

int[] getSelectedIndexes()

int getItemCount()

void Select(int Index)

String getItem(int Index)

// Program on List component

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

/*<applet code="ListEx13.class" height=400 width=400>
</applet>*/

public class ListEx13 extends Applet implements ItemListener {

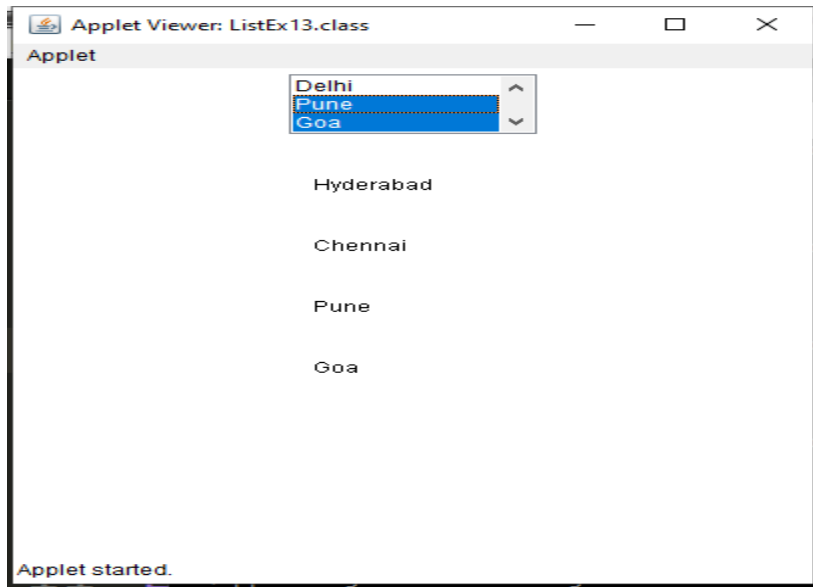
    List l1; // for city

    // First init() and then paint() method will be executed and initial output is printed on the applet window. whenever the state of any List is changed it goes to itemStateChanged method
```

```
public void init(){  
  
    L1=newList(3,true);  
  
    //3visibleoutof6 anyno.of itemsareselected  
  
    L1.add("Hyderabad");  
    L1.add("Bangalore");  
    L1.add("Chennai");  
    L1.add("Delhi");  
    L1.add("Pune");  
    L1.add("Goa");  
  
        L1.addItemListener(this);  
  
    add(L1);  
  
    }  
  
    public void itemStateChanged(ItemEventie){  
  
        repaint();  
  
    }  
  
    public void paint(Graphicsg){  
  
        Stringstr[]=L1.getSelectedItems(); for(int  
        i=0;i<str.length;i++)  
  
            g.drawString(str[i],150,100+i*50);  
  
    }  
  
}
```

Object Oriented Programming Through Java(25CS303)

Note: We can select multiple items by holding shift key. Selected items are printed as output on the applet window.



5. Checkbox

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

Constructors:

`Checkbox()`

`Checkbox(String str)` // default not selected

`Checkbox(String str, boolean on)` // true then selected

`Checkbox(String str, boolean on, CheckboxGroup cbg)`

`Checkbox(String str, CheckboxGroup cbg)`

Methods in Checkbox:

`Boolean getState()`

`void setState(boolean on)`

`String getLabel()`

`void setLabel(String str)`

// Event source is Checkbox ItemEvent:

// Event

`Object getItem()` // it returns the reference variable of the object

ItemListener: // Listener

`void itemStateChanged(ItemEvent e)`

// Program on Checkbox

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

/*<appletcode="CheckboxEx14.class"height=400width=300>
</applet>*/

public class CheckboxEx14 extends Applet implements
ItemListener{

    Checkboxcb1,cb2,cb3,cb4,cb5;

    //Firstinit()andthenpaint()methodwill beexecutedandinitial outputis
    printedontheappletwindow.wheneverthestateof anyCheckboxis
    changeditgoestoitemStateChangedmethod

    public voidinit(){

        cb1=newCheckbox("CSE");//notselected
        cb2=newCheckbox("ECE");
        cb3=newCheckbox("IT");
        cb4=newCheckbox("EEE");
        cb5=newCheckbox("CIVIL");

        add(cb1);
        add(cb2);
        add(cb3);
        add(cb4);
        add(cb5);

        cb1.addItemListener(this);
        cb2.addItemListener(this);
        cb3.addItemListener(this);
        cb4.addItemListener(this);
        cb5.addItemListener(this);

    }

    /*Whenever thestateofcheckboxischangedfromonestatetoanother
    stateItemEventisgeneratedsoitemStateChanged()methodisinvokedbyitemListener*/
```

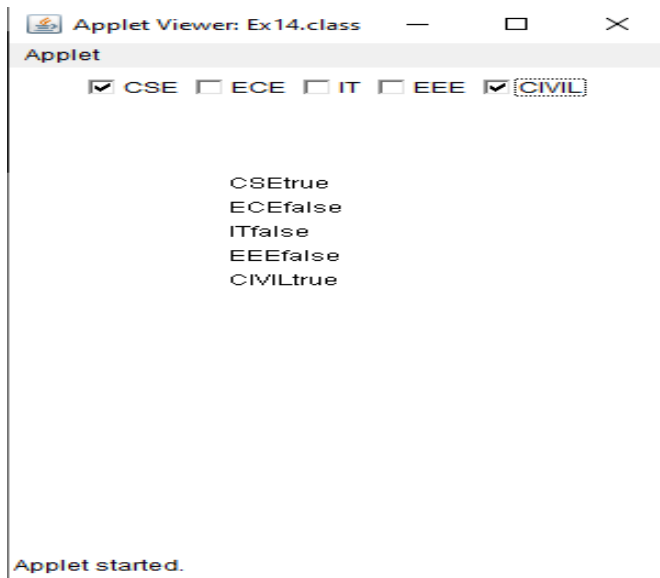
```
public void itemStateChanged(ItemEvent ie){  
  
    repaint();//overwrites the String in the output  
  
}  
  
//As part of Applet Lifecycle methods init() and paint() method will be  
//executed automatically  
  
public void paint(Graphics g){  
  
    String str1=cb1.getLabel()+cb1.getState();  
    String str2=cb2.getLabel()+cb2.getState();  
    String str3=cb3.getLabel()+cb3.getState();  
    String str4=cb4.getLabel()+cb4.getState();  
    String str5=cb5.getLabel()+cb5.getState();  
  
    g.drawString(str1,100,100);  
    g.drawString(str2,100,120);  
    g.drawString(str3,100,140);  
    g.drawString(str4,100,160);  
    g.drawString(str5,100,180);  
  
}  
  
}
```

//multiple items can be selected. The label and the state of each checkbox is printed as output.

Note: 1 When the paint() method is invoked automatically for the first time the state of all the checkboxes in the output is false.

2: Whenever the item state is changed for any Checkbox the repaint() method is called from the itemStateChanged() method then the String (Label and State) is overwritten in the output.

3: Note 1 and Note 2 are applicable for all the Event handling Programs.



6. CheckboxGroup(radiobutton)

The object of CheckboxGroup class is used to group together a set of Checkbox. At a time only one checkbox button is allowed to be in "on" state and remaining checkbox button in "off" state. It inherits the object class.

Constructor:

CheckboxGroup()

Methods

Checkbox.getSelectedCheckbox()

void setSelectedCheckbox(Checkbox which)

//Program on CheckBox

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

/*<appletcode="CheckboxEx15.class"height=400width=300>
</applet>*/

public class CheckboxEx15 extends Applet implements ItemListener{

    CheckboxGroup cbg1,cbg2;
```

```
Checkboxcb1,cb2,cb3,cb4,cb5;

//Firstinit()andthenpaint()methodwill beexecutedandinitial outputisprintedon
theappletwindow.

//wheneverthestateof anyCheckboxischangeditgoestoitemStateChangedmethod

public voidinit(){

    cbg1=newCheckboxGroup();
    cbg2=newCheckboxGroup();

    cb1=newCheckbox("CSE",cbg1,true);
    cb2=newCheckbox("ECE",cbg1,false);
    cb3=newCheckbox("IT",cbg1,false);

    cb4=newCheckbox("B.TECH",cbg2,true);
    cb5=newCheckbox("M.TECH",cbg2,false);

    add(cb1);
    add(cb2);
    add(cb3);
    add(cb4);
    add(cb5);

    cb1.addItemListener(this);
    cb2.addItemListener(this);
    cb3.addItemListener(this);
    cb4.addItemListener(this);
    cb5.addItemListener(this);

}

public voiditemStateChanged(ItemEventie){

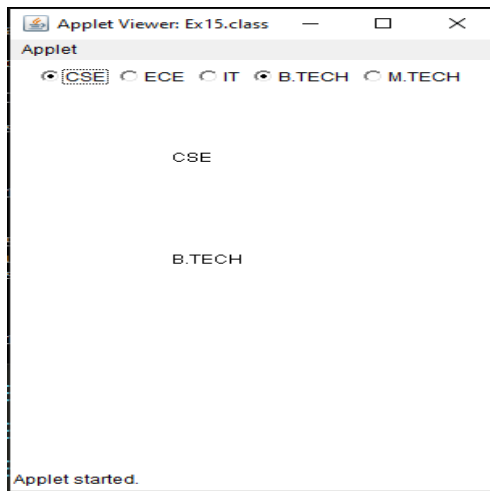
    repaint();

}

public voidpaint(Graphicsg){

    Checkboxcb6=cbg1.getSelectedCheckbox();
    g.drawString(cb6.getLabel(),100,100);
    Stringstr=(cbg2.getSelectedCheckbox()).
    getLabel(); g.drawString(str,100,200);
```

```
}  
  
}
```



Note: At a time from a group we can select one item

7. Scrollbars

The object of Scrollbar class is used to add horizontal and vertical scrollbar. Scrollbar is a GUI component that allows us to see invisible number of rows and columns.

Constructors:

Scrollbar() Scrollbar(int style)

Scrollbar(int style, int initialValue, int thumbSize, int min, int max)

If style is **Scrollbar.VERTICAL**, a vertical scrollbar is created.

If style is **Scrollbar.HORIZONTAL**, the scrollbar is horizontal.

In the third form of the constructor, the initial value of the scrollbar is passed in **initialValue**. The **initialValue** must be in between min and max values. **initialValue is the position of the thumb.**

The number of units represented by the height of the **thumb** is passed in **thumbSize**.

The minimum and maximum values for the scroll bar are specified by **min** and

max. Methods of Scrollbar:

void setValues(int initialValue, int thumbSize, int min, int max)

int getValue() // position of the thumb which is the initial value

void setValue(int newValue)

int getMinimum()

int getMaximum()

void setUnitIncrement(int newIncr) // newIncr = 3 * initialValue + unit * newIncr 14710

void setBlockIncrement(int newIncr) // newIncr = 3 * initialValue + Block * newIncr 1316191

Object Oriented Programming Through Java(25CS303)

//Blockmeans10,unitmeans1,initial valueis 1

Event

AdjustmentEvent

Listener

AdjustmentListener

publicvoidadjustmentValueChanged(AdjustmentEventae)

//ProgramonScrollbars

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

/*<appletcode="ScrollbarEx16.class"height=400width=400>

</applet>*/

public class ScrollbarEx16 extends Applet implements AdjustmentListener{

    Scrollbars1,s2,s3;
    Color c;
    int r=0,g=0,b=0;
    public void init(){

        // Scrollbar(intstyle,intinitialValue,intthumbSize,intmin,intmax)

        s1=new Scrollbar(Scrollbar.VERTICAL,10,2,0,255);
        s2=new Scrollbar(Scrollbar.VERTICAL,20,2,0,255);
        s3=new Scrollbar(Scrollbar.VERTICAL,30,2,0,255);

        add(s1);
        add(s2);
        add(s3);

        s1.setUnitIncrement(10);//10,20,30,...
        s2.setUnitIncrement(10);//20,30,40,50---.
        s3.setUnitIncrement(10);// 30,40,50---

        s1.addAdjustmentListener(this);
        s2.addAdjustmentListener(this);
        s3.addAdjustmentListener(this);
    }
}
```

```
}

public void adjustmentValueChanged(AdjustmentEvent ae){

    r=s1.getValue(); //gives the position of the thumb that is the initial value
    g=s2.getValue();
    b=s3.getValue();
    repaint();

}

public void paint(Graphics g){

    c=new
    Color(r,g,b); setBackground(c)
    ; setForeground(Color.red);

    g.drawString(r+""+g+""+b,100,150);

}

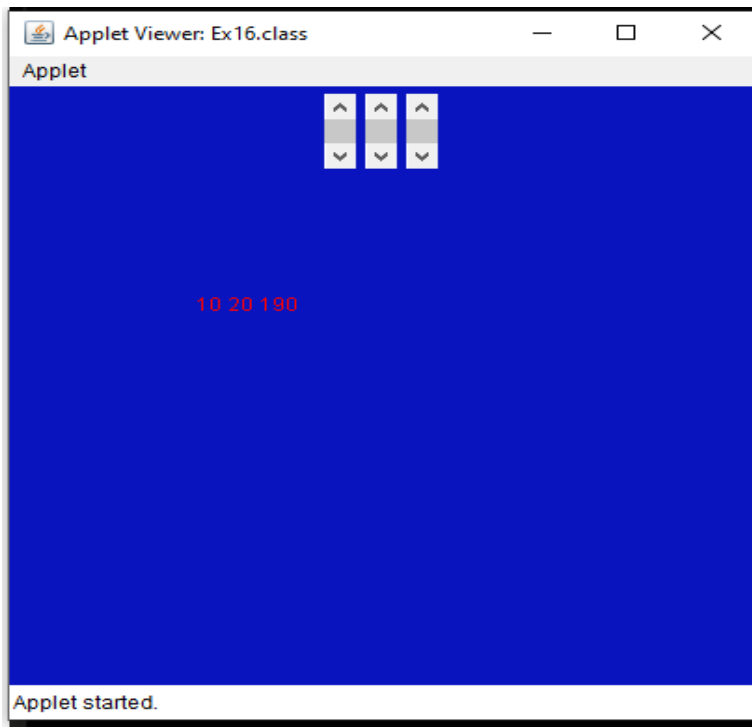
}
```

//If upward arrow is clicked the value is decreased and if downward arrow is clicked the value is increased.

//When the paint() method is invoked for the first time then the output is 000

Note: When we click any scrollbar using downward arrow for the first time then adjustmentValueChanged method will be invoked so the values of r,g,b are changed to the initial values of the scrollbar except for the scrollbar we clicked.





8. TextArea

The object of a `TextArea` class is a multiline region that displays text. It allows the editing of multiple line text. It inherits `TextComponent` class.

Syntax:

Constructors:

`TextArea()`

`TextArea(int numLines, int numChars)`

//numLines is height and numChars is the width of

the `TextArea`

`TextArea(String str, int numLines, int numChars)`

`TextArea(String str, int numLines, int numChars, int sBars)`

Here, `numLines` specifies the height, in lines, of the text area, and `numChars` specifies its width, in characters. Initial text can be specified by `str`. In the fifth form, you can specify the scrollbar that you want the control to have. `sBars` must be one of these values:

`SCROLLBARS_BOTH`

`SCROLLBARS_NONE`

`SCROLLBARS_HORIZONTAL_ONLY`

`SCROLLBARS_VERTICAL_ONLY`

`TextArea` is a subclass of `TextComponent`. Therefore, it supports the `getText()`, `setText()`, `getSelectedText()`, `select()`, `isEditable()`, and `setEditable()` methods described in the `TextField`.

TextArea adds the following methods:

void append(String str)

void insert(String str, int index)

void replaceRange(String str, int startIndex, int endIndex)

// Program on TextArea

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

/*<appletcode="TA.class"height=400width=400>
</applet>*/

public class TA extends Applet{

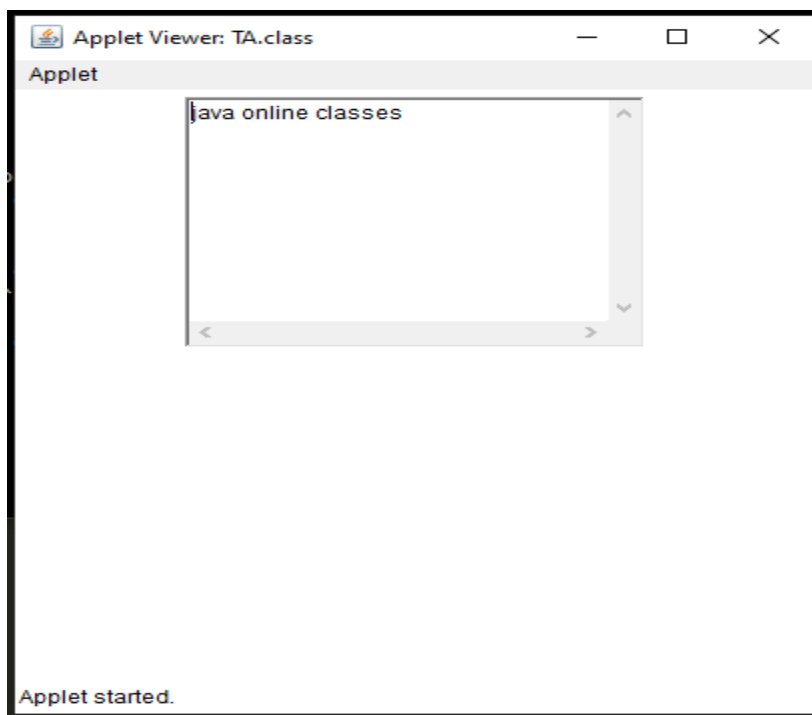
    public void init(){

        TextArea ta=new TextArea("javaonlineclasses",10,30);

        add(ta);

    }

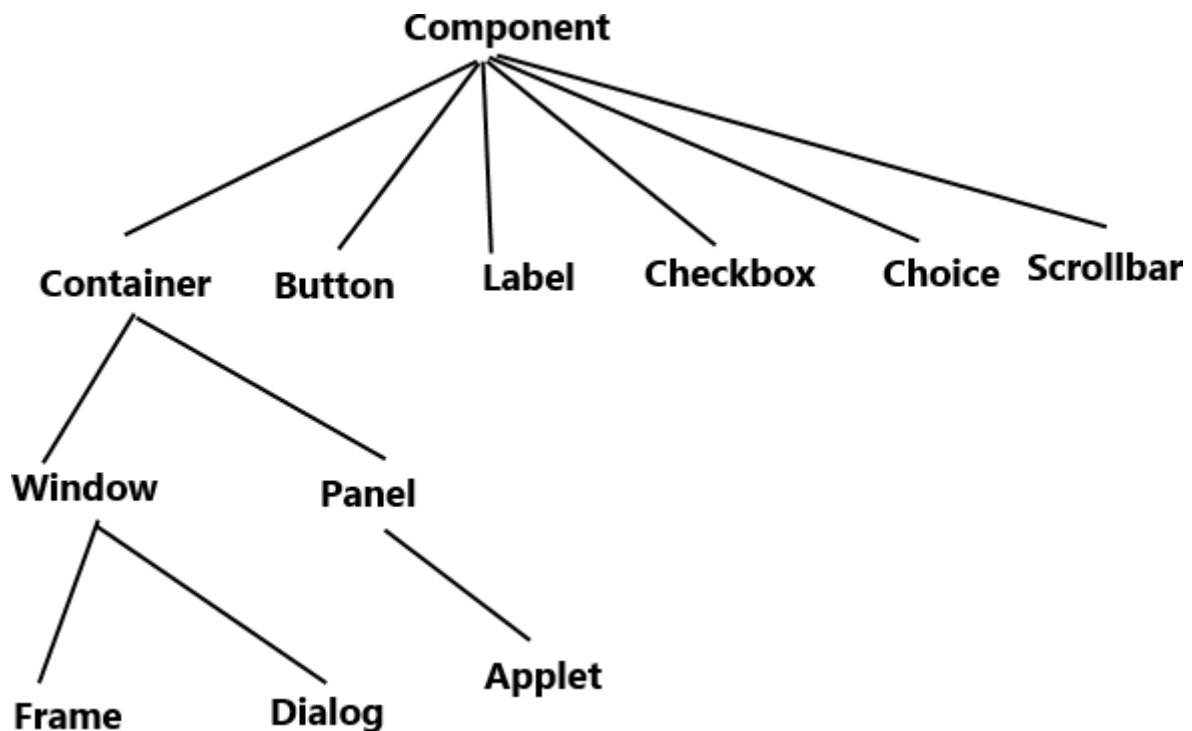
}
```



Object Oriented Programming Through Java(25CS303)

Source	Event	EventListener	Methods in EventListener
Applet	MouseEvent	MouseListener	mouseClicked() mousePressed() mouseReleased() mouseEntered() mouseExited()
		MouseMotionListener	mouseMoved() mouseDragged()
	KeyEvent	KeyListener	keyPressed() keyReleased() keyTyped()
Label	-	-	-
TextField	-	-	-
Button	ActionEvent	ActionListener	actionPerformed()
Choice List Checkbox CheckboxGroup	ItemEvent	ItemListener	itemStateChanged()
Scrollbar	AdjustmentEvent	AdjustmentListener	adjustmentValueChanged() ()
TextArea	-	-	-

Awtclasshierarchy



Container

The Container class is a subclass of Component. It has additional methods that allow other Component objects to be nested within it. Other Container objects can be stored inside of a Container (since they are themselves instances of Component). This makes for a multileveled containment system. A container is responsible for laying out (that is, positioning) any components that it contains. It does this through the use of various layout managers.

Panel

A Panel is a window that does not contain a title bar, menu bar, or border. This is why you don't see these items when an applet is run inside a browser. When you run an applet using an applet viewer, the applet viewer provides the title and border

Window

The Window class creates a top-level window. A top-level window is not contained within any other object; it sits directly on the desktop. Generally, you won't create Window objects directly. Instead, you will use a subclass of Window called Frame, described next.

Frame

Frame encapsulates what is commonly thought of as a “window.” It is a subclass of Window and has a **title bar, borders, and resizing corners**. If you create a Frame object from within an applet, it will contain a warning message, such as “Java Applet Window,” to the user that an applet window has been created. This message warns users that the window they see was started by an applet and not by software running on their computer.

Note: Frame does not have status bar and menu bar. Canvas

Although it is not part of the hierarchy for applet or frame windows, there is one other type of window that you will find valuable: Canvas. **Canvas encapsulates a blank window upon which you can draw.**

Constructors

Frame()

Frame(String title)

//Creating a Frame window by instantiating Frame class

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

public class FrameEx11 implements ActionListener {

    Frame fm;
    FrameEx11() {

        fm = new Frame("JFrame Example");

        //Creating a frame.

        Label lb = new Label("welcome to java Frame Window ");
        Button jb = new Button("close");

        fm.setLayout(new FlowLayout(FlowLayout.CENTER));
        fm.add(lb); //adding label to the frame. fm.add(jb);

        fm.setSize(300, 300); //setting frame size.
        fm.setVisible(true);

        jb.addActionListener(this);

    }

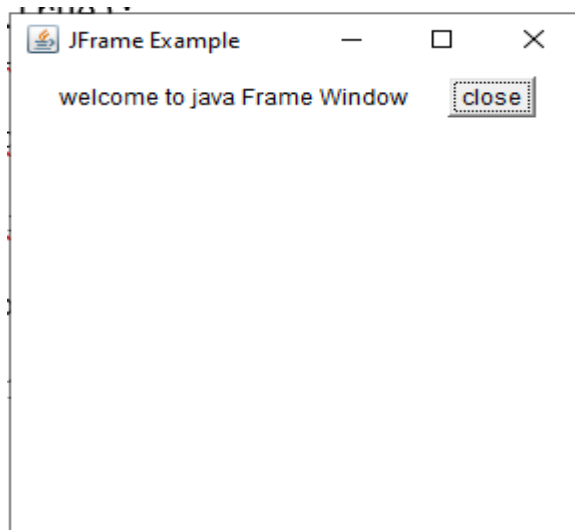
    public void actionPerformed(ActionEvent ae) {

        fm.setVisible(false); //to close the frame

    }

}
```

```
public static void main(String args[]){  
  
    FrameEx11ta=new FrameEx11();  
  
}  
  
}
```



Adapter classes

Java adapter classes provide the default implementation of listener interfaces. If we inherit the adapter class, we need not be forced to provide the implementation of all the methods of listener interfaces. So, it saves code.

The adapter classes are found in **java.awt.event** and **javax.swing.event** packages.

Note:1: In case of `addMouseListener(this)`, this refers to the current class object i.e., child class which extends Applet.

Note2: Interfaces which have more than one method have corresponding Adapter classes.

For Example: `MouseListener` – `MouseAdapter`

`MouseMotionListener` –

`MouseMotionAdapter` `KeyListener` – `KeyAdapter`

//Program on Adapter Classes

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

/*<appletcode="AdapterDemo.class"width=150height=200>
</applet>*/

public class AdapterDemo extends Applet{

    public void init(){

        addMouseListener(new MyMouseAdapter(this));
        //eventsourceobject
        //thisreferstothe current class object that is AdapterDemo
    }

}

class MyMouseAdapter extends MouseAdapter{

    AdapterDemo adapterDemo;

    public MyMouseAdapter(AdapterDemo adapterDemo){

        this.adapterDemo=adapterDemo;

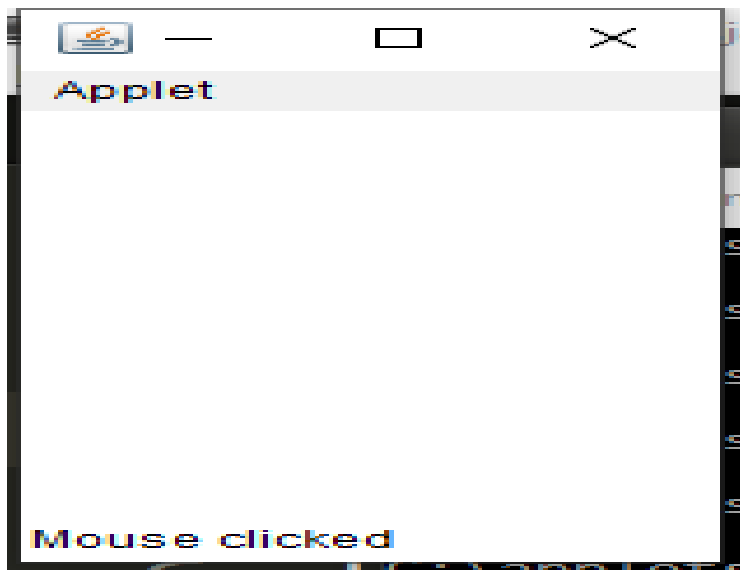
    }

    public void mouseClicked(MouseEvent me){

        adapterDemo.showStatus("Mouse clicked");

    }

}
```



LayoutManager

A **LayoutManager** is useful to **arrange components in a particular order in a frame or container**. Container can be Applet, Window, Panel or a Frame. Components can be a Label, Button, TextField and so on. A layout manager is an instance of any class that implements the **LayoutManager** interface. The following classes represent the layout managers and are available in the `java.awt` package.

1. `FlowLayout`
2. `BorderLayout`
3. `GridLayout`
4. `CardLayout`
5. `GridBagLayout`

Note: When we add AWT controls to an Applet window, the AWT controls are added in the center of the Applet. The default layout is `FlowLayout`.

1. FlowLayout

The `FlowLayout` is used to **arrange the components in a line, one after another** (in a flow). `FlowLayout` is the default layout. It implements a simple layout style, which is similar to how words flow in a text editor.

Constructor of FlowLayout class

FlowLayout()

FlowLayout(int how)

FlowLayout(int how, int howz, int vert)

//horzspacebetweenawtcontrols w.r.t two lines and vert refers to the space between awt controls in a line

The first constructor creates the default layout, which centers components and leaves five pixels of space between each component.

i.e., `FlowLayout(FlowLayout.CENTER, 5, 5)`

Note: `setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5))` is by default available for the applet window. When we add awt controls to the applet window, the default layout is `FlowLayout(FlowLayout.CENTER, 5, 5)`

The second constructor lets you specify how each line is aligned. Valid values for how are as follows:

public static final int LEFT

public static final int RIGHT

public static final int CENTER

public static final int LEADING

public static final int TRAILING

Note: LEADING means leaving some space in the beginning and TRAILING means leaving some space at the end.

//Program for FlowLayout

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

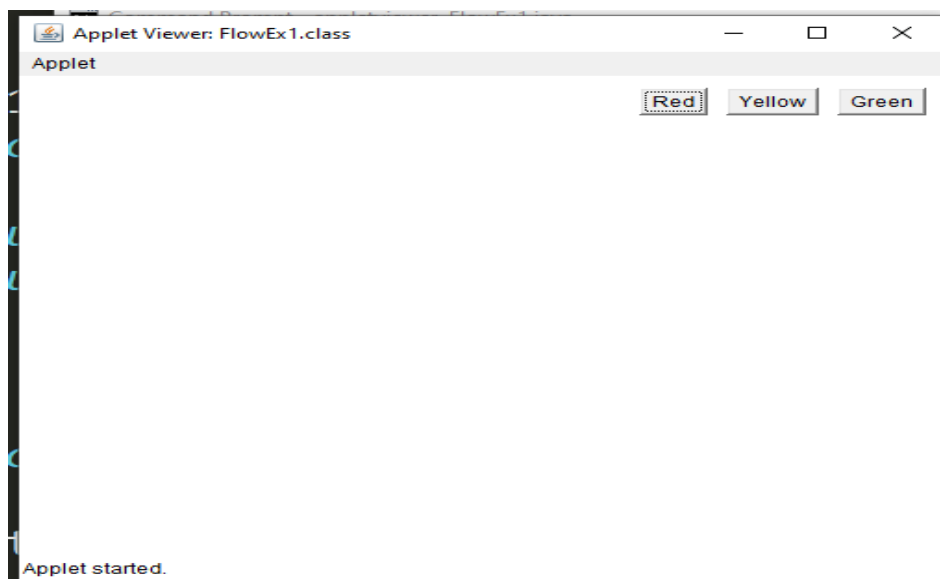
/*<applet code="FlowLayoutEx1.class" width=500 height=400>
</applet>*/

public class FlowLayoutEx1 extends Applet implements ActionListener {
    Button b1, b2, b3;
    public void init() {
        //FlowLayout(int how, int howz, int vert)
        FlowLayout ob1 = new FlowLayout(FlowLayout.RIGHT, 10, 10);
        setLayout(ob1); //the FlowLayout is set to the Applet window

        b1 = new Button("Red");
        b2 = new Button("Yellow");
        b3 = new Button("Green");
    }
}
```

```
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);

add(b1);
add(b2);
add(b3);
}
public void actionPerformed(ActionEvent ae){
    if(ae.getSource()==b1){
        setBackground(Color.red);
    }
    elseif(ae.getSource()==b2){
        setBackground(Color.yellow);
    }
    else{
        setBackground(Color.green);
    }
}
}
```



2. BorderLayout

The BorderLayout class implements a common layout style for top-level windows. It has four narrow, fixed-width components at the edges and one large area in the center. The four sides

Object Oriented Programming Through Java(25CS303)

are referred to as north, south, east and west. The middle area is called the center.

Constructors:

BorderLayout()

BorderLayout(int horz, int vert)

The first form creates a default border layout. The second form allows you to specify the horizontal and vertical space left between components in *horz* and *vert* respectively.

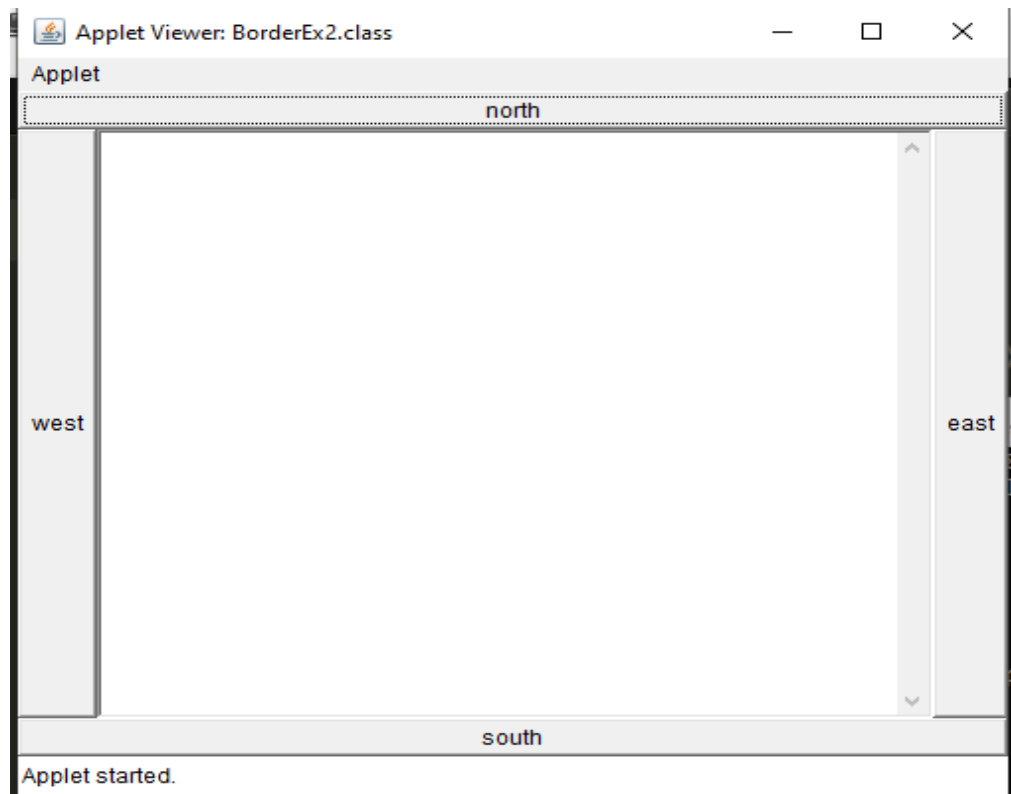
BorderLayout defines the following constants (public, static, final) that specify the regions:

```
BorderLayout.NORTH  
BorderLayout.SOUTH  
BorderLayout.EAST  
BorderLayout.WEST  
BorderLayout.CENTER
```

//Program for BorderLayout

```
import java.applet.*;  
import java.awt.*;  
import java.awt.event.*;  
/*<applet code="BorderLayoutEx2.class" width=500 height=400>  
</applet>*/  
public class BorderLayoutEx2 extends Applet implements ActionListener {  
    TextArea ta;  
    Button nb, sb, eb, wb; public  
    void init() {  
        ta = new TextArea();  
        nb = new Button("north");  
        wb = new Button("west");  
        sb = new Button("south");  
        eb = new Button("east");  
  
        setLayout(new BorderLayout());  
        nb.addActionListener(this);  
        eb.addActionListener(this);  
        sb.addActionListener(this);  
        wb.addActionListener(this);  
  
        add(nb, BorderLayout.NORTH);  
        add(wb, BorderLayout.WEST);  
        add(sb, BorderLayout.SOUTH);
```

```
add(eb, BorderLayout.EAST);
add(ta, BorderLayout.CENTER);
}
public void actionPerformed(ActionEvent ae){
    ta.setText(ae.getActionCommand());
}
}
```



3. GridLayout

GridLayout lays out components in a two-dimensional grid.

The GridLayout is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

GridLayout(): creates a grid layout with one column per component in a row.

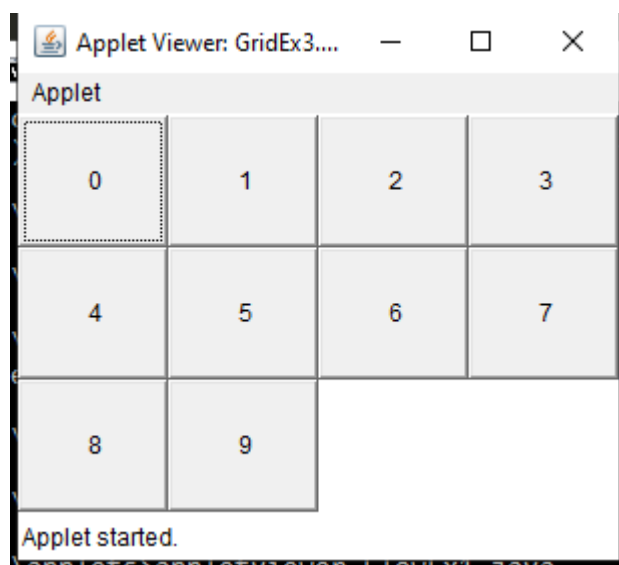
Object Oriented Programming Through Java(25CS303)

GridLayout(int rows, int columns): creates a grid layout with the given rows and columns but no gaps between the components.

GridLayout(int rows, int columns, int hgap, int vgap): creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

//Program for GridLayout

```
import java.awt.*;
import java.applet.*;
/*<appletcode="GridLayoutEx3.class"width=300height=200>
</applet>*/
public class GridLayoutEx3 extends Applet{
    public void init(){
        setLayout(new GridLayout(3,4));
        for(int i=0;i<10;i++){
            add(new Button(""+i));
        }
    }
}
```



4. Card Layout

Constructors:

CardLayout()CardLayout(int

horz,intvert) **Methods:**

voidadd(ComponentpanelObj,StringcardName)

Note:cardNameisnotvisible. EachcardisonePanel. These cardNamesareusefulinselecting the random cards using the show() method.

voidfirst(Containerdeck)

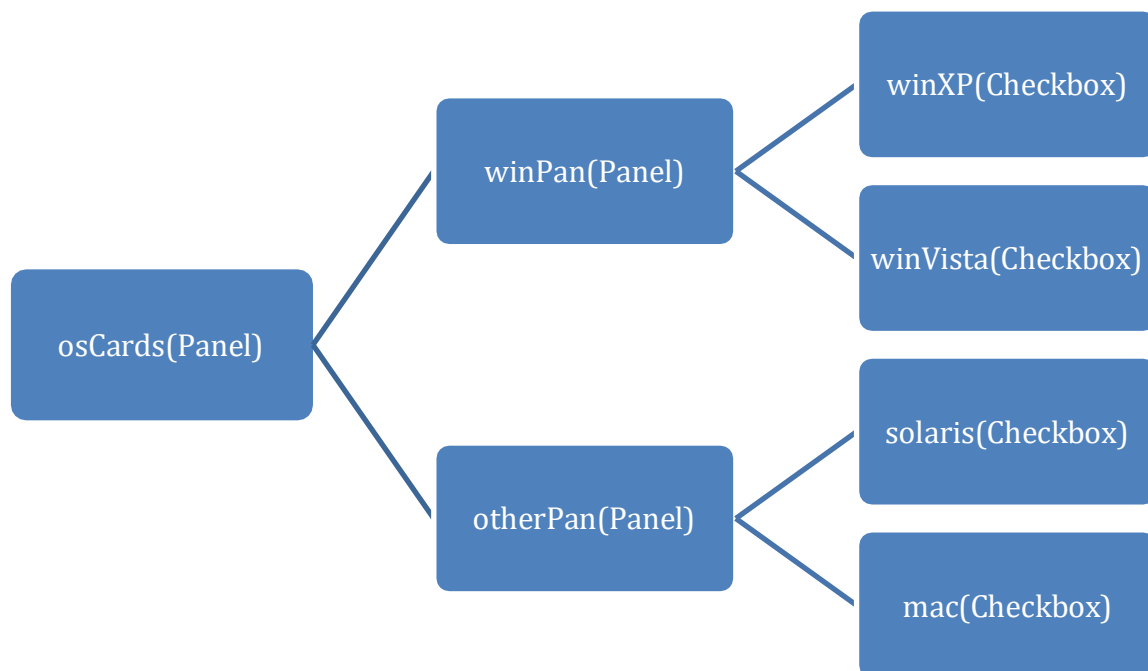
void last(Container deck)

voidnext(Container deck)

voidprevious(Containerdeck)

voidshow(Containerdeck,StringcardName)//toselecttherandomcardwithcardName

Note: In the next Program we are creating the following structure. osCards is a container deck whichholds 2cardswinPanandotherPan.winPanis aPanelwhichholds 2 checkboxes winXP and winVista. otherPan is aPanel which holds 2 checkboxes solaris and mac.



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

/*<appletcode="CardEx3.class"width=300height=100>
</applet>*/

public class CardEx3 extends Applet implements MouseListener{
    Checkbox winXP, winVista, solaris, mac;
    Panel osCards;
    //Panel definition
    CardLayout cardLO;
    public void init(){
        cardLO = new CardLayout();

        osCards = new Panel();
        osCards.setLayout(cardLO); //set card layout to card deck panel

        winXP = new Checkbox("Windows XP", null, true);
        winVista = new Checkbox("Windows Vista");
        solaris = new Checkbox("Solaris");
        mac = new Checkbox("Mac OS");

        //creating two panels
        Panel winPan = new Panel();
        //add winXP and winVista checkboxes to a panel winPan.add(winXP);
        winPan.add(winVista);

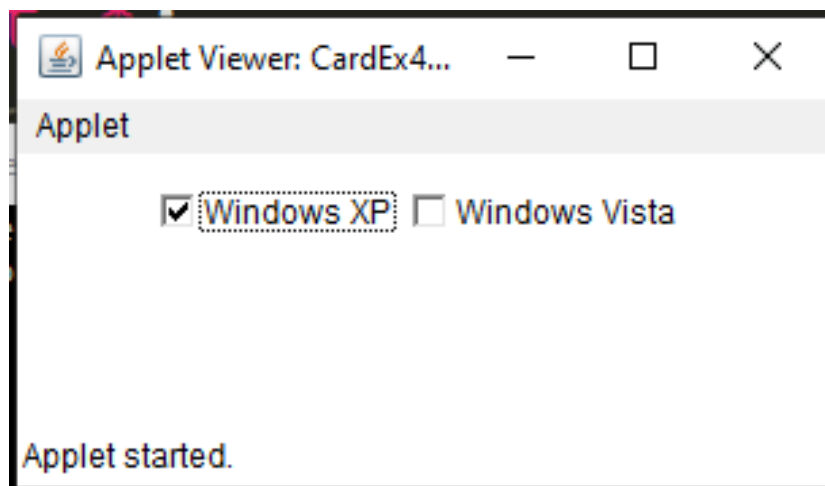
        Panel otherPan = new Panel();
        //add solaris and mac checkboxes to a panel otherPan.add(solaris);
        otherPan.add(mac);

        // add panels to card deck panel
        osCards.add(winPan, "Windows");
        osCards.add(otherPan, "Other");

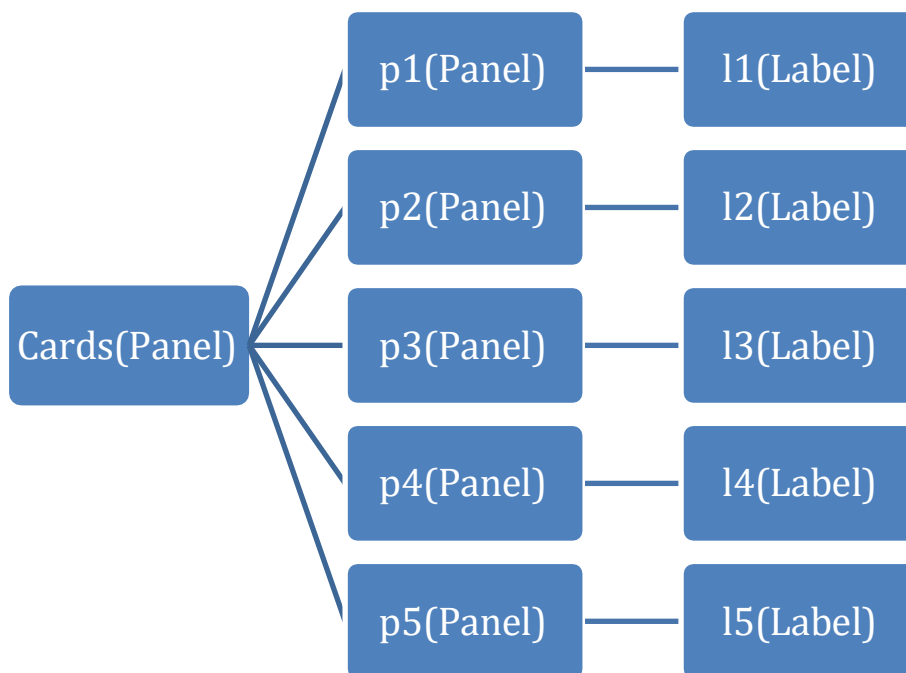
        //add card deck to main applet window add(osCards);

        // register mouse events
        addMouseListener(this);
    }
    //cycle through panels
    public void mousePressed(MouseEvent me){
        cardLO.next(osCards);
    }
    //Provide empty implementations for the other MouseListener methods.
```

```
public void mouseClicked(MouseEventme){  
    }  
public void mouseEntered(MouseEventme){  
    }  
public void mouseExited(MouseEventme){  
    }  
public void mouseReleased(MouseEventme){  
    }  
}
```



Program2:ImplementthefollowingstructureusingCardLayout



```
import java.awt.event.*;
import java.awt.*;
import java.applet.*;
/*<appletcode="CardEx5.class"width=300height=100>
</applet>*/
public class CardEx5 extends Applet implements MouseListener{
    Label l1,l2,l3,l4,l5;
    Panel Cards;//Paneldefinition
    CardLayout cardLO;
    public void init(){
        cardLO=new CardLayout();
        Cards = new Panel();
        Cards.setLayout(cardLO);//setcardlayou ttocarddeckPanel
        l1=new Label("FirstCard");
        l2=new Label("SecondCard"); l3 =
        new Label("Third Card"); l4 =new
        Label("Fourth Card");
        l5 = new Label("Fifth Card");

        // creating two Panels Panel
        p1=new Panel();
        p1.add(l1);
        Panel p2=new Panel(); p2.add(l2);
        Panel p3=new Panel(); p3.add(l3);
        Panel p4=new Panel(); p4.add(l4);
```

```
Panel p5=new Panel(); p5.add(l5);

//addpanelstocarddeckpanel
Cards.add(p1, "First");
Cards.add(p2,"Second");
Cards.add(p3,"Third");
Cards.add(p4,"Fourth");
Cards.add(p5,"Fifth");

//add carddeckto main appletwindow
add(Cards);//register mouse events
addMouseListener(this);
}

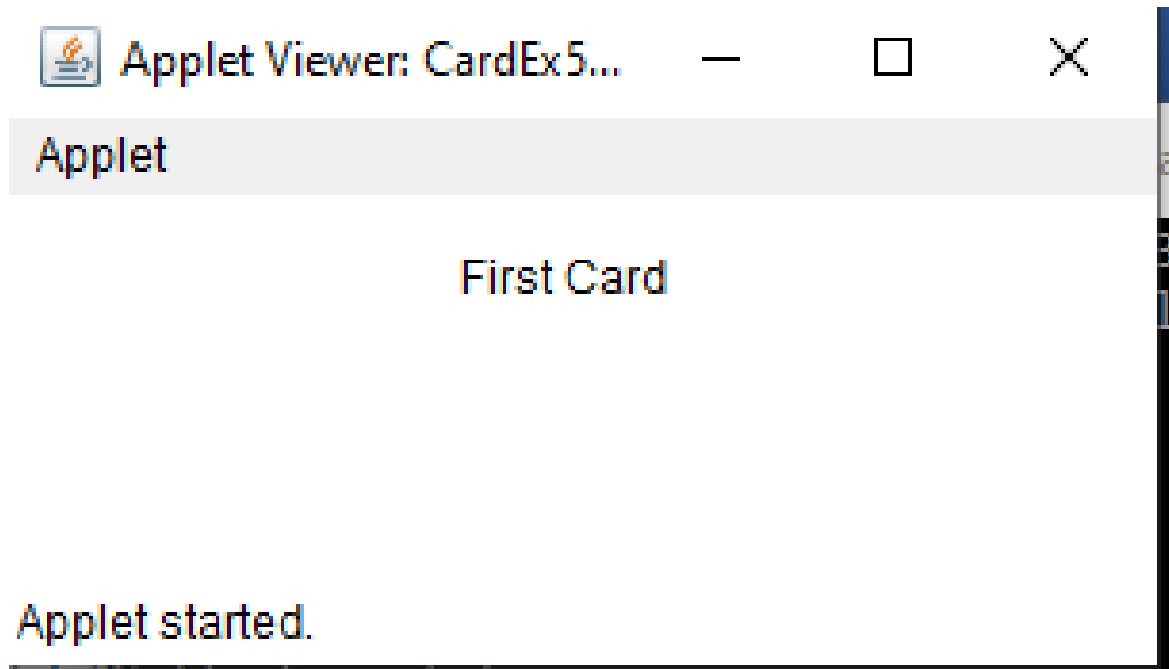
//cycle through panels
public void mousePressed(MouseEvent me){
}

//Provide empty implementations for the other MouseListener methods.
public void mouseClicked(MouseEvent me){
    cardLO.next(Cards);
}

public void mouseEntered(MouseEvent me){
    cardLO.last(Cards);
}

public void mouseExited(MouseEvent me){
    cardLO.show(Cards,"Third");
}

public void mouseReleased(MouseEvent me){
}
}
```



5. GridBagLayout:

The key to the GridBag is that each component can be a different size, and each row in the grid can have a different no. of columns. That is why the layout is called a Grid Bag. It's a collection of small grids joined together.

GridBagLayout defines only one **Constructor**:

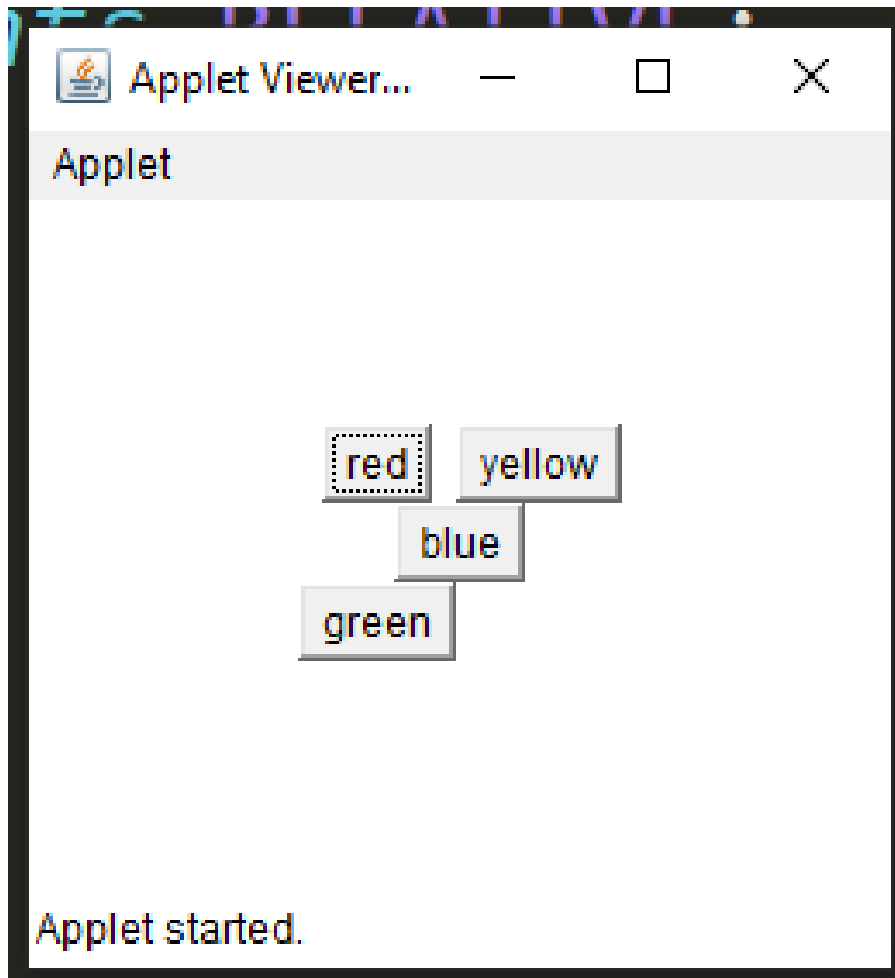
`GridBagLayout()`

Methods:

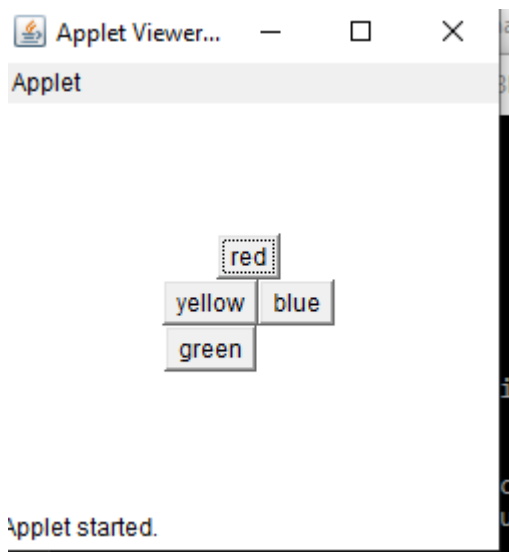
`void setConstraints(Component comp, GridBagConstraints cons)`

//DemonstrationofGridBagLayout

```
import java.awt.event.*;
import java.awt.*;
import java.applet.*;
/*<appletcode="GBLEx6.class"width=250height=200>
</applet>*/
public class GBLEx6 extends Applet{
    Button b1,b2,b3,b4;
public void init(){
    GridBagLayout gbag=new GridBagLayout();
    GridBagConstraints gbc=new GridBagConstraints();
    setLayout(gbag);
    b1=new Button("red");
    b2=new Button("yellow");
    b3=new Button("blue");
    b4=new
    Button("green");gbc.gridwidth=GridBagConstraints.
    RELATIVE;
    //RELATIVE means awtcontrolwilltakethespace required for itinarow
    gbag.setConstraints(b1,gbc);
    add(b1);
    gbc.gridwidth=GridBagConstraints.REMAINDER;
    //REMAINDERmeansawtcontrolwilloccupytheremainingspaceinarow
    gbag.setConstraints(b2,gbc);
    add(b2);
    //thegridwidthconstraintisnotupdatedforthebuttonb3sothegridwidthconstraint
    ofb2 is applicable for
    b3. gbag.setConstraints(b3,gbc);
    add(b3);
    gbc.gridwidth=GridBagConstraints.RELATIVE;
    gbag.setConstraints(b4,gbc);
    add(b4);
    }
}
```



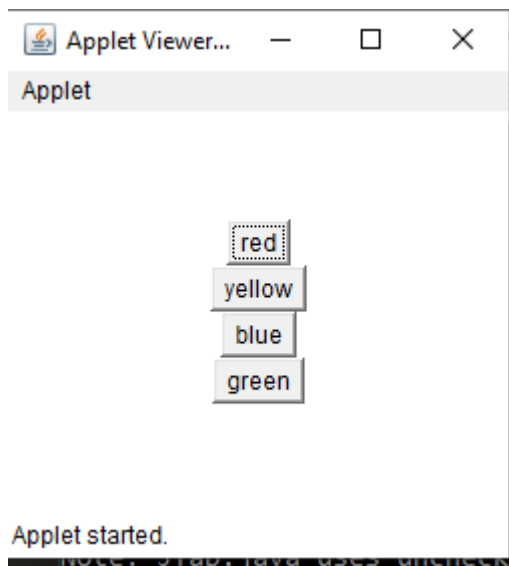
Program1: Write an applet program to obtain the following output using GridbagLayout



Hint: set gridwidth for red remainder, yellow relative, blue remainder and for green relative.

Object Oriented Programming Through Java(25CS303)

Program2: Write an applet program to obtain the following output using GridBagLayout



Hint: if we want to arrange one button in one line then set grid width to remainder for red button. For the remaining buttons if we will not change grid width then remainder will be carried.

Swings

Java Swing is used to create window-based applications or graphical user interface. It is built on the top of AWT (Abstract Windowing Toolkit) **API and entirely written in java**. Unlike AWT, Java Swing provides platform-independent and lightweight components. It is built on top of AWT API and acts as a replacement of AWT API, since it has almost every control corresponding to AWT controls. **Swing component** follows a **Model-View-Controller** architecture, where **model** represents data, **view** represents presentation and **controller** acts as an interface between model and view.

AWT controls like Button, TextField and so on are developed in different languages that are not platform independent. Swing components are developed using Java, so they are written as JButton, JTextField and so on.

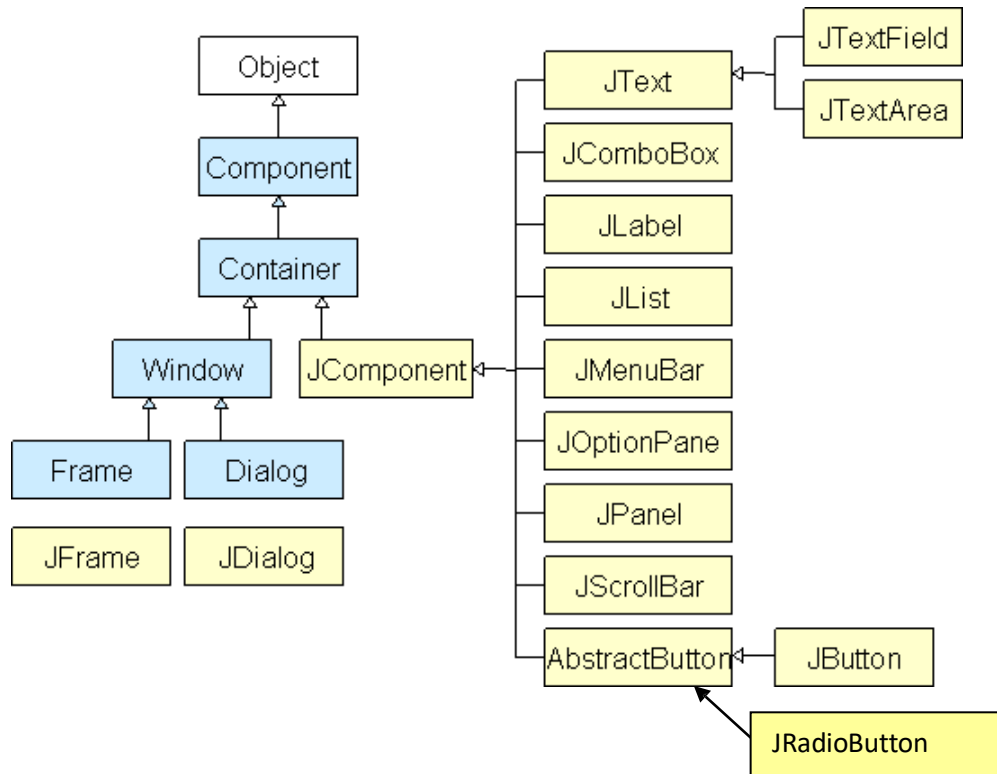
The `javax.swing` package provides classes for Java Swing APIs such as JButton, JTextField, JTextArea, JRadioButton, JCheckBox, JMenu, etc.

Difference between AWT and Swing:-

No.	Java AWT	Java Swing
1)	AWT components are platform-dependent	Java Swing components are platform-independent .
2)	AWT components are heavyweight .	Swing components are lightweight .
3)	AWT doesn't support pluggable look and feel .	Swing supports pluggable look and feel .
4)	AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scroll panes, color chooser, tabbed pane etc.
5)	AWT doesn't follow MVC (Model View Controller).	Swing follows MVC .

ExploringSwing:

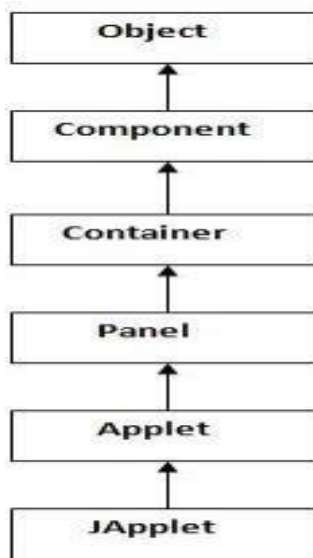
The hierarchy of JAWASWING class



JApplet:-

JApplet is a class which extends **Applet** class. **JApplet** class has all the controls of swing. The **JApplet** class extends the **Applet** class. The `javax.swing.JApplet` class defines the core functionality of an **applet**. (`java.awt.Applet` is the older, AWT- based form)

Hierarchy of Applet



JComponent

The JComponent class is the base class of all Swing components except top-level containers. Swing components whose names begin with "J" are descendants of the JComponent class. For example, JButton, JScrollPane, JPanel, JTable etc. But, JFrame and JDialog don't inherit JComponent class because they are the child of top-level containers.

The JComponent class extends the Container class which itself extends Component. The Container class has support for adding components to the container.

Constructor:

JComponent()

//JavaSwingProgram

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

/*<appletcode="SwingDemo.class"height=400width=400></applet>*/

public class SwingDemo extends JApplet implements ActionListener {

    JButton b1;
```

```
JLabel
l1;JTextField t1,t2;

public voidinit(){

    // These two statements are to be added for all swing programs

    Container c=getContentPane();

    c.setLayout(new FlowLayout());// here FlowLayout is not default layout

    b1=newJButton("Factorial");
    t1=new JTextField("",5);
    t2=new JTextField("",5);
    l1=newJLabel("enteranumber");

    // instead of adding awt controls to Applet Window in Applet programs,
    //for swing programs we have to add to container

    c.add(l1);
    c.add(t1);
    c.add(b1);
    c.add(t2);

    b1.addActionListener(this);

}

public voidactionPerformed(ActionEventae){

Stringstr=t1.getText();

intnum=Integer.parseInt(str);

int fact=1;

for(inti=1;i<=num;i++){

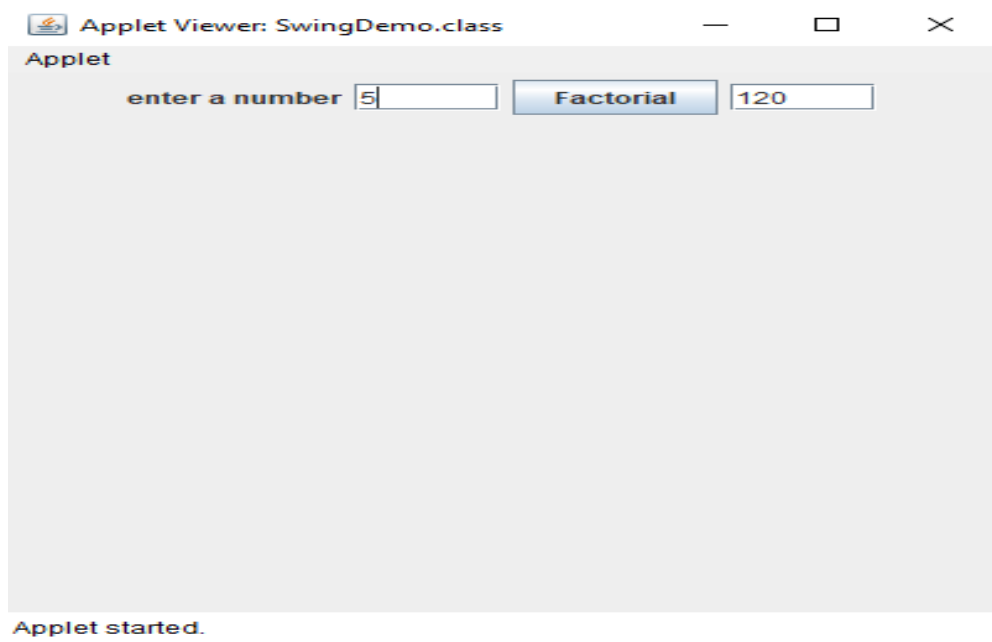
    act=fact*i;

}

t2.setText(fact+"");

}

}
```



2. RadioButton:

Constructor:

JRadioButton(Stringstr)

Method:

voidadd(AbstractButtonab)

Note: After creating the JRadioButton we need to group these buttons. For this we use ButtonGroup class

ButtonGroup()

//Program on Radiobuttons using javax.swing

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

/*<appletcode="JRadioButtonDemo.class"height=300width=300>
</applet>*/

public class JRadioButtonDemo extends JApplet implements ActionListener {
    JLabel jl;
```

```
public void init(){
    Container c=getContentPane();
    c.setLayout(new FlowLayout());

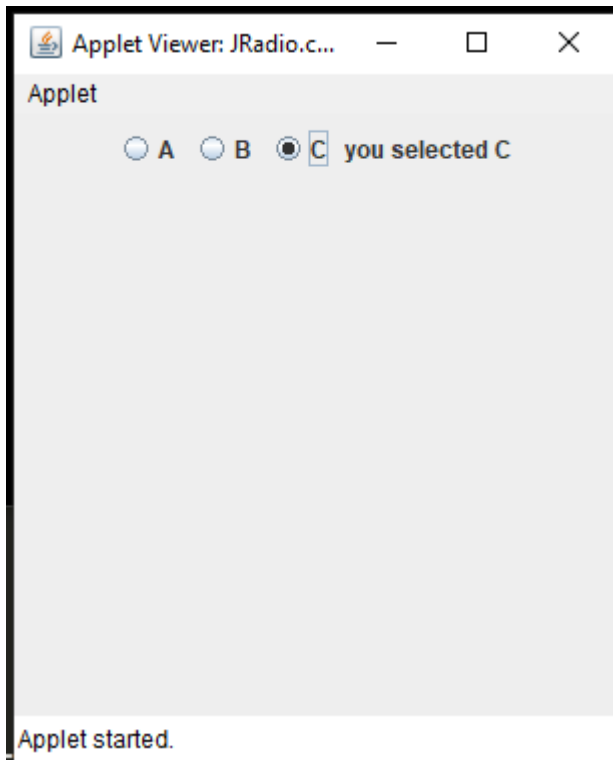
    JRadioButton b1=new JRadioButton("A");
    JRadioButton b2=new JRadioButton("B");
    JRadioButton b3=new JRadioButton("C");

    ButtonGroup bg=new ButtonGroup(); //adding JRadioButtons to ButtonGroup

    bg.add(b1);
    bg.add(b2);
    bg.add(b3);

    b1.addActionListener(this);
    b2.addActionListener(this);
    b3.addActionListener(this);

    c.add(b1);
    c.add(b2);
    c.add(b3);
    jl=new JLabel("select one");
    c.add(jl);
}
public void actionPerformed(ActionEvent ae){
    jl.setText("you selected "+ae.getActionCommand());
}
}
```



3. JComboBox://inAppletsitisChoice

JComboBox:

Swing provides a combobox through the JComboBox class. A combobox normally displays one entry, but it will also display a drop-down list that allows a user to select a different entry.

Constructor:

JComboBox(Object[] items)

Object Oriented Programming Through Java(25CS303)

Methods:

void addItem(Object obj)

Object getSelectedItem()

//Demonstration on JComboBox

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

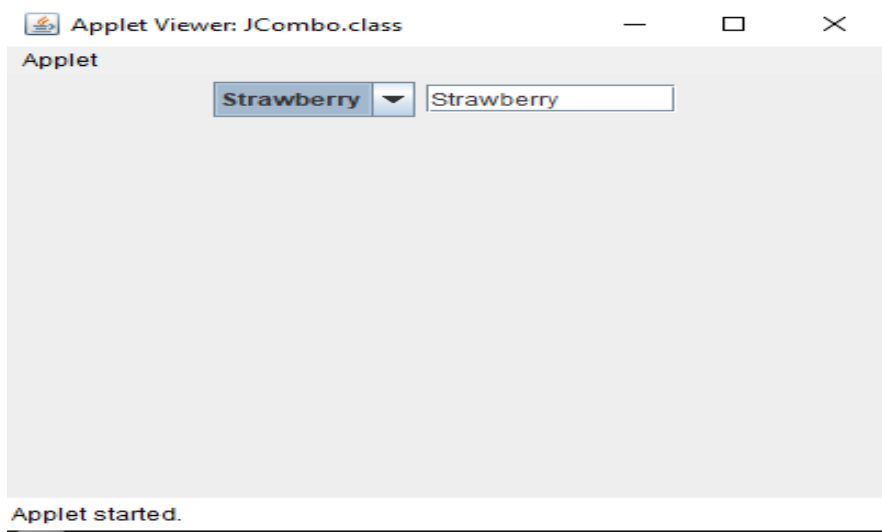
/*<appletcode="JCombo.class"height=300width=400></applet>*/
public class JCombo extends JApplet implements ItemListener {
    JComboBox jcb;
    JTextField tf;

    public void init() {
        Container c = getContentPane();
        c.setLayout(new FlowLayout());

        jcb = new JComboBox();
        jcb.addItem("Vanilla");
        jcb.addItem("Chocolate");
        jcb.addItem("Strawberry");

        c.add(jcb);
        tf = new JTextField("", 10);
        c.add(tf);
        jcb.addItemListener(this);
    }

    public void itemStateChanged(ItemEvent ie) {
        tf.setText(ie.getItem() + "");
    }
}
```



4. JTabbedPane:

JTabbedPane encapsulates a tabbed pane. JTabbedPane defines by using below constructor.

JTabbedPane()

Methods:

void addTab(String name, Component comp)

Procedure:

The general procedure to use a tabbed pane is outlined here:

- 1) create an instance of JTabbedPane.
- 2) Add each tab by calling addTab()
- 3) Add the tabbed pane to the content pane.

//Demonstration of JTabbedPane

```
import javax.swing.*;
import java.awt.*;

/*<appletcode="JTabbedPaneDemo.class"height=300width=400>
</applet>*/

public class JTabbedPaneDemo extends JApplet {
    public void init() {
        Container c = getContentPane();
```

Object Oriented Programming Through Java(25CS303)

```
c.setLayout(new FlowLayout());
JTabbedPanejtp=new JTabbedPane();

jtp.addTab("Cities", new CitiesPanel());
//whentheaddTab()methodisinvokeditgoes toCitiesPanel()constructor
//andtheCities NewYork,London,so onareaddedto theCities Tab jtp.addTab("Colors",
new ColorsPanel());
jtp.addTab("Flavors",newFlavorsPanel());

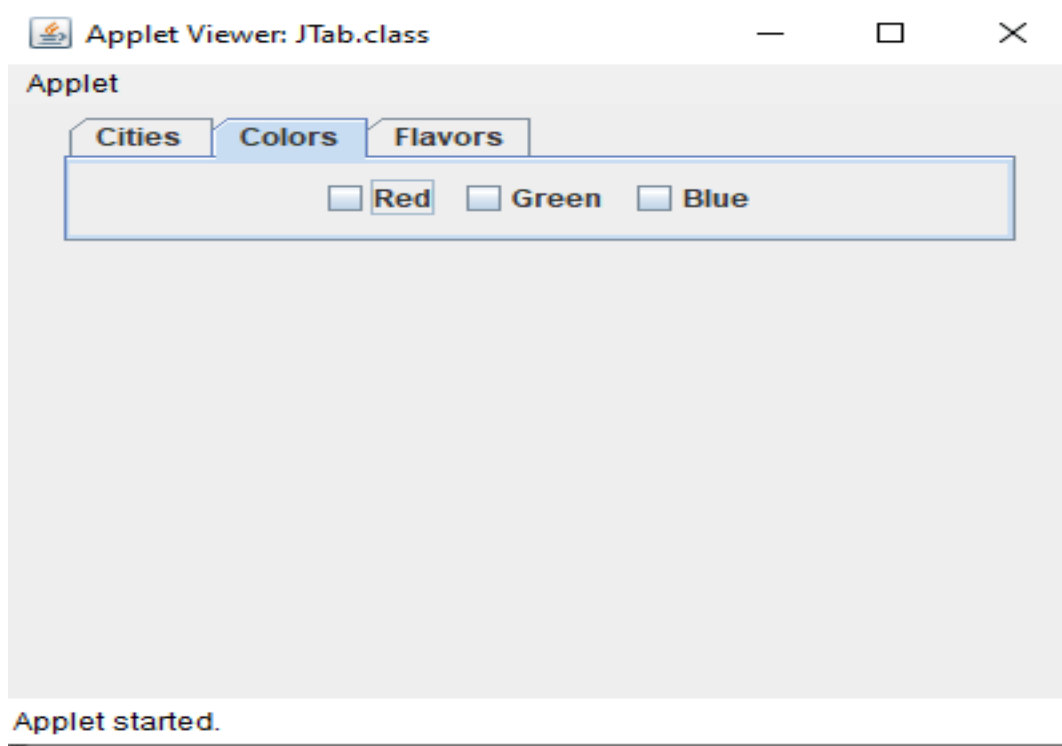
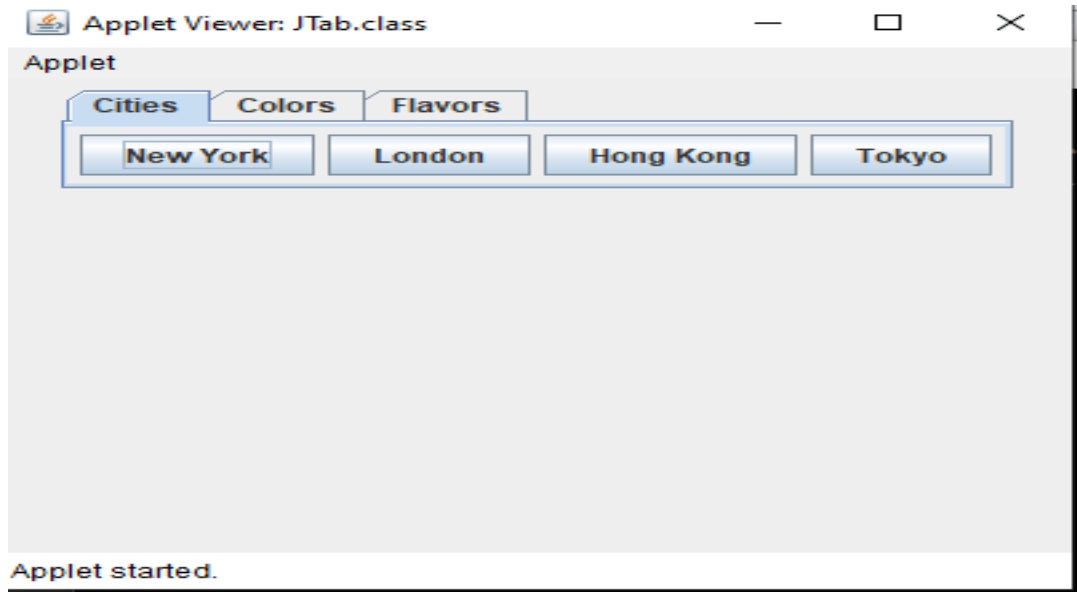
c.add(jtp);
}
}
classCitiesPanelextendsJPanel{
publicCitiesPanel(){
    JButtonb1=newJButton("NewYork");
    add(b1);
    JButtonb2=newJButton("London");
    add(b2);
    JButtonb3=newJButton("HongKong");
    add(b3);
    JButtonb4=newJButton("Tokyo");
    add(b4);
}
}
classColorsPanelextendsJPanel{
publicColorsPanel(){
    JCheckBox cb1=new JCheckBox("Red");
    JCheckBoxcb2=newJCheckBox("Green");
    JCheckBox cb3=new JCheckBox("Blue");

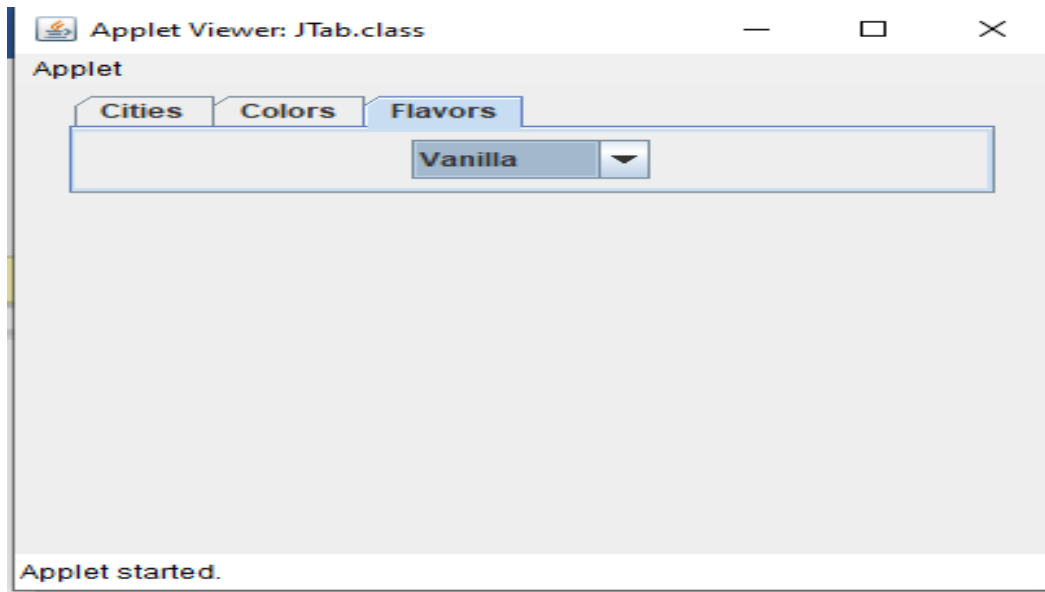
    add(cb1);
    add(cb2);
    add(cb3);
}
}
classFlavorsPanelextendsJPanel{
publicFlavorsPanel(){
    JComboBoxjcb=newJComboBox();

    jcb.addItem("Vanilla");
    jcb.addItem("Choclote");
    jcb.addItem("Strawberry");
    add(jcb);
}
```

Object Oriented Programming Through Java(25CS303)

```
}
```





JScrollPane:

Constructor:

`JScrollPane(Component comp)`

Procedure

1. Create the component to be scrolled. (ex: -Panel)
2. Create an instance of `JScrollPane`, pass the component (Panel) to be scrolled.
3. Add the scroll pane to the content pane (Container).

//Demonstration of JScrollPane

```
import javax.swing.*;
import java.awt.*;

/*<appletcode="JScrollPaneDemo.class"height=250 width=300>
</applet>*/

public class JScrollPaneDemo extends JApplet { //Creating an Applet

    public void init() {

        Container c = getContentPane();

        c.setLayout(new BorderLayout());

        JPanel jp = new JPanel();
```

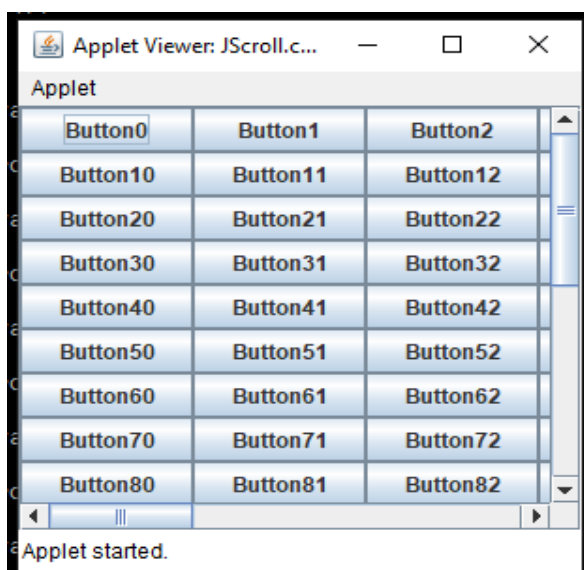
Object Oriented Programming Through Java(25CS303)

```
//1. Create the component to be scrolled. jp.setLayout(new GridLayout(20,20));  
  
for(int i=0;i<200;i++){  
  
    jp.add(new JButton("Button"+i));  
  
}  
  
//2. Create an instance of JScrollPane, pass the component to be scrolled.  
  
JScrollPane jsp=new JScrollPane(jp);  
  
//3. Add the scroll pane to the content pane.  
  
c.add(jsp, BorderLayout.CENTER);  
  
}  
  
}
```

Note 1: Create the JPanel add the Buttons to the JPanel. Add the JScrollPane to the JPanel. Then add the JScrollPane to the ContentPane.

Note 2: If you add less number of Buttons to the JPanel then the Scroll Bars are not visible even if you create JScrollPane. So, add more number of Buttons to visualize the JScrollPane.

Note 3: Vertical ScrollBar is automatically created if we add more number of rows and horizontal ScrollBar is automatically created if we add more number of columns



JTree

A tree is a component that presents a hierarchical view of data. Trees are implemented in swing by the JTree class.

Constructors:

JTree(TreeNode tn)

The TreeNode interface declares methods that obtain information about a tree node.

The mutable TreeNode interface extends TreeNode. It declares methods that can insert and remove child nodes or change the parent node. The DefaultMutableTreeNode class implements the MutableTreeNode interface. It represents a node in a tree.

One of its constructors is shown here: **DefaultMutableTreeNode(Object obj)**

To create a hierarchy of tree nodes, the add() method of DefaultMutableTreeNode can be used.

Its signature is shown here:

void add(MutableTreeNode child)

Here are the steps to follow to use JTree.

1. Create an instance of JTree.
2. Create JScrollPane and pass the tree as the object to be scrolled. (Add JScrollPane to the Tree)
3. Add the JScrollPane to the content pane.

//Demonstration on Trees

```
import javax.swing.*;
import java.awt.*;
import javax.swing.tree.*;

/*<applet code="JTreeDemo.class" height=200 width=400>
</applet>*/

public class JTreeDemo extends JApplet {
    public void init() {
        Container c = getContentPane();
        c.setLayout(new BorderLayout());
    }
}
```

```
DefaultMutableTreeNode top=new DefaultMutableTreeNode("ROOT
NODE");
DefaultMutableTreeNode a=new DefaultMutableTreeNode("A");
top.add(a);
DefaultMutableTreeNode a1=new DefaultMutableTreeNode("A1");
DefaultMutableTreeNode a2=new DefaultMutableTreeNode("A2");

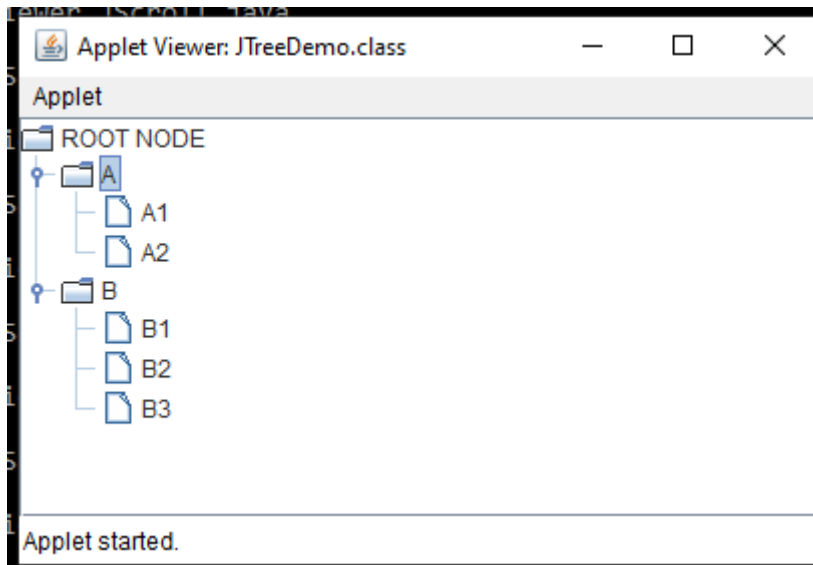
a.add(a1);
a.add(a2);
DefaultMutableTreeNode b=new DefaultMutableTreeNode("B");
top.add(b);
DefaultMutableTreeNode b1=new DefaultMutableTreeNode("B1");
DefaultMutableTreeNode b2=new DefaultMutableTreeNode("B2");
DefaultMutableTreeNode b3=new DefaultMutableTreeNode("B3");
b.add(b1);
b.add(b2);
b.add(b3);

//add the root node to the JTree

JTree tree=new JTree(top); // 1. Create an instance of JTree.

JScrollPane jsp=new JScrollPane(tree); // 2. Add scroll pane to tree

c.add(jsp, BorderLayout.NORTH); // 3. adding scroll pane to container
}
}
```



Note: If you create more number of nodes then the JScrollPane is visible

JTable

JTable is a component that displays rows and columns of data. JTable supplies several constructors.

Constructor:

`JTable(Object data[][], Object colHead[])`

Here are the steps required to set up a simple JTable that can be used to display the data.

1. Create an instance of JTable.
2. Create JScrollPane and pass the table as the object to be scrolled. (Add the table to the scrollpane)
3. Add the scrollpane to the content pane.

//Demonstration on JTable

```
import javax.swing.*;
import java.awt.*;

/*<appletcode="JTableDemo.class" height=200width=400></applet>*/

public class JTableDemo extends JApplet {

    public void init() {

        Container c = getContentPane();
        c.setLayout(new BorderLayout());
    }
}
```

```
String[] colHeads={"Name","ID#","Course"};

Object[][] data={

    {"a","1","Java"},

    {"b","2","C"},

    {"c","3","MachineLearning"},

    {"d","4","CD"},

    {"e","5","DWDm"},

    {"f","6","Java"},

    {"g","7","DBMS"},

    {"h","8","CO"},

    {"i","9","Deep Learning"},

    {"j","10","OS"},

    {"k","11","CO"},

    {"l","12","FLAT"},

    {"m","13","DAA"}

};

JTable table=new JTable(data,colHeads);//1.Create an instance of JTable.

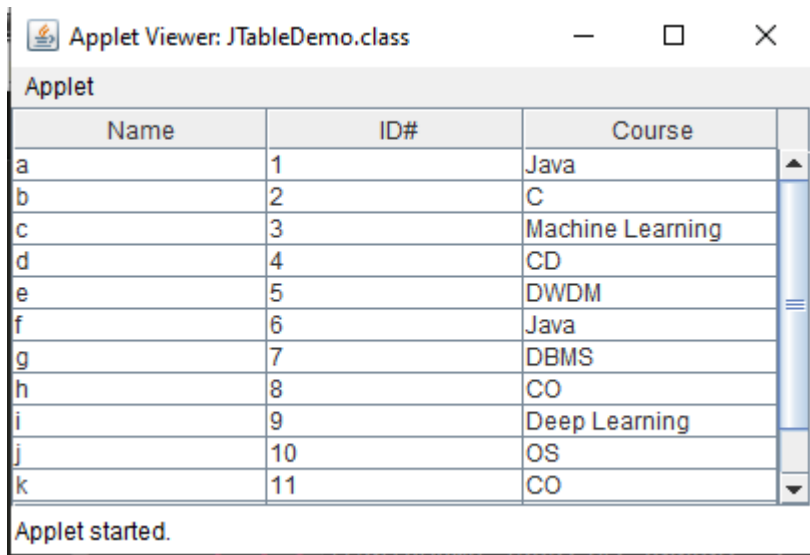
JScrollPane jsp=new JScrollPane(table);//2.Add the table to the scrollpane

c.add(jsp,BorderLayout.CENTER);//3. Add the scrollpane to the content pane.

}

}
```

Object Oriented Programming Through Java(25CS303)



Applet	JApplet
Label	JLabel
Button	JButton
TextField	JTextField
TextArea	JTextArea
Checkbox	JCheckbox
CheckboxGroup	JButtonGroup
Choice	JCombobox
List	JList
	JTabbedPane
	JScrollPane
	JTable
	JTree

JFrame

The `javax.swing.JFrame` class is a type of container which inherits the `java.awt.Frame` class. `JFrame` works like the main window where components like labels, buttons, textfields are added to create a GUI.

Unlike `Frame`, `JFrame` has the option to hide or close the window with the help of `setDefaultCloseOperation(int)` method.

Object Oriented Programming Through Java(25CS303)

Constructors

Constructor	Description
JFrame()	It constructs a new frame that is initially invisible.
JFrame(GraphicsConfiguration gc)	It creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.
JFrame(String title)	It creates a new, initially invisible Frame with the specified title.
JFrame(String title, GraphicsConfiguration gc)	It creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

Note: Without creating an applet tag we can create a window which is a Frame.

JFrame

```
import java.awt.*;
import javax.swing.*;

public class FrameEx11 {
    FrameEx11() {
        JFrame fm = new JFrame("JFrameExample");
        // Creating a frame with Title.

        JLabel lb = new JLabel("welcome to java Frame Window ");
        // Creating a label

        fm.setLayout(new FlowLayout(FlowLayout.CENTER));
        fm.add(lb); // adding label to the frame.
        fm.setSize(300, 300); // setting frame size.
    }
}
```

Object Oriented Programming Through Java(25CS303)

```
fm.setVisible(true);//set frame visibilty true.  
  
fm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    //For closing the Frame  
}  
publicstaticvoidmain(String args[]){  
    FrameEx11ta=new FrameEx11();  
}  
}
```

Compilation:javacFrameEx11.java

Exectuion: java FrameEx11

