Layout Manager

A **Layout** Manager 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 represents the layout managers and available in java.awt package.

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

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

1. Flow Layout

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

Constructors of FlowLayout class

FlowLayout()

FlowLayout(int how)

FlowLayout(int how, int horz, int vert)
// horz space between awt controls 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

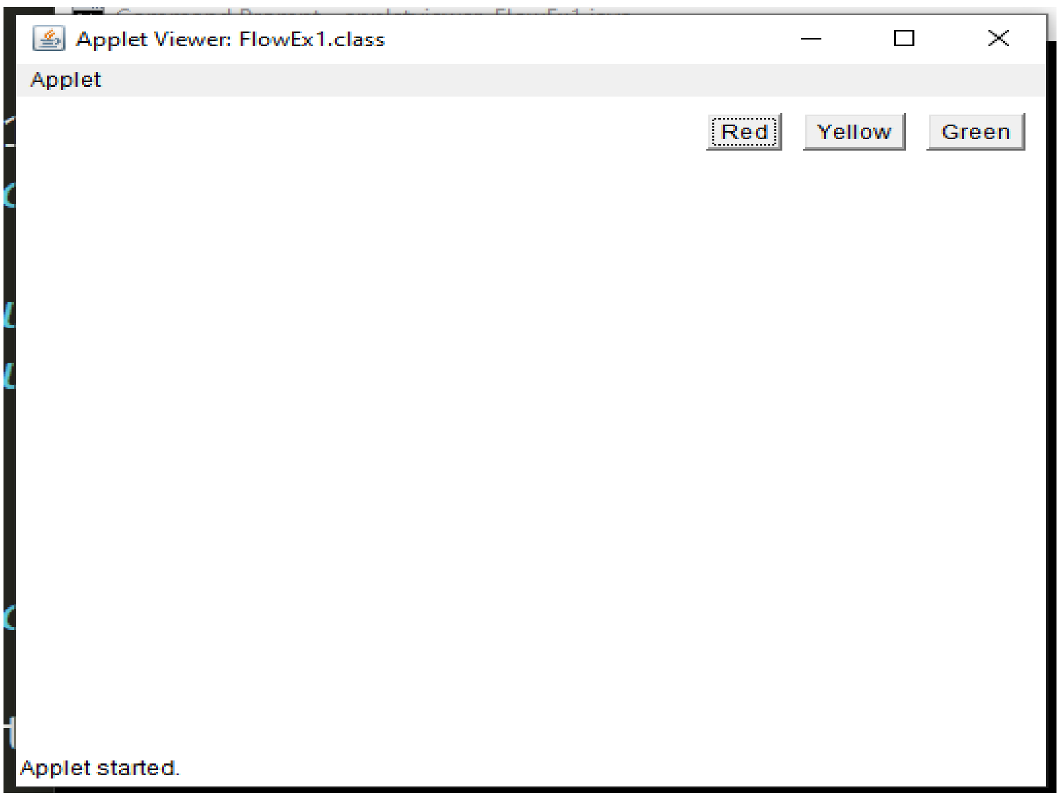
public static final int CENTER

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 Flow Layout

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 horz, 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);
    }
else if(ae.getSource()==b2)
 {
   setBackground(Color.yellow);
 }
else
 {
  setBackground(Color.green);
 }
}
}
```



2. Border Layout

The Border Layout 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 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.

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

BorderLayout.NORTH

BorderLayout.SOUTH

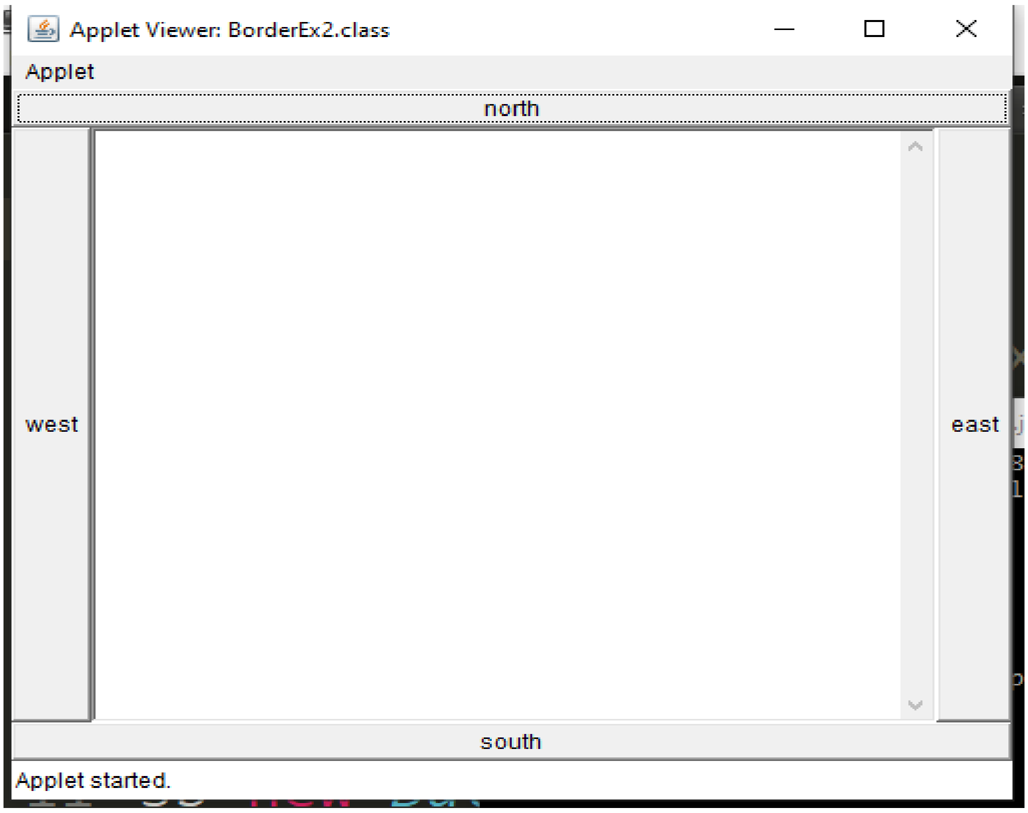
BorderLayout.EAST

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. Grid Layout

GridLayout lays out components in a two-dimensional grid.

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

Constructors of GridLayout class

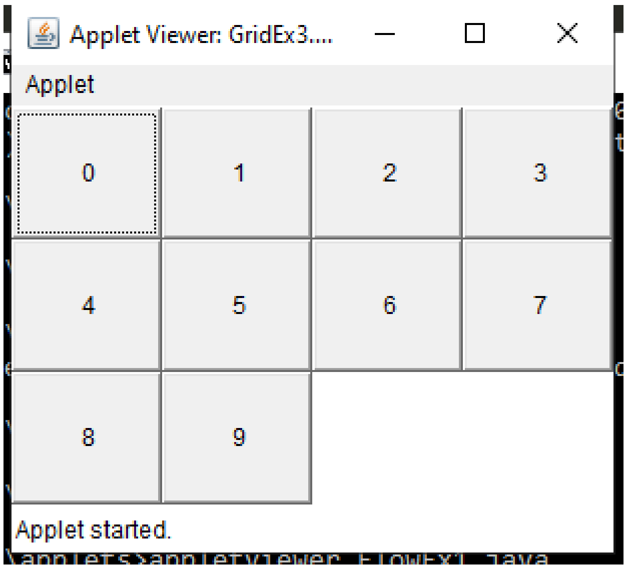
GridLayout(): creates a grid layout with one column per component in a row.

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.*;
/*<applet code="GridLayoutEx3.class" width=300 height=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,int vert)

Methods:

void add(Component panelObj, String cardName)

Note: cardName is not visible. Each card is one Panel. These cardNames are useful in selecting the random cards using the show() method.

void first(Container deck)

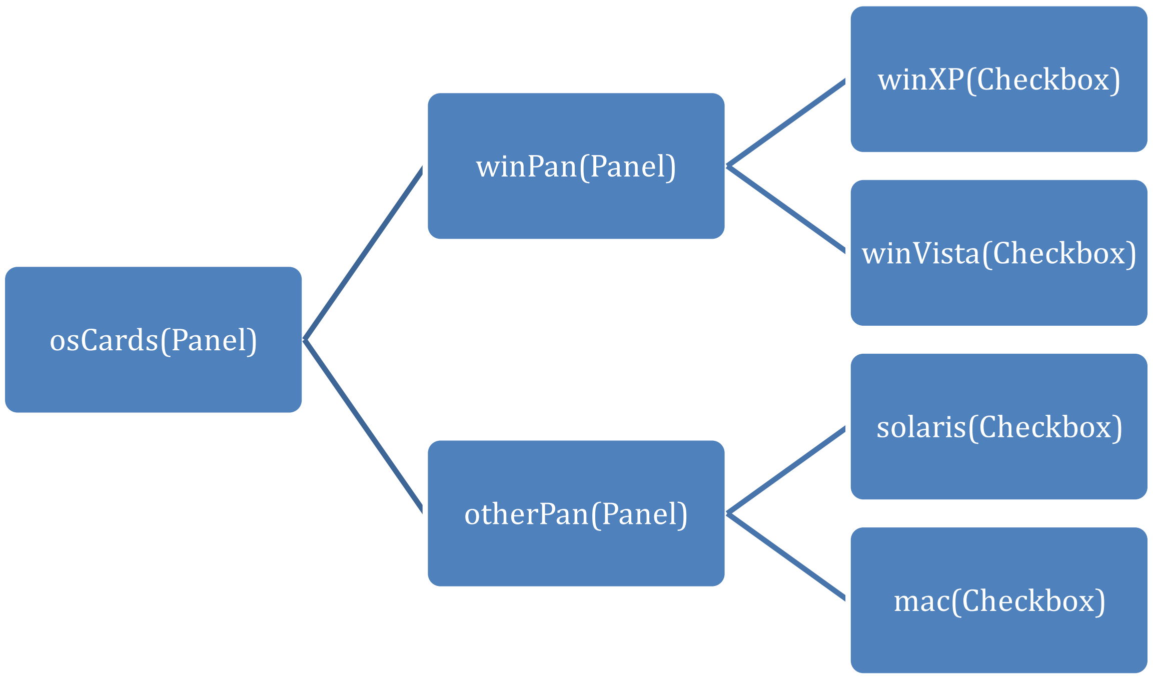
void last(Container deck)

void next(Container deck)

void previous(Container deck)

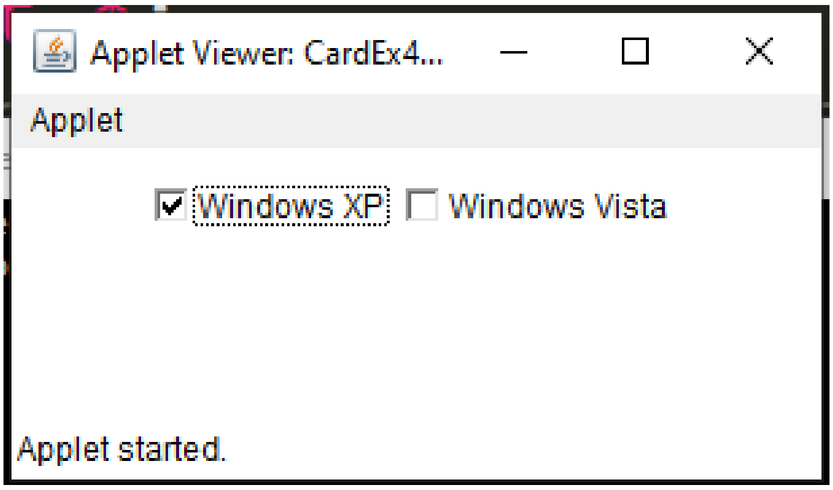
void show(Container deck, String cardName)// to select the random card with cardName

Note: In the next Program we are creating the following structure. osCards is a container deck which holds 2 cards winPan and otherPan. winPan is a Panel which holds 2 checkboxes winXP and winVista. otherPan is a Panel which holds 2 checkboxes solaris and mac.



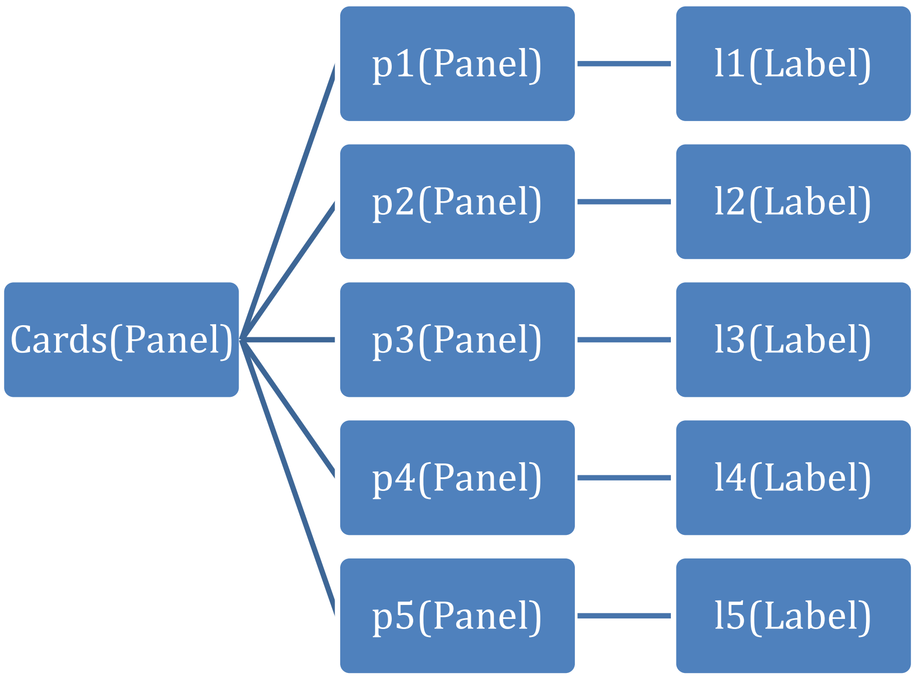
```
import java.awt.event.*;
import java.awt.*;
import java.applet.*;
```

```
/*<applet code="CardEx3.class" width=300 height=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  check boxes to a panel
        winPan.add(winXP);
        winPan.add(winVista);

        Panel otherPan = new Panel();
        // add solaris and mac check boxes 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(MouseEvent me)
        {
        }
        public void mouseEntered(MouseEvent me)
        {
        }
        public void mouseExited(MouseEvent me)
        {
        }
        public void mouseReleased(MouseEvent me)
        {
        }
}
```

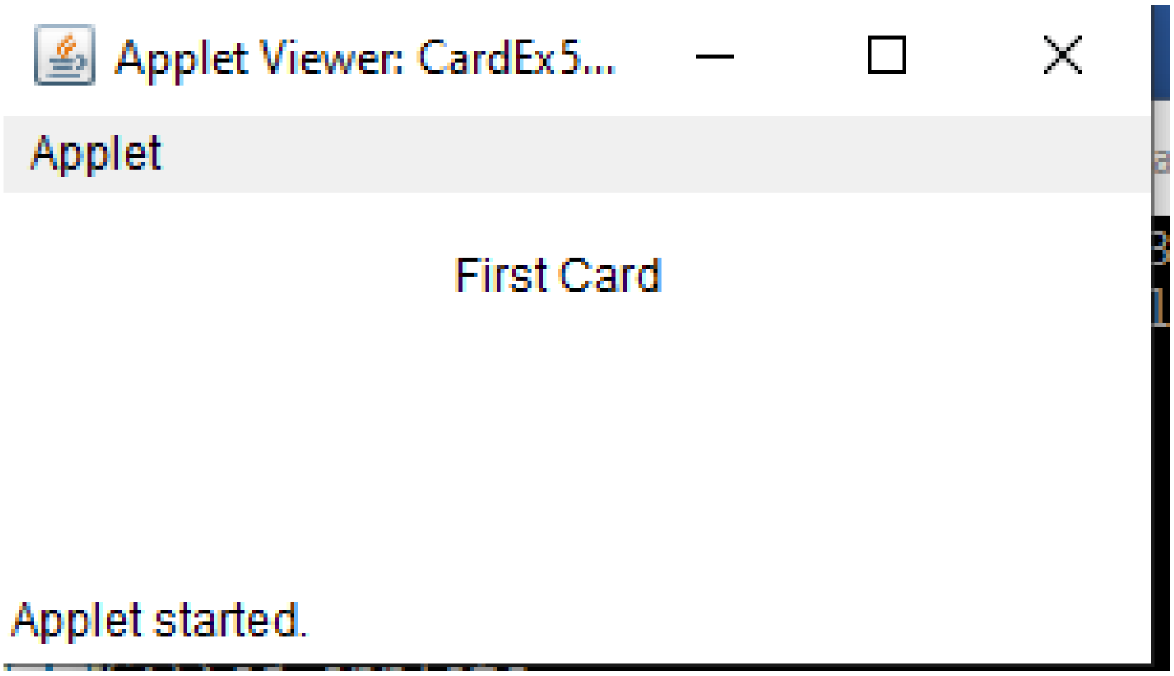


Applet started.

Program 2: Implement the following structure using CardLayout



```
import java.awt.event.*;
import java.awt.*;
import java.applet.*;
/*<applet code="CardEx5.class" width=300 height=100></applet>*/
public class CardEx5 extends Applet implements MouseListener
{
Label l1,l2,l3,l4,l5;
Panel Cards;
//Panel definition
CardLayout cardLO;
public void init()
{
cardLO = new CardLayout();
Cards = new Panel();
Cards.setLayout(cardLO); // set card layout to card deck Panel

l1 = new Label("First Card");
l2 = new Label("Second Card");
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);

// add panels to card deck panel
Cards.add(p1, "First");
Cards.add(p2, "Second");
Cards.add(p3, "Third");
Cards.add(p4, "Fourth");
Cards.add(p5, "Fifth");
// add card deck to main applet window
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)
{
}
}
```



First Card

Applet started.

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.

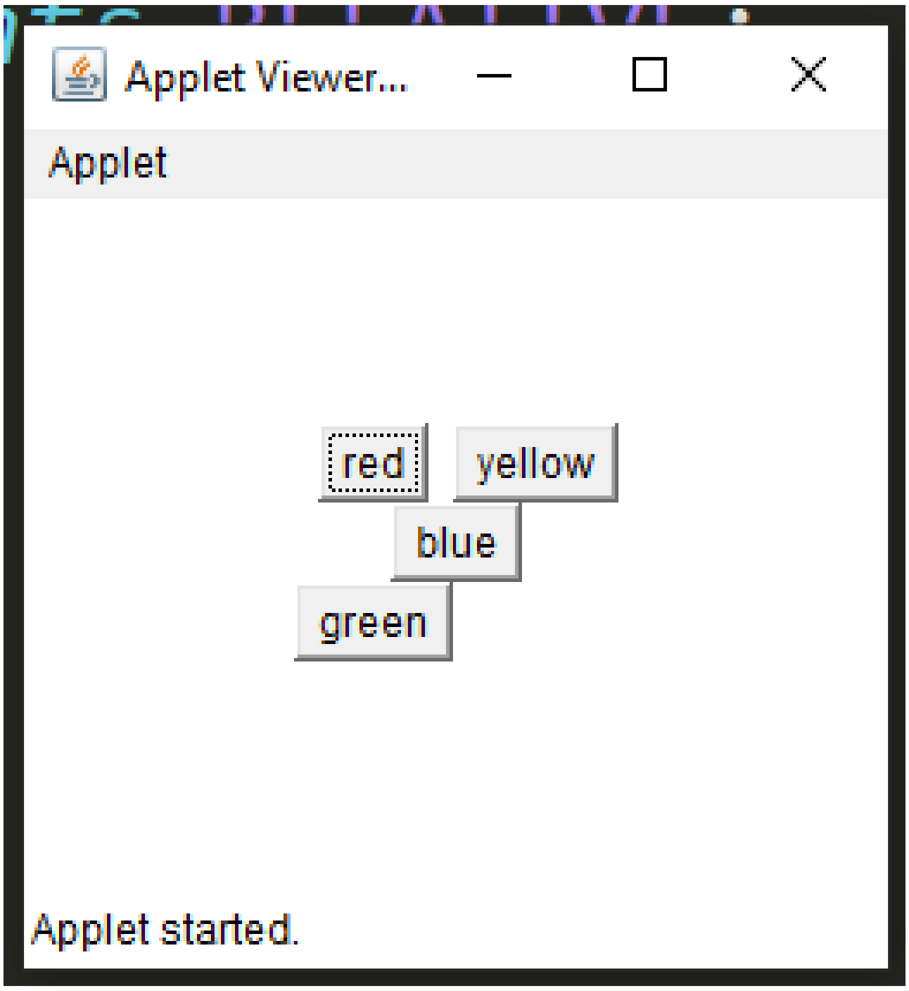
GridBagLayout defines only one **Constuctor:**

GridBabLayout()

Methods:

void setConstraints(Component comp, GridBagConstraints cons)

```
//Demostration of GridBagLayout
import java.awt.event.*;
import java.awt.*;
import java.applet.*;
/*<applet code="GBLEx6.class" width=250 height=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 awt control will take the space required for it in a row
gbag.setConstraints(b1,gbc);
add(b1);
gbc.gridwidth=GridBagConstraints.REMAINDER;
//REMAINDER means awt control will occupy the remaining space in a row
gbag.setConstraints(b2,gbc);
add(b2);
// the gridwidth constraint is not updated for the button b3 so the gridwidth constraint of
//b2 is applicable for b3.
gbag.setConstraints(b3,gbc);
add(b3);
gbc.gridwidth=GridBagConstraints.RELATIVE;
gbag.setConstraints(b4,gbc);
add(b4);
}
}
```

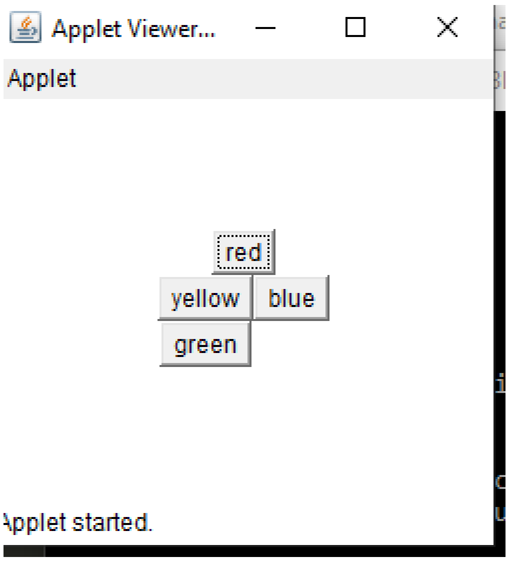


Applet Viewer...

Applet

Applet started.

Program 1: Write an applet program to obtain the following output using GridbagLayout



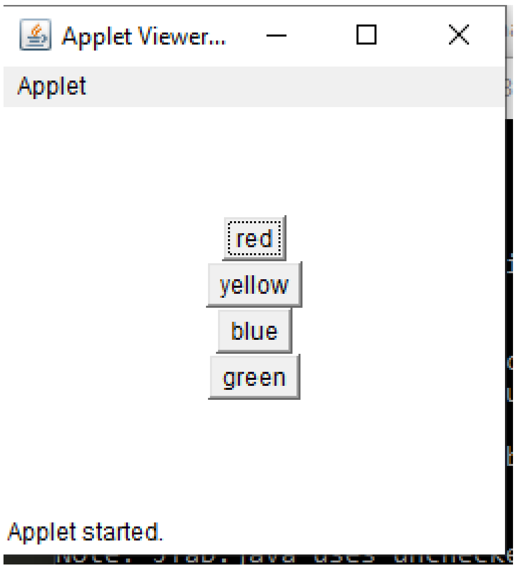
Applet Viewer...

Applet

Applet started.

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

Program 2: Write an applet program to obtain the following output using GridbagLayout



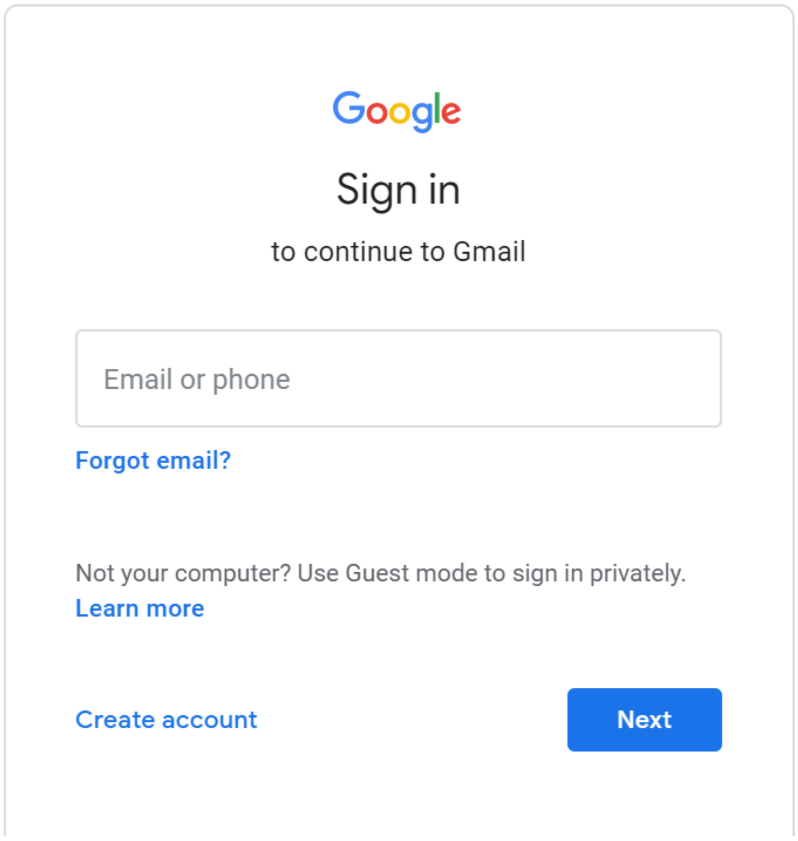
Hint: if we want to arrange one button in one line then set gridwidth to remainder for red button. For the remaining buttons if we will not change gridwidth then remainder will be carried.

Event Handling

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 **Event source.** When we **click the button** an **Event is generated**. The **Event is listened** by the **listeners** to perform the next action going to the next page.


Google

Sign in

to continue to Gmail

Email or phone

Forgot email?

Not your computer? Use Guest mode to sign in privately.
Learn more

Create account

Next

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.

- **Event Source** - 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 tl)

  Note: TypeListener is changed depending on the Event.

- **Event Listener** - 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) Event Listener→predefined interface

- When an action is performed on an event source, event is generated.
- The event listeners will wait until an event is generated.
- Once an event is performed, one or more listeners will listen 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
WindowEvent	WindowListener

ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

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

In 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 listeners to source(source can be applet window/awtcontrols)
                                    (public void addTypeListener(TypeListener tl))
       //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)

void mouseMoved(MouseEvent me)

// Note: For Mouse Event and Key board Event programs awt controls are not required.

For these two programs the source is applet window.

Program: Handling Mouse Event

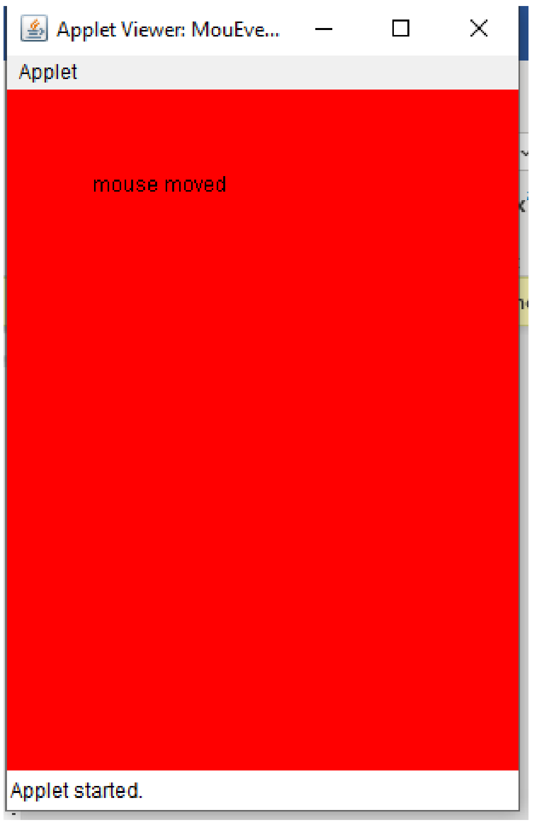
```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/* <applet code="MouEve.class" width=300 height=400></applet>*/
public class MouEve extends Applet implements MouseListener, MouseMotionListener
{
   String msg="";
   public void init()//overriding init()
   {
      addMouseListener(this);   // to register our listener to applet window
      addMouseMotionListener(this);
    }
   public void mouseClicked(MouseEvent me)
   {
       msg="Mouse clicked";
       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="mouse dragged";
         repaint();
      }
     public void mouseMoved(MouseEvent me)
     {
         msg="mouse moved";
         repaint();
      }
}
```

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

Applet Viewer: MouEve...
Applet

Applet started.

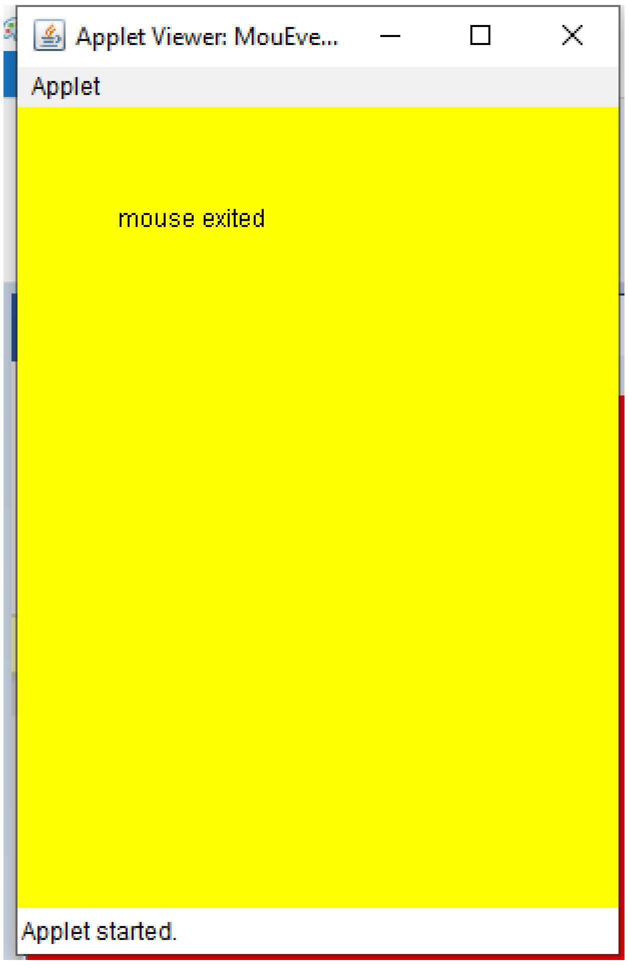
Note: output after moving the mouse pointer into the applet window. When we move the mouse pointer into the applet window two methods are performed. mouseMoved() and mouseEntered() methods. So we are able to see the message “mouse moved” present in the mouseMoved() method and the applet window color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



```
mouse moved
```

```
Applet started.
```

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



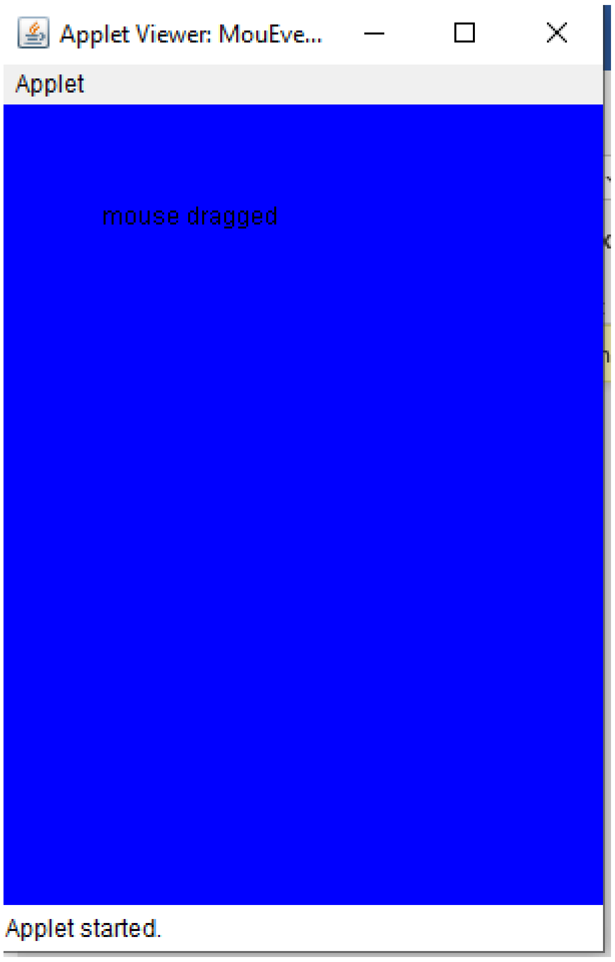
Applet Viewer: MouEve...

Applet

```
mouse exited
```

Applet started.

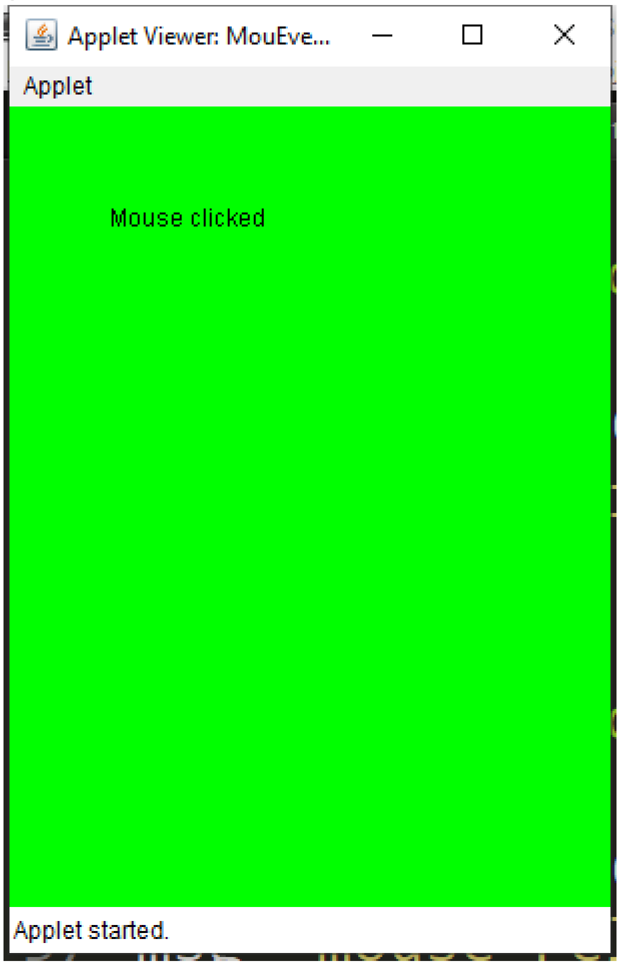
Note: For dragging the mouse we have to press the mouse button and drag so the two methods are performed mousePressed() and mouseDragged(). So from the mousePressed() method the color of the applet window is changed and from the mouseDragged() method the message " mouse dragged" is displayed on the applet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



```
mouse dragged
```

```
Applet started.
```

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.



```
Mouse clicked
```

Applet started.

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 pressed/clicked.

KeyEvent (class)

(i) char getKeyChar()

(ii) int getKeyCode()

KeyListener (interface)

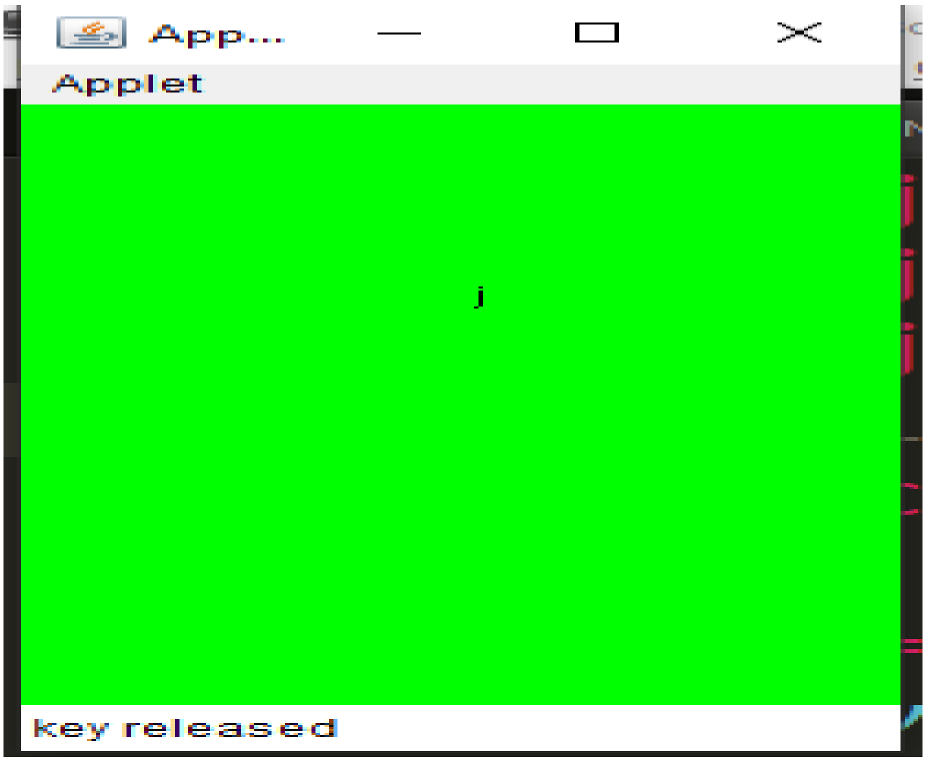
(i) void keyPressed(KeyEvent ke)

(ii) void keyTyped(KeyEvent ke)

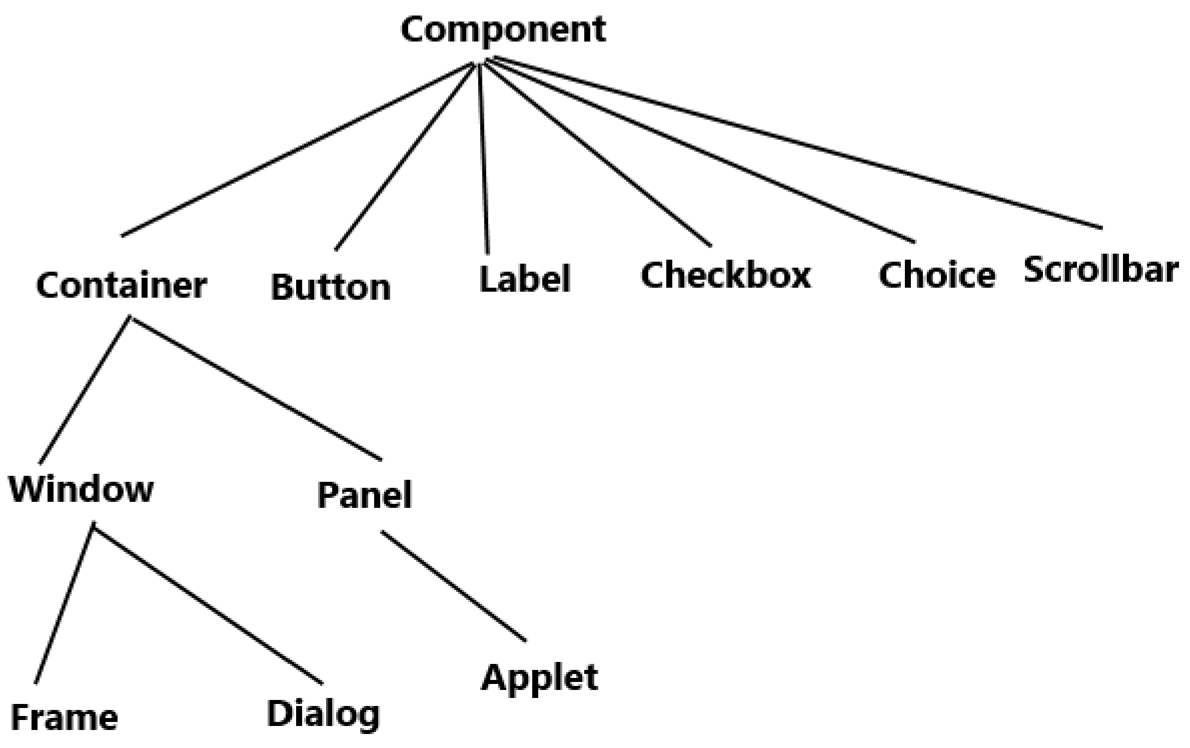
(iii) void keyReleased(KeyEvent ke)

```
// Key events program
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
/* <applet code="KeyEve.class" width=200 height=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");
}
public void keyTyped(KeyEvent ke)
{
        ch=ke.getKeyChar();
        repaint();
}
public void paint(Graphics g)
{
        g.drawString(" "+ch,100,100);
}
}
```

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



AWT Class Hierarchy:-



Awt Components

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 startIndex,int endIndex)

boolean isEditable(Boolean canedit)//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 result 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 Action Event class:

Object getSource()//returns the reference variable of the Button

String getActionCommand()//it returns the label of Button

ActionListener – is an Interface

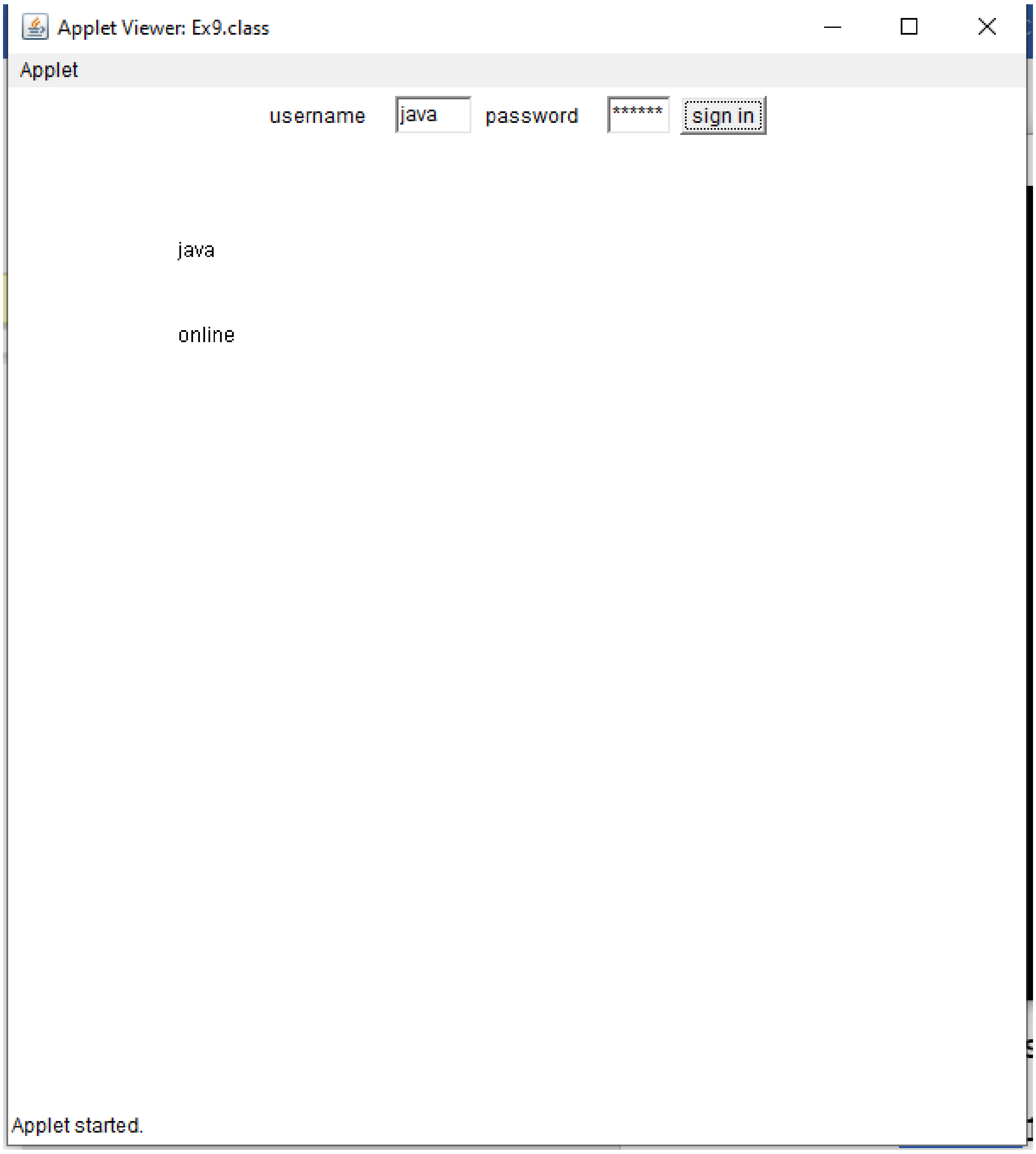
The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event package. It has only one method: actionPerformed().

Method in ActionListener:

public void actionPerformed(ActionEvent ae)

// Program on Labels and Text fields

```
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("sign in");

        b1.addActionListener(this);//registers 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);
     }
  }
```



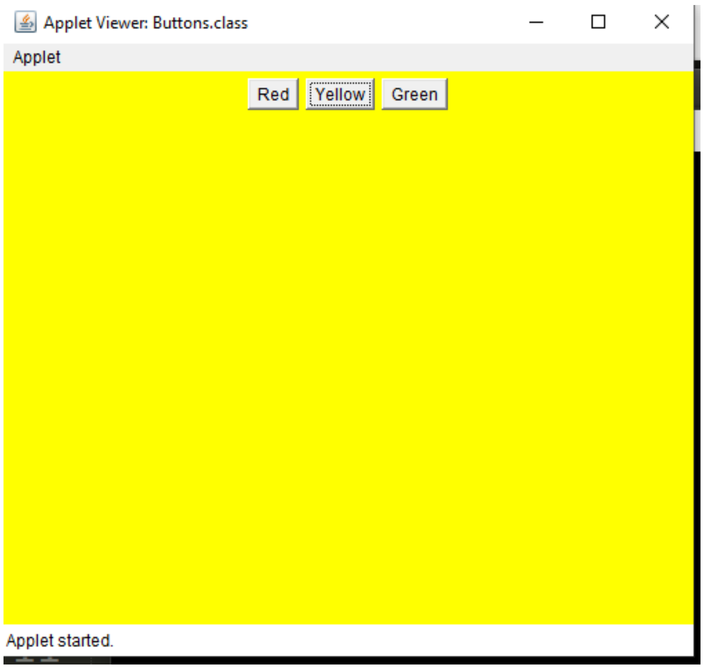
```
java

online
```

Applet started.

// java Program on Buttons

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/* <applet code="ButtonsEx10.class" width=500 height=400></applet> */
public class ButtonsEx10 extends Applet implements ActionListener
{
  Button b1,b2,b3;
// First init() and then paint() method will be executed and initial output is printed on the applet window.
// whenever the Button is clicked actionPerformed() method is executed
```

```
 public void init()
   {
     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)// getSource() method returns the object reference of Button
              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. Choice 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 selected(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

ItemEvents:-

ItemEvent – is a class

Object getItem()

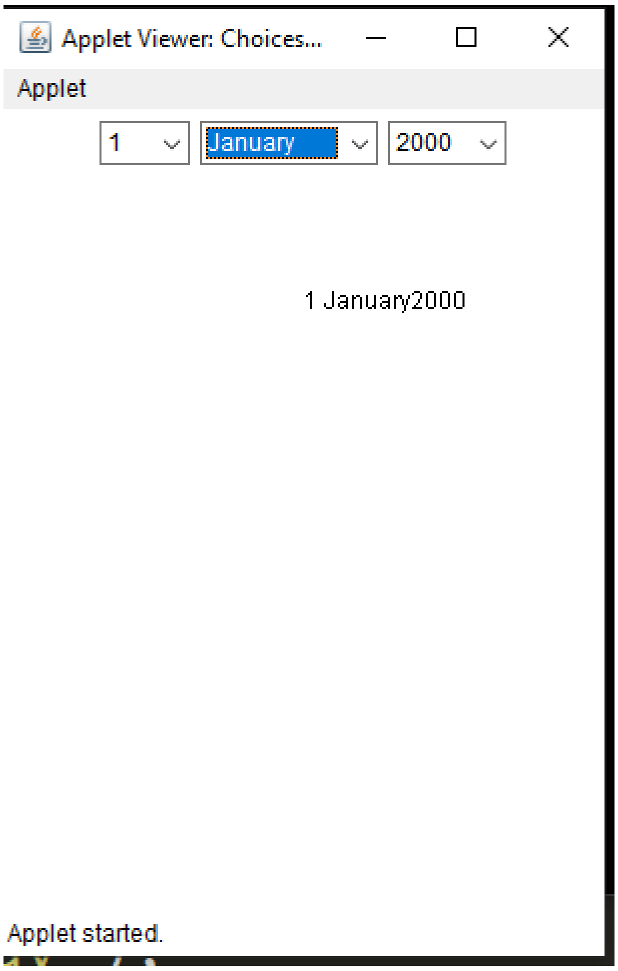
ItemListener – is an Interface

void itemStateChanged(ItemEvent ie)

//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.*;
/*  <applet code="ChoiceEx12.class" height=400 width=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);
        }
}
```



```
1 January2000
```

Applet started.

5. List

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 selections 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 multipleselct=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()

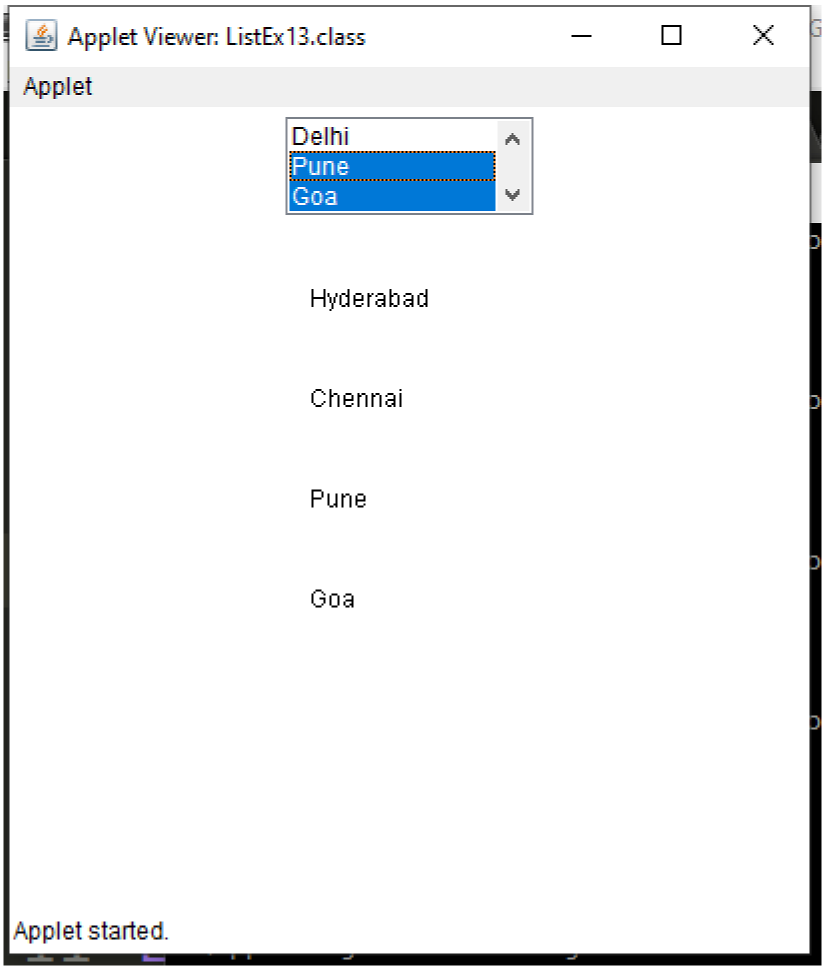
void Select(int Index)

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=new List(3,true);//3 visible out of 6 any no.of items are selected
                        L1.add("Hyderabad");
                        L1.add("Bangolore");
                        L1.add("Chennai");
                        L1.add("Delhi");
                        L1.add("Pune");
                        L1.add("Goa");
                        L1.addItemListener(this);
```

```
                add(L1);
        }
        public void itemStateChanged(ItemEvent ie)
        {
                repaint();
        }
        public void paint(Graphics g)
        {
                String str[]=L1.getSelectedItems();
                for(int i=0;i<str.length;i++)
                        g.drawString(str[i],150,100+i*50);
        }
}
```

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



```
Hyderabad

Chennai

Pune

Goa
```

Applet started.

6. 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 getLable()

void setLabel(String str)

// Event source is Checkbox

ItemEvent: //Event

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

ItemListener: // Listener

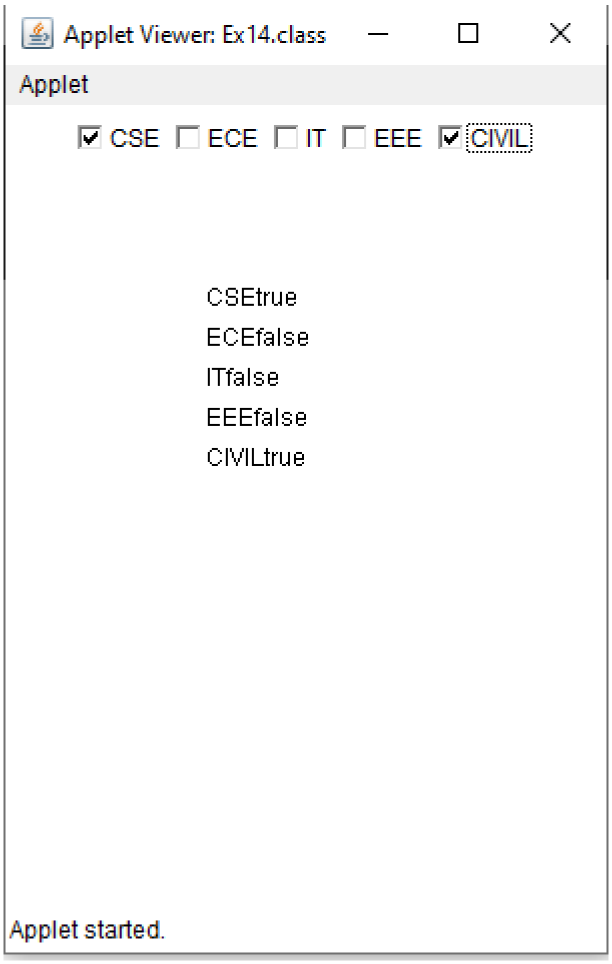
void itemStateChanged(ItemEvent ie)

```
// Program on CheckBox
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*<applet code="CheckboxEx14.class" height=400 width=300></applet>*/
public class CheckboxEx14 extends Applet implements ItemListener
{
Checkbox cb1,cb2,cb3,cb4,cb5;
// First init() and then paint() method will be executed and initial output is printed on the applet window.
// whenever the state of any Checkbox is changed it goes  to itemStateChanged method

public void init()
 {
 cb1=new  Checkbox("CSE");//not selected
 cb2=new  Checkbox("ECE");
 cb3=new  Checkbox("IT");
 cb4=new  Checkbox("EEE");
 cb5=new  Checkbox("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 the state of checkbox is changed from one state to another state ItemEvent is generated so
//itemStateChanged() method is invoked by itemListener
public void itemStateChanged(ItemEvent ie)
{
repaint();//overwrite the String in the output
}
//As part of Applet Life cycle 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 itemStateChange() method then the String(Label and State) is overwritten in the output.

3: Note1 and Note2 are applicable for all the Event handling Programs



```
CSEtrue
ECEfalse
ITfalse
EEEfalse
CIVILtrue
```

7. CheckboxGroup (radio button)

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

Constructor:

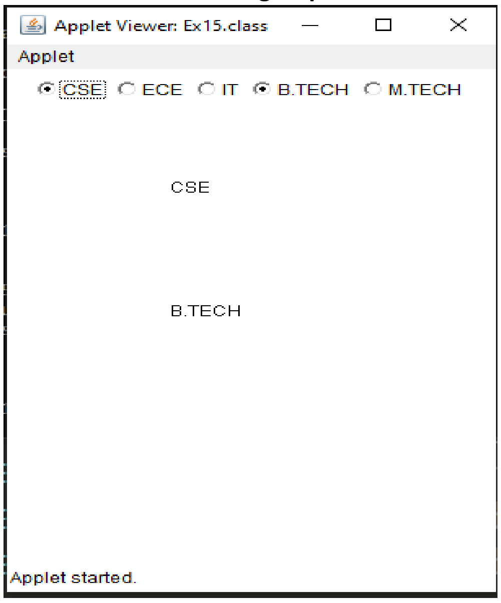
CheckboxGroup()

Methods

Checkbox getSelectedCheckbox( )
void setSelectedCheckbox(Checkbox which)

```
// Program on CheckBox
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*<applet code="CheckboxEx15.class" height=400 width=300></applet>*/
public class CheckboxEx15 extends Applet implements ItemListener
{
CheckboxGroup cbg1,cbg2;
Checkbox cb1,cb2,cb3,cb4,cb5;
// First init() and then paint() method will be executed and initial output is printed on the applet window.
// whenever the state of any Checkbox is changed it goes  to itemStateChanged method
public void init()
 {
 cbg1=new  CheckboxGroup();
 cbg2=new  CheckboxGroup();
 cb1=new  Checkbox("CSE",cbg1,true);
 cb2=new  Checkbox("ECE",cbg1,false);
 cb3=new  Checkbox("IT",cbg1,false);
 cb4=new  Checkbox("B.TECH",cbg2,true);
 cb5=new  Checkbox("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 void itemStateChanged(ItemEvent ie)
{
repaint();
}
public void paint(Graphics g)
{
 Checkbox cb6=cbg1.getSelectedCheckbox();
g.drawString(cb6.getLabel(),100,100);
 String str=(cbg2.getSelectedCheckbox()).getLabel();
g.drawString(str,100,200);
}
}
```

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



CSE

B.TECH

Applet started.

8. Scrollbars

The object of Scrollbar class is used to add horizontal and vertical scrollbar. Scrollbar is a GUI component 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 scroll bar is created.
If style is **Scrollbar.HORIZONTAL**, the scroll bar is horizontal.

In the third form of the constructor, the initial value of the scroll bar 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  intialvalue+unit*newIncr     1  4 7 10
void setBlockIncrement(int newIncr)//newIncr=3  intialvalue+Block*newIncr   1 31 61 91
// Block means 10, unit means 1,  initial value is 1
```

Event

AdjustmentEvent

Listener

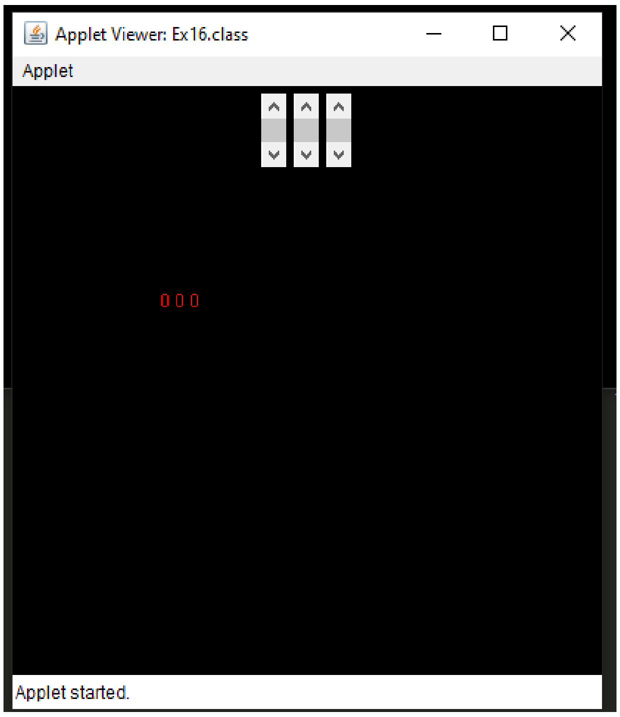
AdjustmentListener
    public void adjustmentValueChanged(AdjustmentEvent ae)

```
// Program on Scrollbars
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*<applet code="ScrollbarEx16.class" height=400 width=400></applet>*/
public class ScrollbarEx16 extends Applet implements AdjustmentListener
{
Scrollbar s1,s2,s3;
Color c;
int r=0,g=0,b=0;
public void init()
{
```

```
// Scrollbar(int style, int initialValue, int thumbSize, int min, int max)
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 intialvalue
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);
}
}
```

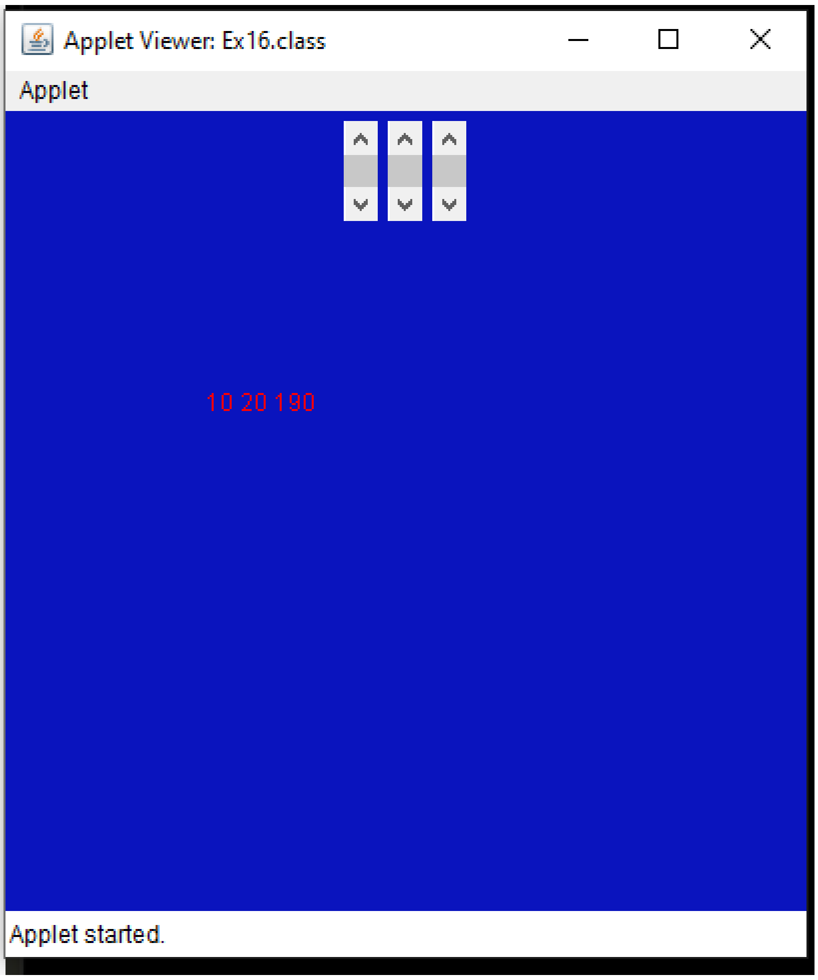
//if upward arrow is clicked the value is decreased and if downward arrow is clicked the value is increased
// When the paint() method is invoked for the first time then the output is 0 0 0

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 Scrollbars except for the Scrollbar we clicked.



```
0 0 0
```

```
Applet started.
```



```
10 20 190
```

```
Applet started.
```

9. Text Area

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

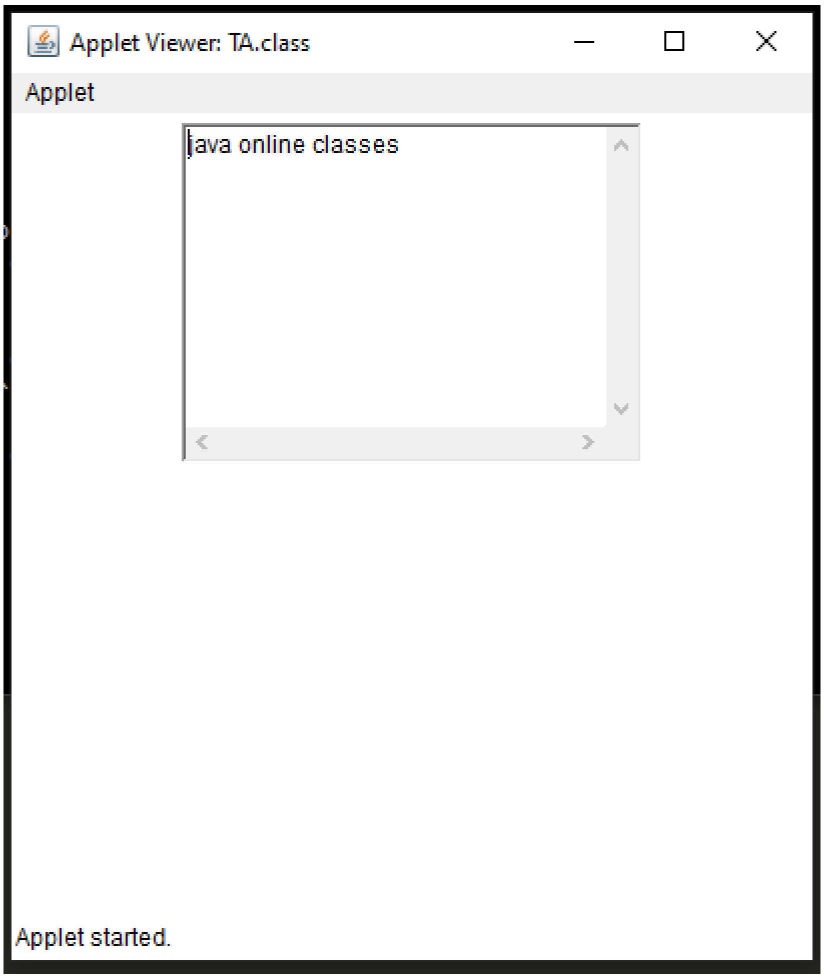
Here, numLines specifies the height, in lines, of the text area, and numChars specifies its width, in characters. Initial text can be specified by str. In the fifth form, you can specify the scroll bars that you want the control to have. sBars must be one of these values:

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.*;
/*<applet code="TA.class" height=400 width=400></applet>*/
public class TA extends Applet
{
        public void init()
        {
                TextArea ta = new TextArea("java online classes", 10, 30);
                add(ta);
        }
}
```



```
java online classes
```

Source	**Event**	**EventListener**	**Mrthod in EventListener**

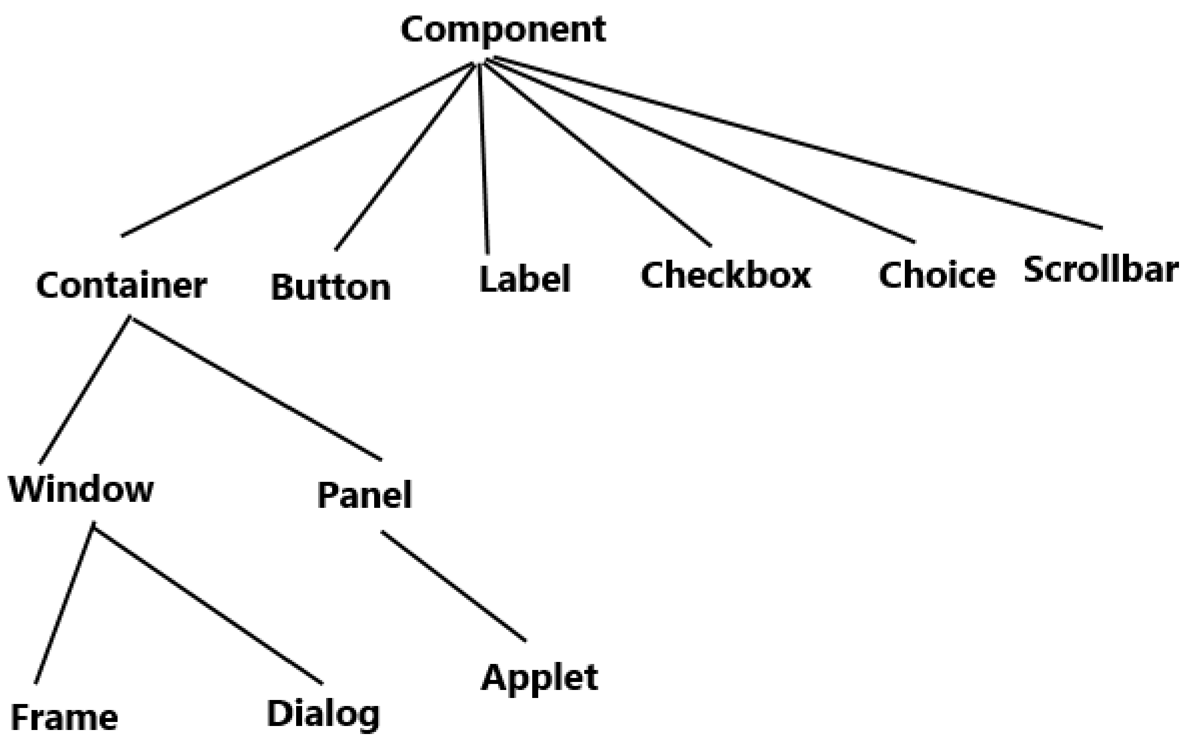
Applet	MouseEvent	MouseListener	mouseClicked()
			mousePressed()
			mouseReleased()
			mouseEntered()
			mouseExited()

		MouseMotionListene r	mouseMoved()
			mouseDragged()

	KeyEvent	KeyListener	keyPressed()
			keyReleased()
			keyTyped()

Label	-	-	-
TextField	-	-	-
Button	ActitonEvent	ActionListener	actionPerformed()
Choice List Checkbox CheckboxGrou p	ItemEvent	ItemListener	itemStateChanged()
Scrollbar	AdjustmentEven t	AdjustmentListener	adjustmentValueChanged( )
TextArea	-	-	-

Awt class hierarchy



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

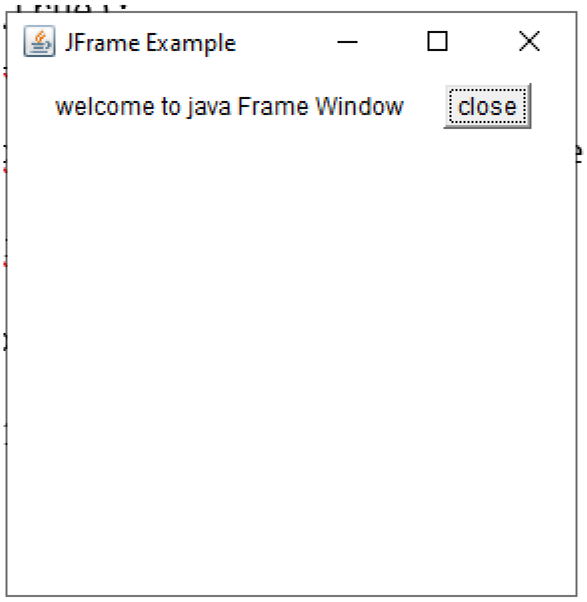
Although it is not part of the hierarchy for applet or frame windows, there is one other type of window that you will find valuable: Canvas. **Canvas encapsulates a blank window upon which you can draw.**

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[])
          {
                FrameEx11 ta = new FrameEx11();
           }
}
```



Adapter classes

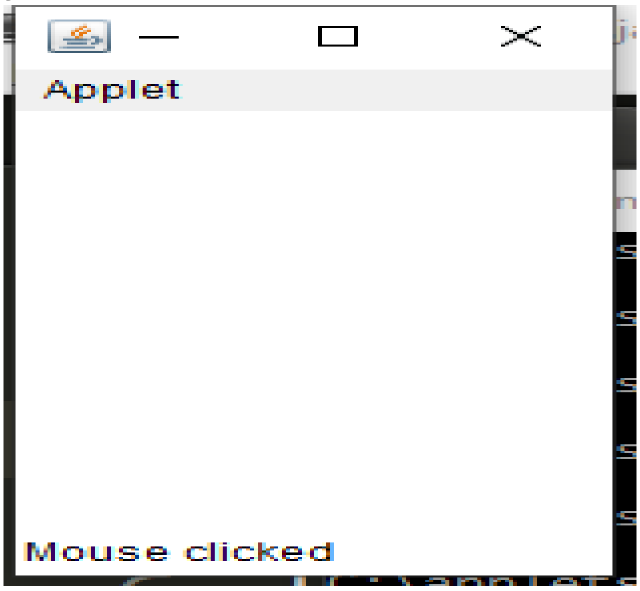
Java adapter classes provide the default implementation of listener *interfaces*. If we inherit the adapter class, we need not be forced to provide the implementation of all the methods of listener interfaces. So, it saves code.

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.
Note 2: Interfaces which have more than one method has 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.*;

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

public class AdapterDemo extends Applet
{
        public void init()
         {
                addMouseListener(new MyMouseAdapter(this));
                //event source object
                // this refers to the 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");
                    }
}
```



Mouse clicked