CHAPTER 3 (12 MARKS)

Event Handling

- **java.awt.event** package.

1)EVENT(action)

an *event* is an **object** that describes a **state change** in a **source** **for eg. Clicking on button.**

2) Event Sources

A *source* is an **object** that generates an event for eg Button.

3) Event Listeners(interface)

- A *listener(interface)* is an **object** that is notified when an event occurs,for eg ActionListener.
- 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.

```
*Button program with Event Handling

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

/*
<applet code="ButtonDemo" width=250 height=150>
</applet>
*/

public class ButtonDemo extends Applet implements ActionListener
{
String msg;
Button b1,b2;

public void init()
 {
b1 = new Button("Yes");
b2 = new Button("No");

add(b1);
add(b2);

b1.addActionListener(this);
```

1)Event handling package

2)Event Listener Interface

3)Registration method of Listener for b1, b2(we are going to click on b1,b2)

```
b2.addActionListener(this);

}

public void actionPerformed(ActionEvent ae)
 {
String str = ae.getActionCommand();
 if(str.equals("Yes"))
   {
      msg = "You pressed Yes.";
    }

    else
     {
      msg = "You pressed No.";
     }

 repaint();
  }

    public void paint(Graphics g)
  {
    g.drawString(msg, 6, 100);
   }
}
```

4)Method for button event handling from ActionListener Interface

The ActionListener Interface

This interface defines the actionPerformed( ) method that is invoked when an action event occurs. Its general form is shown here:

```
void actionPerformed(ActionEvent ae)
```

Sr.No	Source	Event(action)	EventListener	Event handling method	Event Class
1	Button	Clicking on button	ActionListener	actionPerformed( )	ActionEvent
2	Checkbox	Selecting checkbox	ItemListener	itemStateChanged()	ItemEvent
3	Choice	Selecting options from choice	ItemListener	itemStateChanged()	ItemEvent
4	checkboxgroup	Select checkbox from given option	ItemListener	itemStateChanged()	ItemEvent
5	List	Selecting option from list	ActionListener	actionPerformed( )	ActionEvent
6	TextField	Enter value in textfield	ActionListener	actionPerformed( )	ActionEvent
7	Scrollbar				

```
// Demonstrate check boxes with event handling.
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

/*
<applet code="CheckboxDemo" width=250 height=200>
</applet>
*/
public class CheckboxDemo extends Applet implements ItemListener
{
String msg = "";
Checkbox c1,c2;
public void init()
{
c1 = new Checkbox("Cricket", true);
c2= new Checkbox("Football");

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

public void itemStateChanged(ItemEvent ie)
{
repaint();
```

1)Event handling package

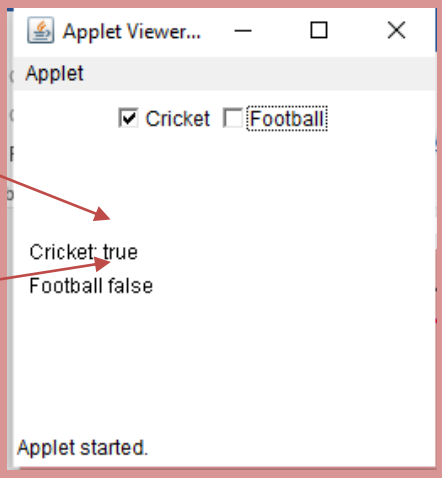
2)Event Listener Interface for Checkbox

3)Registration method of Listener for c1, c2(we are going to select c1,c2)

4)Method for checkbox event handling from ItemListener Interface

```
}
```

```
// Display current state of the check boxes.
public void paint(Graphics g)
{
msg = " Cricket: " + c1.getState();
g.drawString(msg, 6, 100);

msg = " Football " + c2.getState();
g.drawString(msg, 6, 120);
}
}
```



The ItemListener Interface

This interface defines the itemStateChanged( ) method that is invoked when the state of an item changes. Its general form is shown here:

```
void itemStateChanged(ItemEvent ie)
```

Demonstrate Choice with event handling.

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

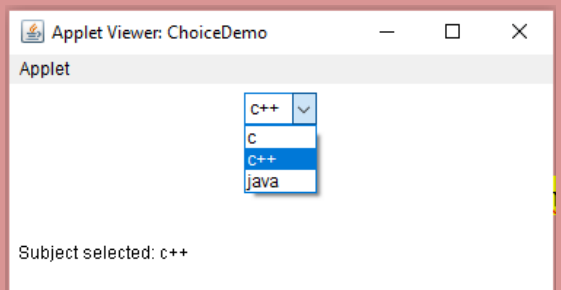
```
import java.applet.*;
import java.awt.event.*;
/*
<applet code="ChoiceDemo" width=300 height=180>
</applet>
*/
public class ChoiceDemo extends Applet implements ItemListener
{
Choice sub;
String msg = "";

public void init()
{
sub= new Choice();
// add items to subject list
sub.add("c");
sub.add("c++");
sub.add("java");
add(sub);

// register to receive item events
sub.addItemListener(this);
}

public void itemStateChanged(ItemEvent ie)
{
repaint();
}
// Display current selections.
```

1)Event handling package

2)Event Listener Interface for Choice

3)Registration method of Listener for sub(we are going to select subjects from choice sub)

4)Method for choice event handling from ItemListener Interface



```
public void paint(Graphics g)
 {
msg = "Subject selected: ";
msg += sub.getSelectedItem();
g.drawString(msg, 6, 120);
}
}
```

Demonstrate CheckboxGroup with event handling

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="CbGroup" width=250 height=200>
</applet>
*/
```

1)Event handling package

```
public class CbGroup extends Applet implements ItemListener
{
String msg = "";
Checkbox c1,c2 ;
CheckboxGroup cbg;

public void init()
 {
cbg = new CheckboxGroup();
c1 = new Checkbox("Male", cbg, true);
```

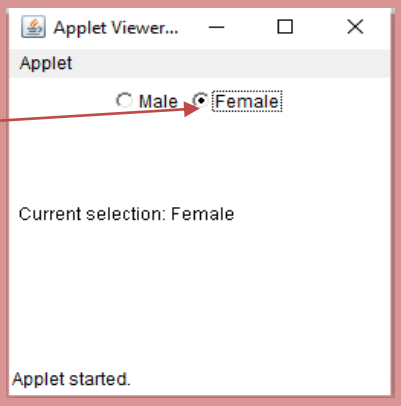
2)Event Listener Interface for ChechboxGroup

```
c2= new Checkbox("Female", cbg, false);


add(c1);

add(c2);


c1.addItemListener(this);

c2.addItemListener(this);

}

public void itemStateChanged(ItemEvent ie)

{

repaint();

}
```

3)Registration method of Listener(we are going to select option from given options)

4)Method for checkboxgroup event handling from ItemListener Interface

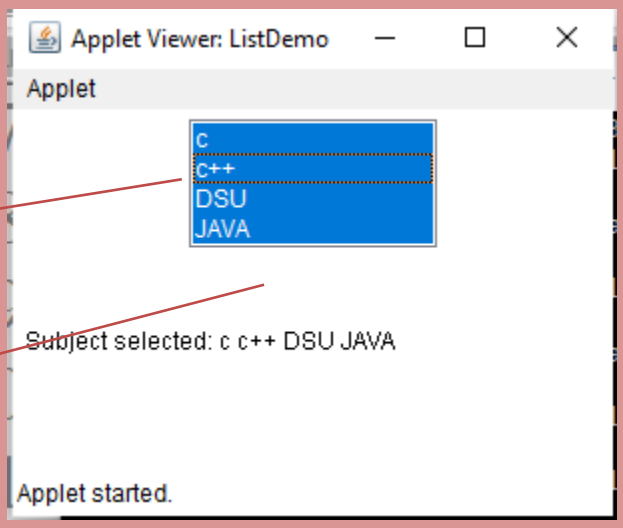
```
// Display current state of the check boxes.

public void paint(Graphics g)

{

msg = "Current selection: ";

msg += cbg.getSelectedCheckbox().getLabel();

g.drawString(msg, 6, 100);

}

}
```



List with event handling

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="ListDemo" width=300 height=180>
</applet>
*/
public class ListDemo extends Applet implements ActionListener
{
List sub;
String msg;

public void init()
{
sub = new List(4, true);

// add items to os list
sub.add("c");
sub.add("c++");
sub.add("DSU");
sub.add("JAVA");

sub.select(1);
// add lists to window
add(sub);

// register to receive action events
sub.addActionListener(this);
}
```

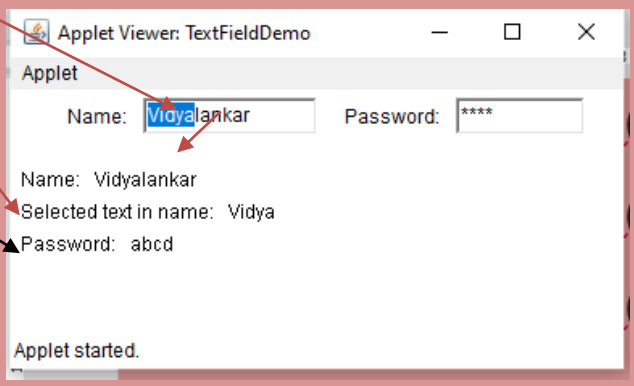
```
public void actionPerformed(ActionEvent ae)
{
repaint();
}
// Display current selections.
public void paint(Graphics g)
{
int idx[ ];
msg = "Subject selected: ";
idx = sub.getSelectedIndexes();

for(int i=0; i<idx.length; i++)
msg += sub.getItem(idx[i]) + " ";
g.drawString(msg, 6, 120);
}
}
```




```
// Demonstrate text field with event handling.
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="TextFieldDemo" width=380 height=150>
</applet>
*/
public class TextFieldDemo extends Applet implements ActionListener
```

```
{
TextField name, pass;

public void init()
{
Label l1 = new Label("Name: ");
Label l2 = new Label("Password: ");

name = new TextField(12);
pass = new TextField(8);
pass.setEchoChar('*');

add(l1);
add(name);
add(l2);
add(pass);



// register to receive action events
name.addActionListener(this);
pass.addActionListener(this);
}

// User pressed Enter.
public void actionPerformed(ActionEvent ae)
{
repaint();
}
public void paint(Graphics g)
```

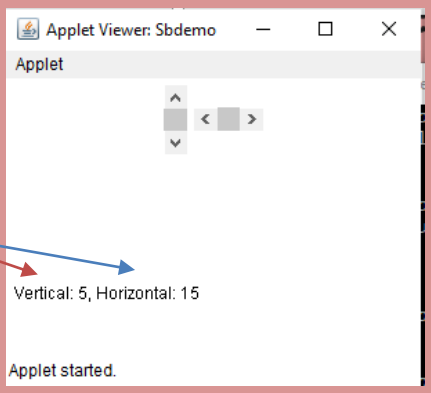
```
{
g.drawString("Name: " + name.getText(), 6, 60);
g.drawString("Selected text in name: "+ name.getSelectedText(), 6, 80);
g.drawString("Password: " + pass.getText(), 6, 100);
}
}
```



// Demonstrate scroll bars with event handling.

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

/*
<applet code="Sbdemo" width=300 height=200>
</applet>
*/

public class Sbdemo extends Applet implements AdjustmentListener
{
String msg = "";
Scrollbar vsbar, hsbar;

public void init()
{
```

```
vsbar = new Scrollbar(Scrollbar.VERTICAL,0, 100, 0, 300);
hsbar = new Scrollbar(Scrollbar.HORIZONTAL,10, 100, 0, 200);

add(vsbar);
add(hsbar);

// register to receive adjustment events
vsbar.addAdjustmentListener(this);
hsbar.addAdjustmentListener(this);
}

public void adjustmentValueChanged(AdjustmentEvent ae)
{
repaint();
}

// Display current value of scroll bars.
public void paint(Graphics g)
{
msg = "Vertical: " + vsbar.getValue();
msg += ", Horizontal: " + hsbar.getValue();
g.drawString(msg, 6, 160);
}
}
```



Mouse Event Handling

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="MouseEvents" width=300 height=100>
</applet>
*/
public class MouseEvents extends Applet implements MouseListener, MouseMotionListener
{
String msg = "";
int mousex , mousey ; // coordinates of mouse

public void init()
 {
addMouseListener(this);
addMouseMotionListener(this);
}

// Handle mouse clicked.
public void mouseClicked(MouseEvent me)
{
// save coordinates
mousex = 0;
mousey = 10;
```

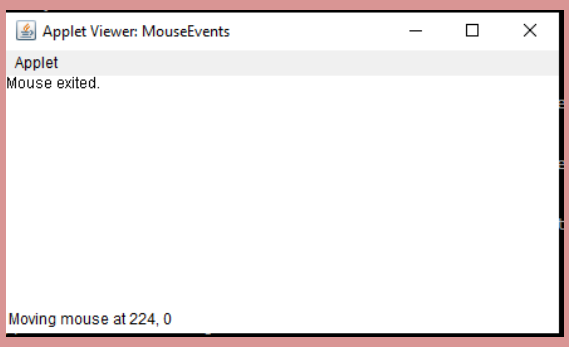
```
msg = "Mouse clicked.";
repaint();
}

// Handle mouse entered.
public void mouseEntered(MouseEvent me)
{
// save coordinates
mousex = 0;
mousey = 10;
msg = "Mouse entered.";
repaint();
}

// Handle mouse exited.
public void mouseExited(MouseEvent me)
 {
// save coordinates
mousex = 0;
mousey = 10;
msg = "Mouse exited.";
repaint();
}
```

```
public void mousePressed(MouseEvent me)
{
// save coordinates
mousex = me.getX();
mousey = me.getY();
msg = "u pressed mouse";
repaint();
}
// Handle button released.
```

```
public void mouseReleased(MouseEvent me)
{
// save current coordinates
mousex = me.getX();
mousey = me.getY();
msg = "U released the mouse";
repaint();
}
```

```
// Handle mouse dragged.
```

```
public void mouseDragged(MouseEvent me)
{
// save current coordinates
mousex = me.getX();
mousey = me.getY();
msg = "dragging mouse";
repaint();
}

// Handle mouse moved.
public void mouseMoved(MouseEvent me)
{
// show status
showStatus("Moving mouse at " + me.getX() + ", " + me.getY());
}
// Display msg in applet window at current X,Y location.
public void paint(Graphics g)
{
g.drawString(msg, mousex, mousey);
}
}
```



The MouseListener Interface(5 methods)

The general forms of these methods are shown here:
void mouseClicked(MouseEvent *me*)
void mouseEntered(MouseEvent *me*)
void mouseExited(MouseEvent *me*)
void mousePressed(MouseEvent *me*)
void mouseReleased(MouseEvent *me*)

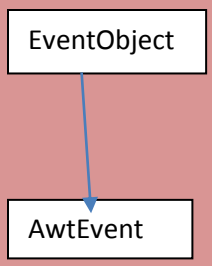
The MouseMotionListener Interface(2 methods)

Their general forms are shown here:
void mouseDragged(MouseEvent *me*)
void mouseMoved(MouseEvent *me*)

4)Event Classes
The classes that represent events are at the core of Java's event handling mechanism

- At the root of the Java event class hierarchy is **EventObject**, which is in **java.util.** It is the superclass for all events
- The class **AWTEvent**, defined within the **java.awt** package, is a subclass of **EventObject**.

To summarize:
■ **EventObject** is a superclass of all events.
■ **AWTEvent** is a superclass of all AWT events that are handled by the delegation event model.



Sr.No	Source	Event(action)	EventListener	Event handling method	Event Class
1	Button	Clicking on button	ActionListener	actionPerformed( )	ActionEvent
2	Checkbox	Selecting checkbox	ItemListener	itemStateChanged()	ItemEvent
3	Choice	Selecting options from choice	ItemListener	itemStateChanged()	ItemEvent
4	checkboxgroup	Select checkbox from given option	ItemListener	itemStateChanged()	ItemEvent
5	List	Selecting option from list	ActionListener	actionPerformed( )	ActionEvent
6	TextField	Enter value in textfield	ActionListener	actionPerformed( )	ActionEvent
7	Scrollbar				AdjustmentEvent

Adapter Classes

- An adapter class provides an **empty implementation of all methods** in an event listener interface.
- Adapter classes are useful when you want **to receive and process only some of the events** that are handled by a particular event listener interface.
- You can define a new class to act as an event listener by extending one of the **adapter classes** and implementing only those events in **which you are interested.**

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ItemAdapter	ItemListener
ActionAdapter	ActionListener

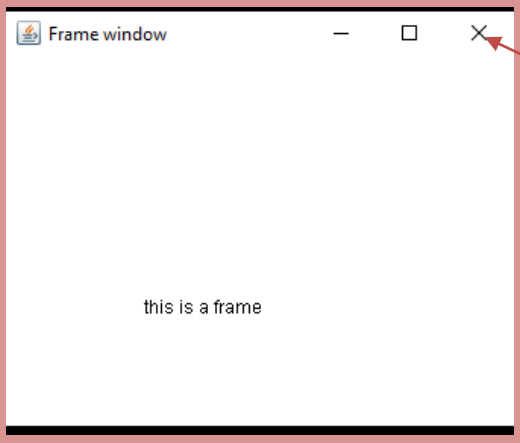
The WindowListener Interface(7 methods)

methods are-
void windowActivated(WindowEvent *we*)
void windowClosed(WindowEvent *we*)
void **windowClosing**(WindowEvent *we*)
void windowDeactivated(WindowEvent *we*)
void windowDeiconified(WindowEvent *we*)
void windowIconified(WindowEvent *we*)
void windowOpened(WindowEvent *we*)

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

 class FrameDemo extends Frame implements WindowListener
{

FrameDemo()
        {
        addWindowListener(new Myclass());
        \\step no 3-registration of window listener
        \\instead of “this” keyword we r passing object of class which includes windowClosing method
        }

        public static void main(String args[])
        {
        FrameDemo f=new FrameDemo();
        f.setSize(100,200);
        f.setTitle("Frame window");
        f.setVisible(true);
        }
        public void paint(Graphics g)
        {
        g.drawString("this is a frame",100,200);
        }

        class Myclass extends WindowAdapter
                {
                    public void windowClosing(WindowEvent we)
                    {
                    System.exit(0);
```

This is inner class as it is defined in another class i.e FrameDemo

```
        }
        }
}
```



Now we can close frame window

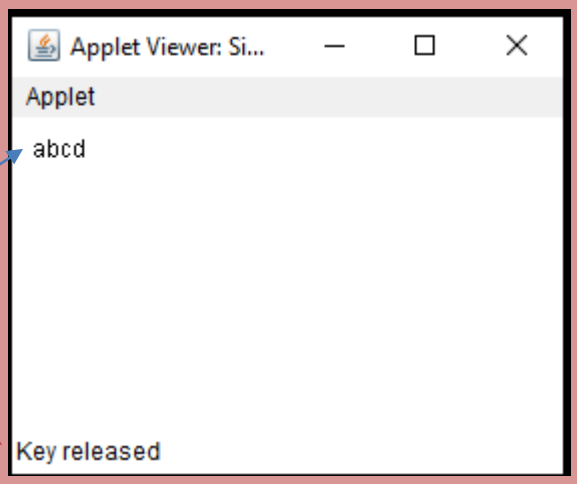
Inner Classes

An inner class is a class defined within other class, or even within an expression.

Handling Keyboard Events

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="SimpleKey" width=30 height=10>
</applet>
*/
```

```
public class SimpleKey extends Applet implements KeyListener
 {
String msg = "";
int X = 10, Y = 20; // output coordinates

public void init()
{
addKeyListener(this);
}

public void keyPressed(KeyEvent ke)
{
showStatus("Key pressed");
}
public void keyReleased(KeyEvent ke)
{
showStatus("Key released");
}
public void keyTyped(KeyEvent ke)
{
msg += ke.getKeyChar();
repaint();
}
// Display keystrokes.
public void paint(Graphics g)
{
g.drawString(msg, X, Y);
}
}
```



There are many other integer constants that are defined by KeyEvent. For example, VK_0 through VK_9 and VK_A through VK_Z

Here are some others:

VK_ENTER VK_ESCAPE VK_CANCEL VK_UP VK_DOWN VK_LEFT VK_RIGHT
VK_PAGE_DOWN VK_PAGE_UP VK_SHIFT VK_ALT VK_CONTROL

1ST CHAPTER

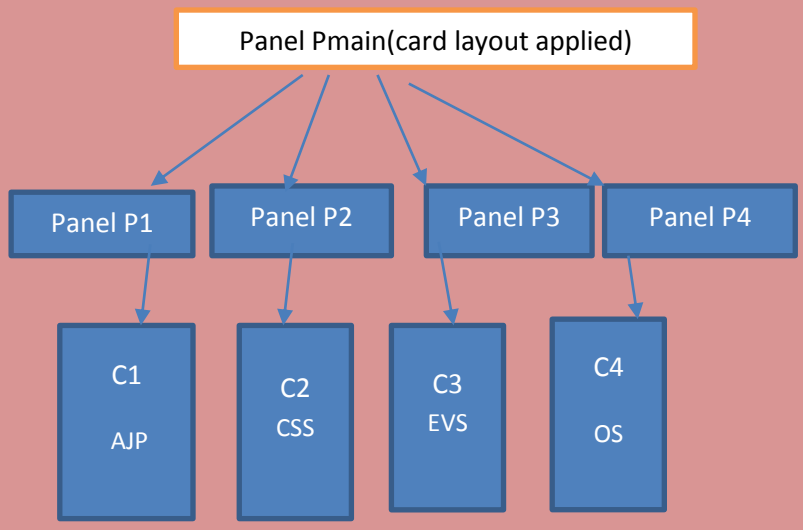
CardLayout

- The CardLayout class is unique among the other layout managers in that it **stores several different layouts**
- Use of a card layout requires a bit more work than the other layouts. The cards are typically held in an object of type **Panel**

Card Layout

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="CardLayoutDemo1" width=300 height=100>
</applet>
*/
```

```
public class CardLayoutDemo1 extends Applet implements ActionListener
 {
Checkbox c1, c2, c3, c4;
Panel pmain, p1,p2,p3,p4;
CardLayout c;
Button b1,b2,b3,b4;

        public void init()
{
b1 = new Button("first");
b2 = new Button("last");
b3= new Button("next");
b4= new Button("previous");

add(b1);
add(b2);
add(b3);
add(b4);

c = new CardLayout();
pmain = new Panel();
pmain.setLayout(c);
 // set panel layout to card layout
```



```
c1 = new Checkbox("1--AJP");

c2 = new Checkbox("2-CSS");

c3 = new Checkbox("3-EVS");

c4 = new Checkbox("4-OS");



// add Windows check boxes to a panel

Panel p1 = new Panel();

p1.add(c1);


Panel p2 = new Panel();

p2.add(c2);


      // Add other OS check boxes to a panel

Panel p3 = new Panel();

p3.add(c3);


Panel p4= new Panel();

p4.add(c4);


     // add panels to card deck panel

pmain.add(p1, "first panel");

pmain.add(p2, "second panel");

pmain.add(p3, "third panel");

pmain.add(p4, "fourth panel");


      // add cards to main applet panel

add(pmain);
```

```
      // register to receive action events
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
}

public void actionPerformed(ActionEvent ae)
{
if(ae.getSource() == b1)
{
c.first(pmain);
}
else if(ae.getSource() == b2)
{
c.last(pmain);
}
else if(ae.getSource() == b3)
{
c.next(pmain);
}
else
 {
c.previous(pmain);
}}
}
```