CS 8392 OBJECT ORIENTED PROGRAMMING

EVENT DRIVEN PROGRAMMING

Event

- Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Graphics programming

- Java contains support for graphics that enable programmers to visually enhance applications

- Java contains many more sophisticated drawing capabilities as part of the Java 2D API

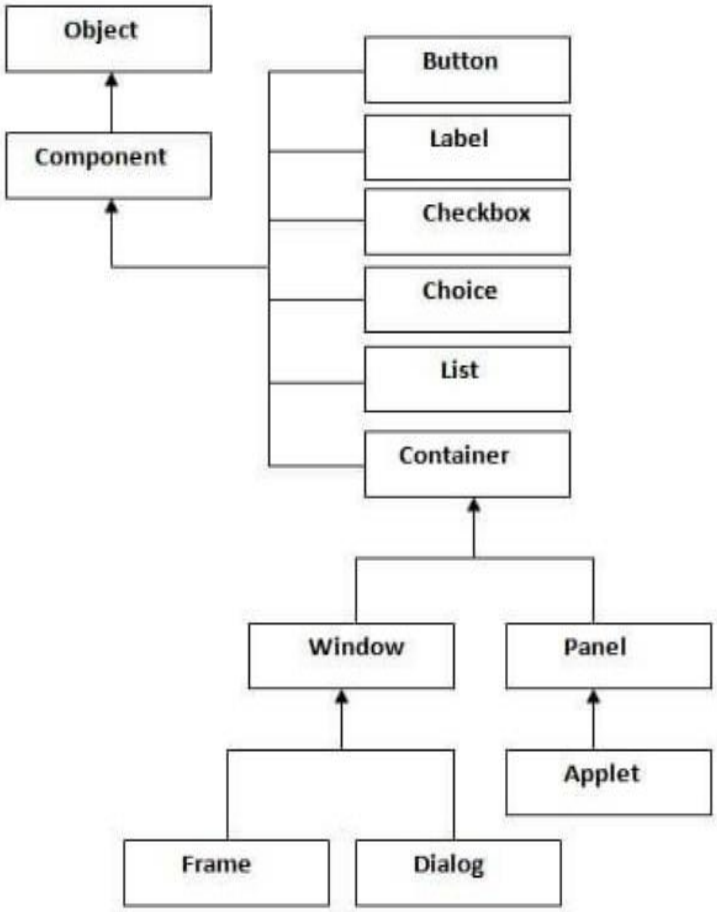
AWT

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

- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system.

- AWT is heavyweight i.e. its components are using the resources of OS.The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy



Java AWT Hierarchy Cont…..

Container

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

Window

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

Panel

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

Frame

- The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.
- There are two ways to create a Frame.

They are, By Instantiating Frame class

- By extending Frame class

```
import java.awt.*;
import java.awt.event.*;
class MyLoginWindow extends Frame
{
TextField name,pass; Button b1,b2; MyLoginWindow()
{
setLayout(new FlowLayout()); this.setLayout(null);
Label n=new Label("Name:",Label.CENTER);
Label p=new Label("password:",Label.CENTER);
name=new TextField(20);
pass=new TextField(20);
pass.setEchoChar('#');
b1=new Button("submit");
b2=new Button("cancel");
this.add(n);
this.add(name);
this.add(p);
this.add(pass);
this.add(b1);
this.add(b2);
```

```
n.setBounds(70,90,90,60);
p.setBounds(70,130,90,60);
name.setBounds(200,100,90,20);
pass.setBounds(200,140,90,20);
b1.setBounds(100,260,70,40);
b2.setBounds(180,260,70,40);
}
public static void main(String args[])
{
MyLoginWindow ml=new MyLoginWindow();
ml.setVisible(true);
ml.setSize(400,400);
ml.setTitle("my login window");
}}
```

Event handling:

- Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling. Event handling has three main components,

Events : An event is a change in state of an object.

Events Source : Event source is an object that generates an event.

Listeners : A listener is an object that listens to the event. A listener gets notified when an event occur

How Events are handled ?

- A source generates an Event and send it to one or more listeners registered with the source. Once event is received by the listener, they process the event and then return. Events are supported by a number of Java packages, like java.util, java.awt and java.awt.event.

Steps to handle events:

- Implement appropriate interface in the class.
- Register the component with the listener.

How to implement Listener

1. Declare an event handler class and specify that the class either implements an ActionListener(any listener) interface or extends a class that implements an ActionListener interface.

For example:

```
public class MyClass implements ActionListener
{
// Set of Code
}
```

2.Register an instance of the event handler class as a listener on one or more components. For example:

```
someComponent.addActionListener(instanceOfMyClass);
```

3. Include code that implements the methods in listener interface.

For example:

```
public void actionPerformed(ActionEvent e) {
//code that reacts to the action
}
```

Mouse Listener

```
package Listener;
import java.awt.Frame;
 import java.awt.Label;
import java.awt.TextArea;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
public class Mouse implements MouseListener
{
TextArea s;
public Mouse()
{
Frame d=new Frame("kkkk");
 s=new TextArea("");
d.add(s);
 s.addMouseListener(this);
d.setSize(190, 190);  d.show();
}
public void mousePressed(MouseEvent e) {
System.out.println("MousePressed"); int a=e.getX();
int b=e.getY(); System.out.println("X="+a+"Y="+b);
}
```

Mouse Listener

```
public void mouseReleased(MouseEvent e)
{
 System.out.println("MouseReleased");
}
public void mouseEntered(MouseEvent e)
{
 System.out.println("MouseEntered");
}
public void mouseExited(MouseEvent e)
{
System.out.println("MouseExited");
}
public void mouseClicked(MouseEvent e)
{
System.out.println("MouseClicked");
}
public static void main(String arg[])
{
Mouse a=new Mouse();
}
}
```

Mouse Motion Listener

```
package Listener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
public class MouseMotionEventDemo extends JPanel implements MouseMotionListener {
 MouseMotionEventDemo()
{
JTextArea a=new JTextArea();
a.addMouseMotionListener(this);
JFrame b=new JFrame();
b.add(a);
b.setVisible(true);
}
public void mouseMoved(MouseEvent e)
{
System.out.println("Mouse is Moving");
}
```

Mouse Motion Listener

```
public void mouseDragged(MouseEvent e)
 {
System.out.println("MouseDragged");
}
public static void main(String arg[])
{
MouseMotionEventDemo a=new MouseMotionEventDemo();
}
}
```

KEY LISTENER

```
package Listener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JTextField;
public class KeyEventDemo implements KeyListener
{
public KeyEventDemo()
{
JFrame s=new JFrame("hai");
JTextField typingArea = new JTextField(20);
typingArea.addKeyListener(this);
s.add(typingArea);
s.setVisible(true);
}
public void keyTyped(KeyEvent e)
 {
System.out.println("KeyTyped");
}
```

```
public static void main(String g[])
{
itemlistener l=new itemlistener();
}
public void itemStateChanged(ItemEvent arg0) {
System.out.println("State changed");
}
}
```

Window Listener

```
package Listener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
 import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
 import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
public class window extends JPanel implements WindowListener
 {
window()
{
JFrame b=new JFrame();
 b.addWindowListener(this);
b.setVisible(true);
}
```

```
public static void main(String arg[])
{
window a=new window();
}
public void windowActivated(WindowEvent arg0)
{
System.out.println("Window activated");
}
public void windowClosed(WindowEvent arg0)
{
// TODO Auto-generated method stub System.out.println("Window closed");
}
public void windowClosing(WindowEvent arg0)
{
// TODO Auto-generated method stub System.out.println("Window closing");
}
public void windowDeactivated(WindowEvent arg0)
{ // TODO Auto-generated method stub System.out.println("Window deactivated");
}
public void windowDeiconified(WindowEvent arg0) {
// TODO Auto-generated method stub System.out.println("Window deiconified");
}
public void windowIconified(WindowEvent arg0) {
// TODO Auto-generated method stub System.out.println("Window Iconified");
}
public void windowOpened(WindowEvent arg0) {
// TODO Auto-generated method stub System.out.println("Window opened");
```

WINDOW FOCUS LISTENER

```
package Listener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowFocusListener;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
public class window1 extends JPanel implements WindowFocusListener
{
window1()
{
JFrame b=new JFrame();
b.addWindowFocusListener(this);
b.setVisible(true);
public static void main(String arg[])
{
window1 b=new window1();
}
public void windowGainedFocus(WindowEvent e)
{
// TODO Auto-generated method stub System.out.println("Window gained");
}
public void windowLostFocus(WindowEvent e)
{
// TODO Auto-generated method stub System.out.println("Windowlostfocus");
```

WindowStateListener

```
package Listener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowStateListener;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
public class window2 extends JPanel implements WindowStateListener
{
window2()
{
JFrame b=new JFrame();
b.addWindowStateListener(this);
b.setVisible(true);
}
public static void main(String arg[])
{
window2 b=new window2();
}
public void windowStateChanged(WindowEvent e)
{
// TODO Auto-generated method stub System.out.println("State Changed");
```

ACTION LISTENER

```
import java.awt.*; import javax.swing.*;
 import java.awt.event.*;
import javax.swing.event.*;
public class A extends JFrame implements ActionListener
{
Scientific()
{
JPanel buttonpanel = new JPanel();
JButton b1 = new JButton("Hai"); buttonpanel.add(b1);
b1.addActionListener(this);
}
public void actionPerformed(ActionEvent e)
{
System.out.println(“Hai button”);
}
public static void main(String args[])
{
A f = new A();
f.setTitle("ActionListener");
f.setSize(500,500);
f.setVisible(true);
}}
```

Java adapter classes

- Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will 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, java.awt.dnd and javax.swing.event packages

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

Java WindowAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class AdapterExample
{
Frame f;
AdapterExample(){
f=new Frame("Window Adapter");
 f.addWindowListener(new WindowAdapter()
{
public void windowClosing(WindowEvent e)
 {
f.dispose();
}};
f.setSize(400,400); f.setLayout(null); f.setVisible(true);
}
public static void main(String[] args)
 {
 new AdapterExample();
} }
```

Java MouseAdapter Example

```
import java.awt.*;

import java.awt.event.*;
public class MouseAdapterExample extends MouseAdapter
{
Frame f; MouseAdapterExample()
{
f=new Frame("Mouse Adapter");
f.addMouseListener(this); f.setSize(300,300);
f.setLayout(null); f.setVisible(true);
}
public void mouseClicked(MouseEvent e)
{
Graphics g=f.getGraphics(); g.setColor(Color.BLUE);
g.fillOval(e.getX(),e.getY(),30,30);
}
public static void main(String[] args)
{
 new MouseAdapterExample();
}}
```

Java MouseMotionAdapter Example

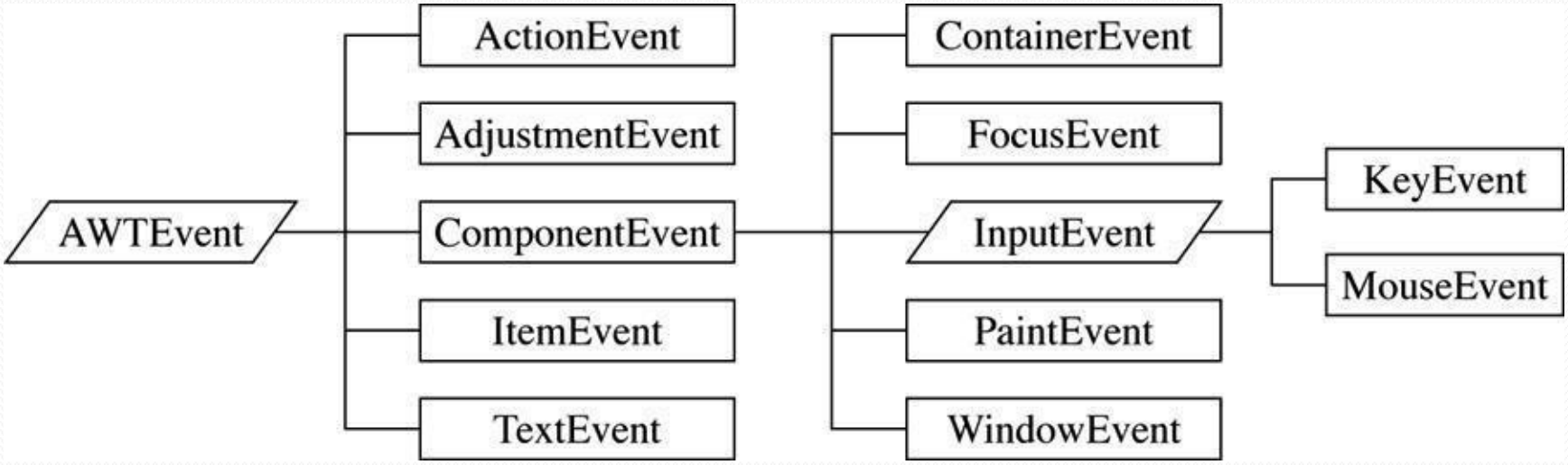
```
import java.awt.*;
 import java.awt.event.*;
public class MouseMotionAdapterExample extends MouseMotionAdapter
{
Frame f; MouseMotionAdapterExample()
{
f=new Frame("Mouse Motion Adapter");
f.addMouseMotionListener(this);
 f.setSize(300,300);
f.setLayout(null); f.setVisible(true);
}
public void mouseDragged(MouseEvent e)
 {
Graphics g=f.getGraphics(); g.setColor(Color.ORANGE);
g.fillOval(e.getX(),e.getY(),20,20);
}
public static void main(String[] args)
 {
new MouseMotionAdapterExample();
} }
```

Java KeyAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class KeyAdapterExample extends KeyAdapter
{
Label l; TextArea area;
Frame f; KeyAdapterExample()
{
f=new Frame("Key Adapter");
 l=new Label();
l.setBounds(20,50,200,20);
 area=new TextArea();
area.setBounds(20,80,300, 300);
area.addKeyListener(this);
f.add(l);f.add(area);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
```

```
public void keyReleased(KeyEvent e)
 {
String text=area.getText();
String words[]=text.split("\\s");
l.setText("Words: "+words.length+" Characters:"+text.length());
}
public static void main(String[] args)
{
new KeyAdapterExample();
} }
```

AWT EVENT HIERARCHY



Difference between AWT and Swing

JAVA AWT	JAVA SWING
AWT Components are Platform Independent	JAVA SWING Components are Platform dependent
AWT components are **heavyweight**	Swing components are lightweight
AWT doesn't Support pluggable look and feel	Swing supports pluggable Look and feel.
AWT provides **less components** than Swing	Swing provides **more powerful components** such as tables, lists,scrollpanes, colorchooser, tabbedpane etc.
AWT doesn't follows MVC(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view	Swing follows MVC.

Layout management

- Java LayoutManagers
- The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:
- AWT Layout Manager Classes

Following is the list of commonly used controls while designing GUI using AWT.

BorderLayout

- The borderlayout arranges the components to fit in the five regions: east, west,north, south, and center.

CardLayout

- The CardLayout object treats each component in the container as a card. Only one card is visible at a time.

FlowLayout

- The FlowLayout is the default layout. It layout the components in a directional flow.

GridLayout

- The GridLayout manages the components in the form of a rectangular grid.

GridBagLayout

- This is the most flexible layout manager class. The object of GridBagLayout aligns the component vertically, horizontally, or along their baseline without requiring the components of the same size.

GroupLayout

- The GroupLayout hierarchically groups the components in order to position them in a Container

SpringLayout

- A SpringLayout positions the children of its associated container according to a set of constraints.

BoxLayout

- The BoxLayout is used to arrange the components either vertically or horizontally.

ScrollPaneLayout

- The layout manager used by JScrollPane. JScrollPaneLayout is responsible for nine components: a viewport, two scrollbars, a row header, a column header, and four "corner" components.

Border layout:

Example:

```
import java.awt.*;
import javax.swing.*;
public class Border
{
JFrame f;
Border()
{
f=new JFrame();
 JButton b1=new JButton("NORTH");;
JButton b2=new JButton("SOUTH");; JButton b3=new JButton("EAST");; JButton b4=new JButton("WEST");;
   JButton b5=new
JButton("CENTER");;
        f.add(b1,BorderLayout.NORTH);  f.add(b2,BorderLayout.SOUTH);
   f.add(b3,BorderLayout.EAST); f.add(b4,BorderLayout.WEST);
   f.add(b5,BorderLayout.CENTER);  f.setSize(300,300);
f.setVisible(true);
}
public static void main(String[] args) {  new Border();
} }
```

ScrollPaneLayout:

```
import javax.swing.ImageIcon;
import javax.swing.JFrame;  import javax.swing.JLabel;  import javax.swing.JScrollPane;
```

```
{
public ScrollPaneDemo() {
super("ScrollPane Demo");
ImageIcon img = new ImageIcon("child.png");
 JScrollPane png = new
 JScrollPane(new  JLabel(img));
 getContentPane().add(png);
setSize(300,250);
setVisible(true);
}
 public static void main(String[]
args) {  new ScrollPaneDemo();
} }
```

```
import java.awt.*;
import javax.swing.*;
 public class BoxLayoutExample1 extends Frame
 {
   Button buttons[];
 public BoxLayoutExample1 ()
 {
 buttons = new Button [5];
for (int i = 0;i<5;i++)
{
buttons[i] = new Button ("Button " + (i + 1));  add (buttons[i]);
}
setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));  setSize(400,400);
setVisible(true);
}
public static void main(String args[])
{
BoxLayoutExample1 b=new BoxLayoutExample1();
}
```

Group layout:

```
public class GroupExample
{
public static void main(String[] args)
{
 JFrame frame = new JFrame("GroupLayoutExample");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  Container contentPanel = frame.getContentPane();
      GroupLayout  groupLayout = new GroupLayout(contentPanel);  contentPanel.setLayout(groupLayout);
  JLabel clickMe = new JLabel("Click Here");
 JButton button = new JButton("This Button");  groupLayout.setHorizontalGroup(
          groupLayout.createSequentialGroup()
                    .addComponent(clickMe)
                    .addGap(10, 20, 100)
                         .addComponent(button));  groupLayout.setVerticalGroup(
                   groupLayout.createParallelGroup(GroupLayout.Alignment.BASELINE)
                    .addComponent(clickMe)
                         .addComponent(button)); frame.pack();
      frame.setVisible(true);
} } }
```

Swing components:

Text Fields

- The object of a JTextField class is a text component that allows the editing of a single line text.
- It inherits JTextComponent class.

Text Areas

- The object of a JTextArea class is a multi line region that
- displays text. It allows the editing of multiple line text. It

inherits JTextComponent class

Buttons

- The JButton class is used to create a labeled button that has platform independent implementation.
- The application result in some action when the button is pushed. It inherits
- AbstractButton class

Check Boxes

- The JCheckBox 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 ".It inherits JToggleButton class.

Choices (JComboBox)

- 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 JComponent class.

- The class JWindow is a container that can be displayed but does not have the title bar

Menus

- The JMenuBar class is used to display menubar on the window or frame. It may have several menus.
- The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.
- The object of JMenuItem class adds a simple labeled menu item. The items
- used in a menu must belong to the JMenuItem or any of its subclass.

Dialog Boxes

The JDialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Dialog class.Unlike JFrame, it doesn't have maximize and minimize buttons.