Object Oriented Programming with Java - 203105333

Prof. Kiran Parmar, Assistant Professor
Computer Science and Engineering

CHAPTER - 3

EVENT DRIVEN PROGRAMING

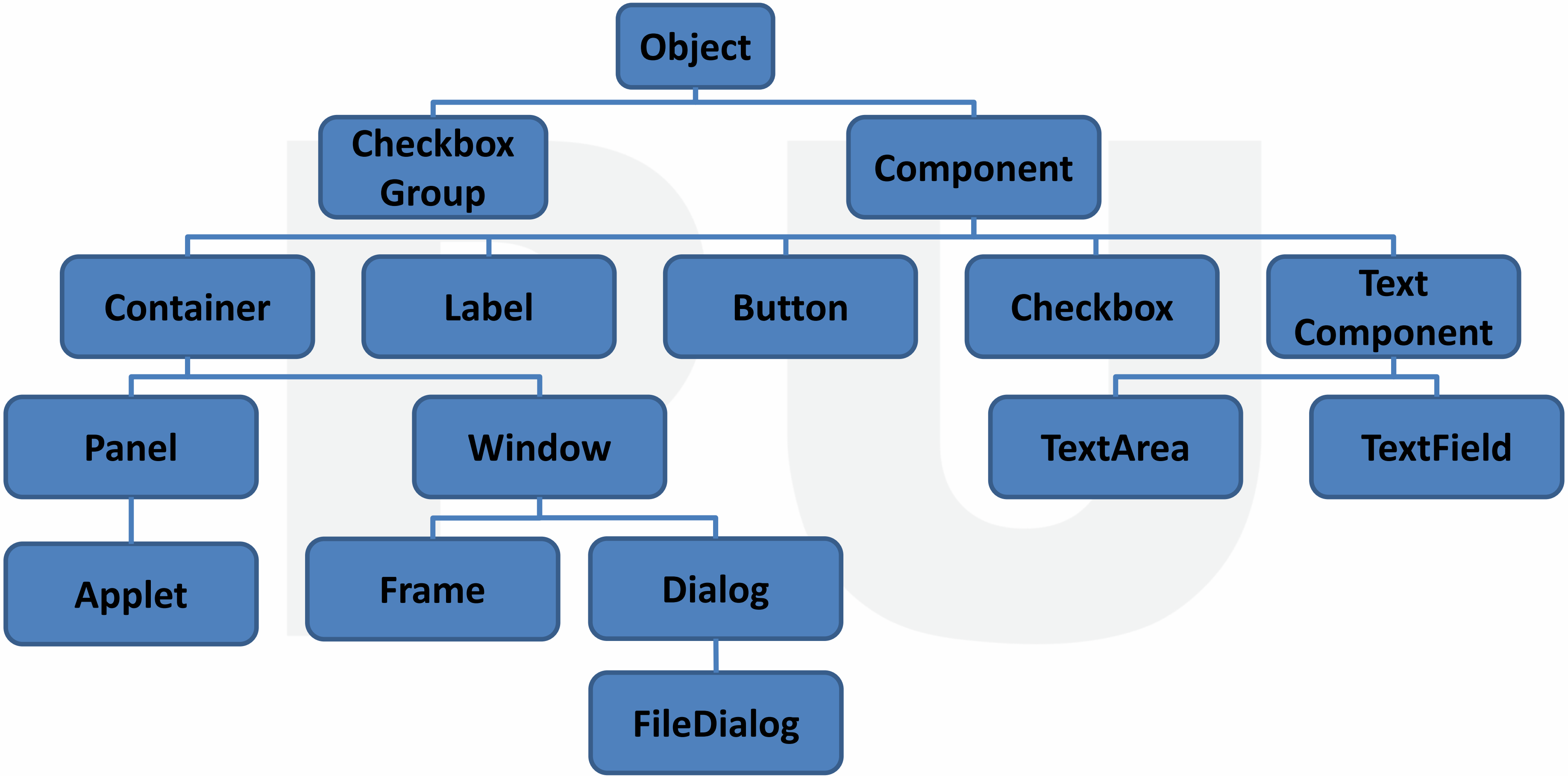
Topics

- AWT Event Hierarchy, Graphics programming, Frame
- Components, Working with shapes, Using color, fonts, and images
- Basics of event handling, Event Handlers, Adapter classes
- Actions, Mouse Events
- Introduction to Swing, Model-View- Controller
- Layout Management,
- Swing Components - Design pattern ,Buttons

Abstract Window Toolkit(AWT)

- The Abstract Window Toolkit (AWT) is Java’s original platform-independent windowing, graphics, and user-interface widget toolkit. The AWT classes are contained in the java.awt package.
  - Contains all of the classes for creating user interfaces and for painting graphics and images.
  - an API to develop GUI or window-based applications in java.

The Hierarchy of Java AWT classes



AWT Classes

- **Component**
  - A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user.

Examples :

buttons, checkboxes, and scrollbars

- The Component class is the abstract superclass of all user interface elements that are displayed on the screen.
- A Component object remembers current text font, foreground and background color.

AWT Classes

- **Container**
  - Container is a component that can contain other components like buttons, textfields, labels etc. in a specific layout.
  - It is a subclass of component class.
  - It keeps track of components that are added to another component.
  - In “Front to back” order components are listed within the container.
  - The classes that extend Container class are known as container such as Frame, Dialog and Panel.

AWT Classes

- **Window**
  - The class Window is a top level window with **no border** and **no menubar**.
  - The default layout for a window is **BorderLayout**.
  - A window must have either a frame, dialog, or another window defined as its owner when it's constructed.

AWT Classes

- **Panel**
  - The class Panel is the simplest container class.
  - It provides space in which an application can attach any other component, including other panels.
  - The default layout manager for a panel is the FlowLayout layout manager

- **Frame**
  - A Frame is a top-level window with a title and a border.
  - It uses **BorderLayout** as default layout manager.

AWT Classes

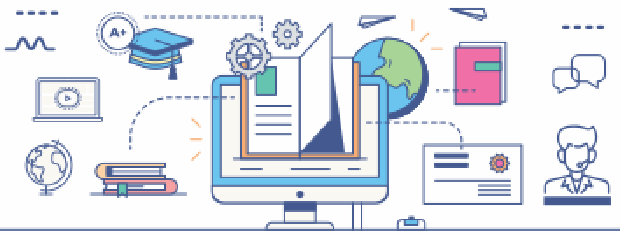
- **Dialog**
  - A Dialog is a **top-level window with a title and a border** that is typically used to take some form of **input from the user**.

- **Canvas**
  - Canvas component represents a rectangular area where application can draw something or can receive inputs created by user.
  - Drawing is not implemented on the canvas itself, but on the Graphics object provided by the canvas.
  - The Canvas is a section of a window to draw graphics or display images.
  - It is **not a part of hierarchy of Java AWT.**

Java Graphics Programming

- The java.awt.Graphics class provides many methods for graphics programming.

- A graphics context is encapsulated by the Graphics class and is obtained in two ways:
  - It is passed to an applet when one of its various methods, such as paint( ) or update( ) is called.
  - It is returned by the getGraphics( ) method of Component.



Graphics Methods

Methods	Description
abstract Graphics create()	Creates a new Graphics object that is a copy of this Graphics object
abstract void drawString(String str, int x, int y)	Draws the text given by the specified String
void drawRect(int x, int y, int width, int height)	draws a rectangle with the specified width and height
void draw3DRect(int x, int y, int width, int height, boolean raised)	Draws a 3-D highlighted outline of the specified rectangle.
abstract void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)	Draws an outlined round-cornered rectangle using this graphics context's current Color

Graphics Methods

Methods	Description
abstract void fillRect(int x, int y, int width, int height)	fill rectangle with the default color and specified width and height
abstract void drawOval(int x, int y, int width, int height)	draw oval with the specified width and height.
abstract void fillOval(int x, int y, int width, int height)	fill oval with the default color and specified width and height
abstract void drawLine(int x1, int y1, int x2, int y2)	draw line between the points(x1, y1) and (x2, y2).
abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)	draw the specified image

Graphics Methods

Methods	Description
abstract void drawArc(int x, int y, int width, int height, int startAngle, int arc Angle)	draw a circular or elliptical arc.
abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)	fill a circular or elliptical arc.
abstract void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)	Draws a closed polygon defined by arrays of x and y coordinates
abstract void fillPolygon(int[] xPoints, int[] yPoints, int nPoints)	Fills a closed polygon defined by arrays of x and y coordinates.
abstract void setFont(Font font)	set the graphics current font to the specified font.

Example: Graphics in Java

```
import java.applet.Applet;
import java.awt.*;
/* <applet code = ”GraphicsDemo.class”
width=”300” height=”300”>
</applet> */
public class GraphicsDemo extends
Applet {
public void paint(Graphics g) {
g.setColor(Color.red);

g.drawString(“Welcome”,50, 50);
g.drawLine(120,120,200,300);
g.drawRect(170,100,60,50);
g.fillRect(170,100,60,50);
g.drawOval(70,200,50,50);
g.setColor(Color.green);
g.fillOval(170,200,50,50);
// draw and fill arc
g.drawArc(90,150,70,70,0,75);
g.fillArc(270,150,70,70,0,75);
}
}
```

Frame Class

- **Frame Window**
  - A Frame provides the "**main window**" for the GUI application, which has title bar (containing an icon, a title, minimize, maximize/restore-down and close buttons), an optional menu bar, and the content display area.
  - When a frame object is created, by **default** it is **invisible**. You must call **setVisible**() method to make the frame appear on the screen.
  - Then, you must set the size of the frame using **setSize**() or **setBound**() method.

Commonly used methods of Frame class

Methods	Description
String getTitle()	Gets the title of the frame
void setBackground(Color bgColor)	Sets the background color of this window.
void setResizable (boolean resizable)	Sets whether this frame is resizable by the user.
void setShape (Shape shape)	Sets the shape of the window.
void setTitle (String title)	Sets the title for this frame to the specified string.
void setSize (Dimension d)	Resizes this component so that it has width d.width and height d.height.
void setVisible(boolean b)	Shows or hides this Window depending on the value of parameter b.

Creating a Frame

- There are **two ways to create a Frame:-**

1. By Instantiating Frame class
2. By extending Frame class

Creating Frame Window by Instantiating Frame class

```
import java.awt.*;
public class AwtFrame {
public static void main(String[] args)
{
Frame frm = new Frame(“Java AWT
Frame”);
Label lbl = new Label(“Welcome”,
Label.CENTER);
frm.add(lbl);
frm.setSize(400,400);
frm.setVisible(true);
}
}
```

Creating Frame window by extending Frame class

```
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
class DemoFrameExample extends Frame
{
DemoFrameExample()
{
super("Frame Example");
Label lb = new Label("Welcome to java world");
lb.setBounds(30,100,150,30);
add(lb);
setSize(300,300);
setLayout(null);
setVisible(true);
addWindowListener (new
WindowAdapter() {
public void windowClosing (WindowEvent
we) { System.exit(0); }
});
}
public static void main(String args[])
{
DemoFrameExample f = new
DemoFrameExample ();
} }
```

Creating an Frame Window in an Applet

- The steps to be followed to create a child frame within an applet are as follows.

1) Create a **subclass of Frame**
2) Override any of the standard window methods, such as **init(), start(), stop(), and paint()**.
3) Implement the **windowClosing()** method of the **windowListener interface**, **calling setVisible(false)** when the window is closed.
4) Once you have defined a Frame subclass, you can create an object of that class. But it will note be initially visible.
5) When created, the window is given a **default height** and **width.**
6) You can set the **size of the window** explicitly by calling the **setSize()** method.

Creating an Frame Window in an Applet

```
import java.awt.*;
import java.awt.Label;
import java.awt.event.*;
import java.applet.*;
public class DemoFrameApplet extends Applet {
  Frame f;
 public void init()
 {
  f = new Frame("A Frame Window");
  f.setSize(300, 300);
  Label lb = new Label("You are in a frame ");
  f.add(lb);
  f.setVisible(true);
}
public void start()
{  f.setVisible(true);  }
public void stop()
{  f.setVisible(false);  }
public void paint(Graphics g)
{
g.drawString("** You are in Applet **", 15, 30);
} }
```

Components

- Java AWT Component classes exist in java.awt package.

- The Component class is a super class of all components such as buttons, checkboxes, scrollbars, etc.

- **Component class constructor:**

  Component() // constructs a new component

Properties of Java AWT Components

- A Component object represents a graphical interactive area displayable on the screen that can be used by the user.

- Any subclass of a Component class is known as a component. **For example, button** is a component.

- Only components can be added to a container, like frame.

Commonly used methods of Component class

Methods	Description
setBackground(Color)	Sets the background color of this component.
setSize(int, int)	Resizes this component so that it has width width and height.
setVisible(boolean)	Shows or hides this component depending on the value of parameter b.
setFont(Font)	Sets the font of this component.
add(Component c)	Inserts a component on this component.
remove(Component c)	Removes the specified component from this component.
setBounds(int, int, int, int)	Moves and resizes this component

Working with shapes

- Java supports **2-dimensional shapes, text and images using methods available in Graphics2D class.**

- The **Graphics2D** class extends the Graphics class to provide more sophisticated control over geometry, coordinate transformations, color management, and text layout.

Example - Working with shapes

```
import java.awt.*;
import java.applet.*;
/* <applet code=”ShapesDemo”
width=350 height=300>
</applet> */
public class ShapesDemo extends
Applet {
public void init() { }
public void paint(Graphics g) {
Graphics2D g2 = (Graphics2D) g;
g2.setColor(Color.blue);
g2.drawRect(75,75,300,200);
Font exFont = new
Font(“TimesRoman”,Font.PLAIN,40);
g2.setFont(exFont);
g2.setColor(Color.black);
g2.drawString(“Graphics2D
Example”,120.0f,100.0f);
g2.setColor(Color.green);
g2.drawLine(100,100,300,200);
g2.drawOval(150,150,100,200);
g2.fillOval(150,150,100,200);
} }
```

Colors in Java

- To support different colors Java package comes with the Color class.
- The Color class states colors in the default sRGB color space or colors in arbitrary color spaces identified by a ColorSpace.
- Color class static color variables available are:

Color.black
Color.magenta
Color.darkGray
Color.white

Color.lightGray
Color.cyan
Color.red
Color.yellow

Color.blue
Color.orange
Color.green
Color.gray

Colors in Java

- **Color class constructor:**

  Color(float r, float g, float b) -

  - create color with specified red, green, and blue values in the range (0.0 - 1.0)

  Color(int r, int g, int b) -

  - create color with the specified red, green, and blue values in the range (0 - 255).

Example - Colors in Java

```
import java.awt.*;
import java.applet.*;
/*
<applet code=”ColorDemo” width=350
height=300>
</applet>
*/
public class ColorDemo extends Applet
{
public void init() {
setBackground(Color.CYAN);
}
public void paint(Graphics g) {
g.setColor(Color.red);
g.drawRect(50, 100, 150, 100);
Color clr = new Color(200, 100, 150);
g.setColor(clr);
g.fillRect(220,100, 150, 100);
} }
```

Font in Java

- The **Font class** states fonts, which are used to render text in a visible way.

- **Font class constructor**
  - **Font(Font font)** //Creates a new Font from the specified font.
  - **Font(String name, int style, int size)** //Creates a new Font from the specified name, style and point size.

Commonly used methods supported by the Font class

Method	Description
String getFamily()	Returns the family name of this Font.
int getStyle()	Returns the style of this Font
boolean isBold()	Indicates whether or not this Font object’s style is BOLD
boolean isItalic()	Indicates whether or not this Font object’s style is ITALIC
boolean isPlain()	Indicates whether or not this Font object’s style is PLAIN

Font in Java

- Font variables available in Font class are:

Font.BOLD

Font.ITALIC

Font. PLAIN

Font. MONOSPACED

Font. TRUETYPE_FONT

Font.SANS_SERIF

Font. CENTER_BASELINE

Font. DIALOG

Font. SERIF

Font. TYPE1_FONT

Example - Font in Java

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
/* <applet code=”FontDemo.class”
WIDTH=300 HEIGHT=200>
</applet> */
public class FontDemo extends
java.applet.Applet {
Font f;
String m;
public void init()
{
f=new Font(“Arial”,Font.ITALIC,20);
m=”Welcome to Java”;
setFont(f);
}
public void paint(Graphics g)
{
Color c=new Color(100,100,255);
g.setColor(c);
g.drawString(m,4,20);
Font italicFont = new Font(“Serif”,
Font.ITALIC, 24);
g.setFont(italicFont);
g.drawString(“Font in ITALIC”, 50, 120);
} }
```

Images in Java

- Image control is superclass for all image classes representing graphical images.

- The **java.applet.Applet** class provides following methods to access image.
  - **getImage()** method that returns the object of Image. Its syntax is as follows.

    public Image getImage(URL u, String image){ }
  - **getDocumentBase()** method returns the URL of the document in which applet is embedded.

    public URL getDocumentBase(){}
  - URL **getCodeBase()** method returns the base URL.

    public URL getCodeBase()

Example - Image in Java

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.net.URL;
/* <applet code =”ImageDemo.class”
width=300 height=200> </applet> */
public class ImageDemo extends
java.applet.Applet
{
Image img;
public void init() { }
public void paint(Graphics g)
{
URL url1 = getCodeBase();
img = getImage(url1,”java.jpg”);
g.drawImage(img, 60, 120, this);
} }
```

Event Handling

- Any change in the state of any object is called event.
- For example, Pressing a button, entering a character in Textbox, Clicking or dragging a mouse, etc.
- The three main elements in event handling are:
  - Event
  - Events Source
  - Listeners

Elements in Event handling

- **Event:**
  - An event is a change in state of an object. For example, mouseClicked, mousePressed.
- **Events Source:**
  - Event source is an object that generates an event. Example: a button, frame, textfield.
- **Listeners:**
  - A listener is an object that listens to the event. A listener gets notified when an event occurs. When listener receives an event, it process it and then return. For example, MouseListener handles all MouseEvent.

Event Classes and Listener interfaces

Event Class	Generated When	Listener Interfaces
ActionEvent	button is pressed, menu-item is selected, list-item is double clicked	Action Listener
MouseEvent	mouse is dragged, moved, clicked, pressed or released and also when it enters or exit a component	Mouse Listener and Mouse Motion Listener
MouseWheelEvent	mouse wheel is moved	Mouse Wheel Listener
KeyEvent	input is received from keyboard	Key Listener
ItemEvent	check-box or list item is clicked	Item Listener

Example - Event in Java

```
import java.awt.event.*;
import java.applet.*;
import java.awt.*;
/* <applet code= ”Test.class”
width=400 height=300>
</applet> */
public class Test extends Applet
implements KeyListener {
String msg= “ “;
public void init()
{ addKeyListener(this); }
public void keyPressed(KeyEvent k)
{ showStatus(“KeyPressed”); }
public void keyReleased(KeyEvent k)
{ showStatus(“KeyRealesed”); }
public void keyTyped(KeyEvent k)
{
msg = msg+k.getKeyChar();
repaint();
}
public void paint(Graphics g)
{ g.drawString(msg, 20, 40); }
}
```

Adapter Classes

- An adapter class provides the **default implementation of all methods in an event listener *interfaces***.

- Adapter classes are very useful when you want to process only few of the events that are handled by a particular event listener interface.

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

Adapter classes with their corresponding listener interfaces

Adapter Class	Listener Interface
Mouse Adapter	Mouse Listener
Mouse Motion Adapter	Mouse Motion Listener
Key Adapter	Key Listener
Window Adapter	Window Listener
Focus Adapter	Focus Listener

Example – Adapter class in Java

```
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 Actions

- The ***Java Action interface*** and ***AbstractAction*** class are terrific ways of ***encapsulating behaviors (logic)***, especially when an action can be triggered from more than one place in your ***Java/Swing application***.

- An ***Action*** can be used to separate functionality and state from a component.

- ***For example,*** if you have two or more components that perform the same function, consider using an Action object to implement the function.

Java Actions

- An ***Action object*** is an ***action listener*** that provides not only action-event handling, but also centralized handling of the state of action-event-firing components such as tool bar buttons, menu items, common buttons, and text fields.

- The most common way an action event can be triggered from multiple places in a Java/Swing application is through the Java menubar (JMenuBar) and toolbar (JToolBar).

Example – Java Actions

```
JButton button = new JButton(“ << Java Action >>”);

// Add action listener to button

button.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{
  System.out.println(“You clicked the button”);
}
});
```

MouseEvent

- An event which indicates that a mouse action occurred in a component.

- A ***mouse action*** is considered to occur in a particular component if and only if the ***mouse cursor*** is over the part of the component’s bounds when the action happens.

- For ***lightweight components***, such as ***Swing’s components, mouse events*** are only dispatched to the component if the mouse event type has been enabled on the component.

MouseEvent

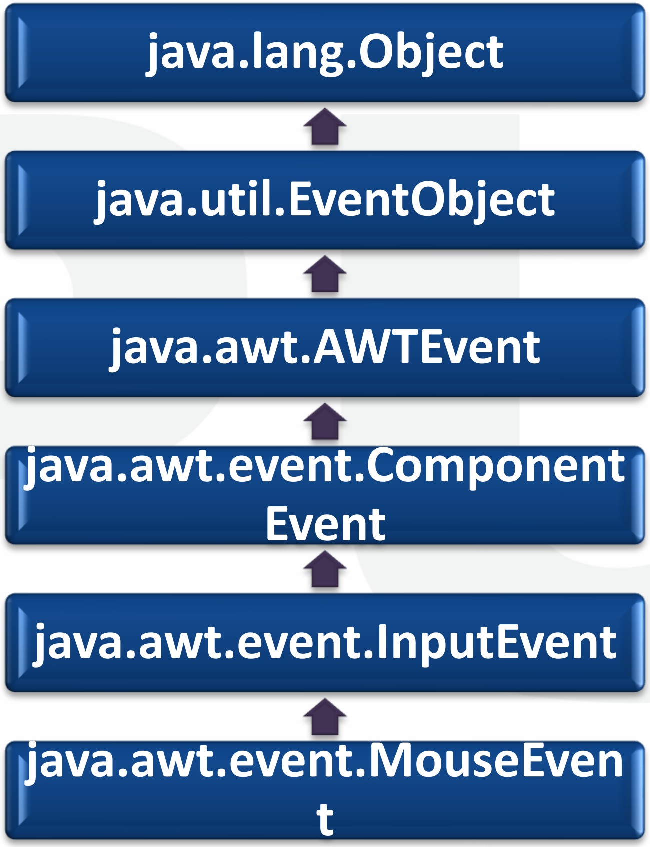
- A ***mouse event*** type is enabled by adding the appropriate mouse-based ***EventListener*** to the component (***Mouse Listener or Mouse Motion Listener***), or by invoking ***Component.enableEvents*** (long) with the appropriate mask parameter

  (AWTEvent.MOUSE_EVENT_MASK

  or

  AWTEvent.MOUSE_MOTION_EVENT_ MASK).

Hierarchy of MouseEvent class



java.lang.Object

java.util.EventObject

java.awt.AWTEvent

java.awt.event.ComponentEvent

java.awt.event.InputEvent

java.awt.event.MouseEvent

List of MouseEvent

- **mousePressed :** mouse button is pressed
- **mouseReleased :** a mouse button is released
- **mouseClicked :** a mouse button is clicked (pressed and released)
- **mouseEntered :** the mouse cursor enters the unobscured part of component’s geometry
- **mouseExited :** the mouse cursor exits the unobscured part of component’s geometry

List of Mouse Motion Event

- **mouseMoved :** the mouse is moved
- **mouseDragged :** the mouse is dragged

Example: MouseEvent

```
import java.awt.*;
import java.awt.event.*;
pubic class MouseListenerExample
extends Frame implements
MouseListener {
Label l;
MouseListenerExample() {
addMouseListener(this);
l=new Label();
l.setBounds(20,50,100,20);
add(l);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void mouseClicked(MouseEvent
e) {
l.setText(“Mouse Clicked”);
} ……… //add all implemented methods
of MouseEvent
}
```

Java Swing

- ***Swing*** was developed to provide a more sophisticated ***set of GUI components*** than the earlier ***Abstract Window Toolkit (AWT)***.

- ***Java Swing*** is a part of ***Java Foundation Classes (JFC)*** that is used to create window based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

- ***Swing*** provides programmer the facility to change the ***look and feel of components*** being displayed on any system. This is called ***'plaf' (pluggable look and feel)***.

Java Swing

- Unlike ***AWT, Java Swing*** provides ***platform-independent*** and ***lightweight components.***

- **There are 3 types of Look & Feel available in Swing :**
  - Metal Look & Feel
  - Motif Look & Feel
  - Windows Look & Feel
- By default, Swing programmer use ‘**metal look and feel**’.

- The **javax.swing** package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Swing Features

- ***Light Weight*** - Swing component are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.

- ***Rich controls*** - Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colourpicker, table controls

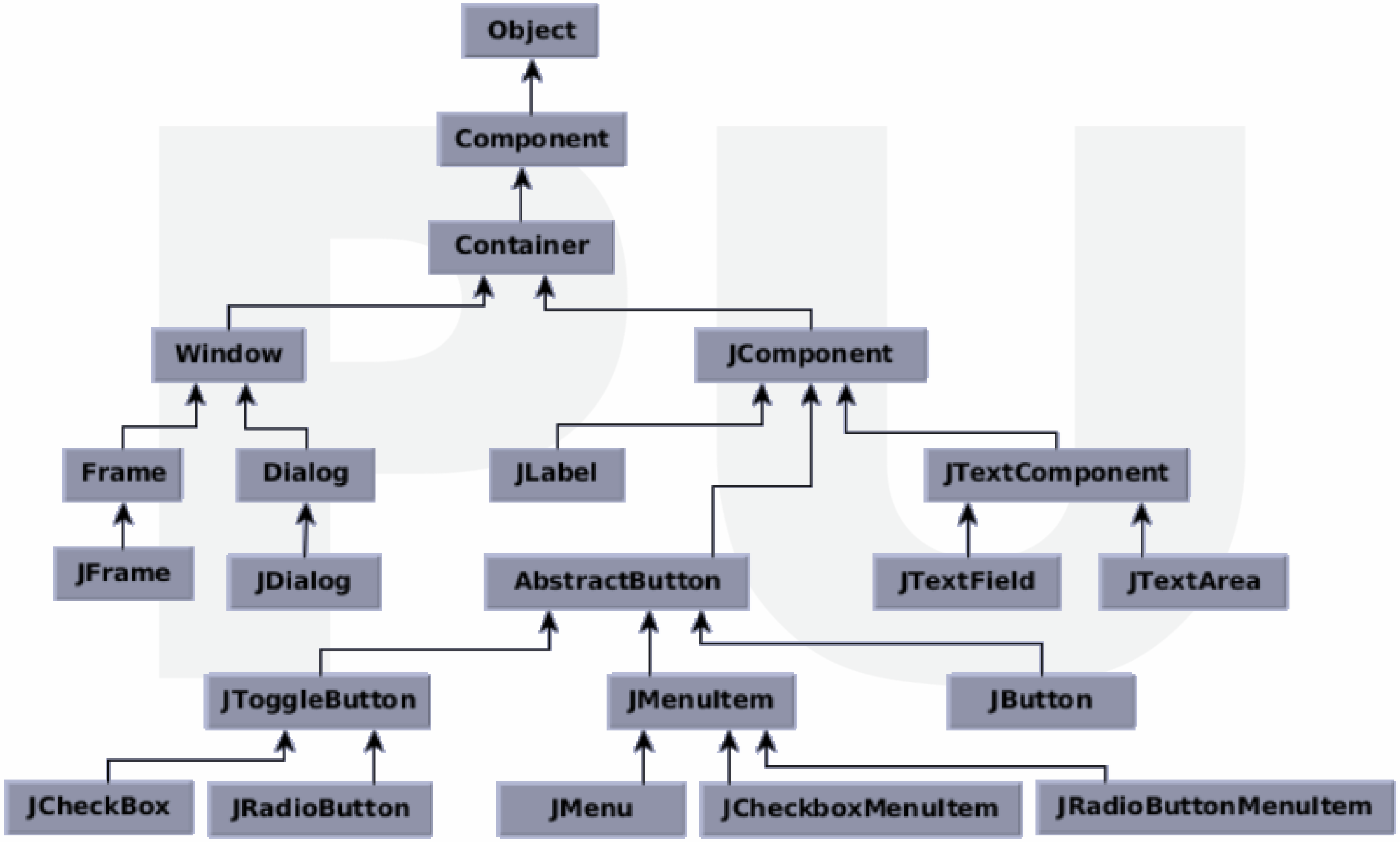
- **Highly Customizable** - Swing controls can be customized in very easy way as visual appearance is independent of internal representation.

- ***Pluggable look-and-feel -*** SWING based GUI Application look and feel can be changed at run time based on available values.

Differences between Java AWT and Swing

Java AWT	Java Swing
AWT stands for Abstract windows toolkit.	Swing is also called as JFC.
AWT components are platform-dependent.	Java swing components are platform-independent.
AWT components are heavyweight.	Swing components are lightweight.
AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
AWT components require java.awt package.	Swing components require javax.swing package.
AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.

Hierarchy of Java Swing classes



Swing Components

- ***Swing components*** are basic building blocks of an application.
- Swing has a wide range of various components, including buttons, check boxes, sliders, and list boxes.
- They all are derived from JComponent class. All these components are lightweight components.
- This class provides some common functionality like pluggable look and feel, support for accessibility, drag and drop, layout, etc.

Swing Components

- **Container:** An abstract class that extends Component. Containers can hold multiple components.

- **Containers are of two types:**
  - Top level Containers
  - Lightweight Containers

Swing Components

- **Container:** An abstract class that extends Component. Containers can hold multiple components.

- **Containers are of two types:**
  - Top level Containers
  - Lightweight Containers

Containers

- **Top Level Containers:**
  - It inherits Component and Container of AWT.
  - It cannot be contained within other containers.
  - Heavyweight.
  - Example: JFrame, JDialog, Japplet

- **Lightweight Containers:**
  - It inherits JComponent class.
  - It is a general purpose container.
  - It can be used to organize related components together.
  - Example: JPanel

JButton

- ***JButton*** extends Component , displays a string, and delivers an ActionEvent for each mouse click.
- Normally buttons are displayed with a border.
- In addition to text, JButtons can also display icons.
- JButton class has three constuctors:
  - JButton(Icon ic)
  - JButton(String str)
  - JButton(String str, Icon ic)

JLabels

- JLabels are components that you can fill with text.
- When creating a label you can specify the initial value and the alignment you wish to use within the label.
- You can use getText() and setText() to get and change the value of the label.
- The JLabel Contains 4 constructors:
  - JLabel()
  - JLabel(String s)
  - JLabel(Icon i)
  - JLabel(String s, Icon i, int horizontalAlignment)

JTextField

- JTextField is used for taking input of single line of text.
- It is widely used text Component.
- It has three constructor:
  - JTextField(int cols)
  - JTextField(String str, int cols)
  - JTextField(String str)

Commonly used Methods of Component class

- The methods of ***Component class*** are widely used in ***Java Swing*** :

Method	Description
public void add(Component c)	add a component on another component.
public void setSize (int width, int height)	sets size of the component.
public void setLayout (LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Java Swing Examples

- There are two ways to create a frame:
  - By creating the object of Frame class (association)
  - By extending Frame class (inheritance)

Simple Java Swing Example

```
import javax.swing.*;
public class FirstSwingExample {
public static void main(String[] args) {
JFrame f=new JFrame(); //creating instance of JFrame
JButton b=new JButton(“click”); //creating instance of JButton
b.setBounds(130,100,100, 40); //x axis, y axis, width, height
f.add(b); //adding button in JFrame
f.setSize(400,500); //400 width and 500 height
f.setLayout(null); //using no layout managers
f.setVisible(true); //making the frame visible
} }
```

Example : Swing by Association inside constructor

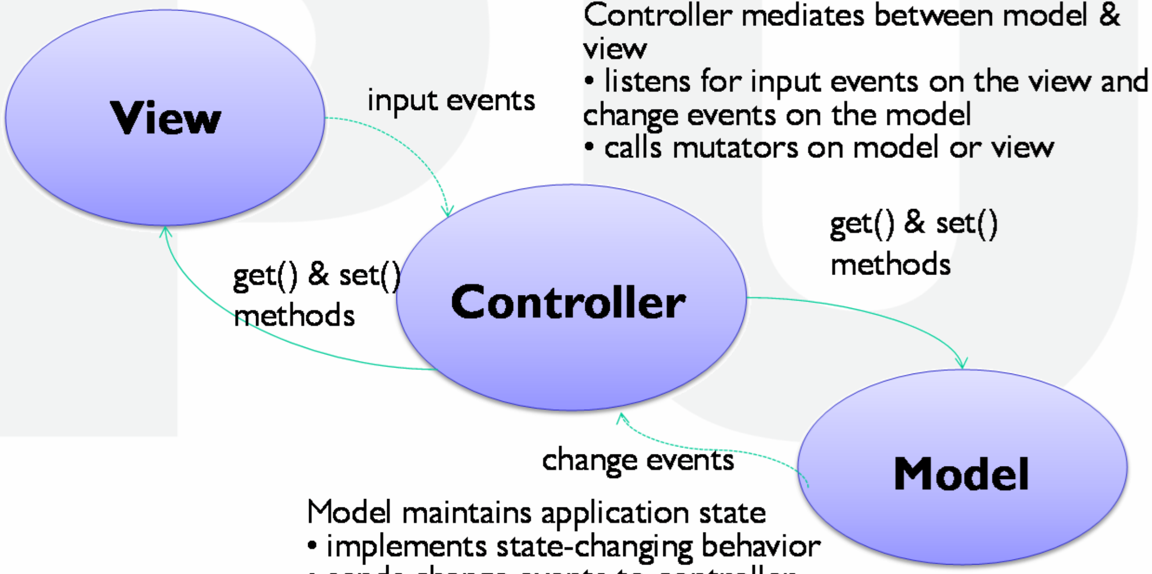
```
import javax.swing.*;
public class Simple {
JFrame f;
Simple() {
f=new JFrame();
JButton b=new Jbutton ("click");
b.setBounds(130,100,100,  40);

f.add(b);

f.setSize(400,500);
f.setLayout(null);

f.setVisible(true);
}
public static void main (String[] args)
{
new Simple();
}  }
```

Example : Swing by inheritance

```
import javax.swing.*;
public class Simple2 extends JFrame {
JFrame f;
Simple2() {
JButton b=new Jbutton ("click");
b.setBounds(130,100,100, 40);
add(b);
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main (String[] args)
{
new Simple2();
}
}
```

Model-View-Controller

- Swing uses the model-view-controller architecture (MVC) as the fundamental design behind each of its components.



View handles output & low-level input
• sends high-level events to the controller

Controller mediates between model & view
• listens for input events on the view and change events on the model
• calls mutators on model or view

Model maintains application state
• implements state-changing behavior
• sends change events to controller

Layout manager

- 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:
  i. java.awt.BorderLayout
  ii. java.awt.FlowLayout
  iii. java.awt.GridLayout
  iv. java.awt.CardLayout
  v. java.awt.GridBagLayout
  vi. javax.swing.BoxLayout

Java FlowLayout

- **Fields of FlowLayout class**

i. public static final int **LEFT**
ii. public static final int **RIGHT**
iii. public static final int **CENTER**
iv. public static final int **LEADING**
v. public static final int **TRAILING**

- **Constructors of FlowLayout class**

i. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.

ii. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.

iii. **FlowLayout(int align, int hgap, int vgap)**: creates a flow layout with the given alignment and the given horizontal and vertical gap

Example : FlowLayout

```
import java.awt.*;
import javax.swing.*;
public class MyFlowLayout {
JFrame f;
MyFlowLayout()
{
   f=new JFrame();
   f.setLayout(new FlowLayout(FlowLayout
.RIGHT));
  JButton b1=new JButton("1");
   JButton b2=new JButton("2");
   JButton b3=new JButton("3");
   JButton b4=new JButton("4");
   JButton b5=new JButton("5");

   f.add(b1);f.add(b2);f.add(b3);
   f.add(b4);f.add(b5);

   f.setSize(300,300);
   f.setVisible(true);
}
public static void main(String[] args)
{
   new MyFlowLayout();
} }
```

Java BorderLayout

- **Fields of BorderLayout class**

i. public static final int **NORTH**

ii. public static final int **SOUTH**

iii. public static final int **EAST**

iv. public static final int **WEST**

v. public static final int **CENTER**

- **Constructors of BorderLayout class**

i. **BorderLayout():** creates a border layout but with no gaps between the components.

ii. **BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components

Example : BorderLayout

```
import java.awt.*;
import javax.swing.*;
public class BorderDemo {
        JFrame f;
BorderDemo()
 {
   f=new JFrame();
   f.setLayout(new BorderLayout());
   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  BorderDemo();
} }
```

Java BoxLayout

- **Fields of BoxLayout class**

i. public static final int **X_AXIS**

ii. public static final int **Y_AXIS**

- **Constructors of BoxLayout class**

i. BoxLayout(Container c, int axis):

creates a box layout that arranges the components with the given axis

Example : BoxLayout

```
import java.awt.*;
import javax.swing.*;
public class BoxDemo extends  Frame {
Button buttons[];

public  BoxDemo  () {
   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, BoxLayou
t.Y_AXIS));
setSize(400,400);
setVisible(true);
}

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

Java GridLayout

- The **GridLayout** is used to arrange the components in rectangular grid.

- One component is displayed in each rectangle.

- **Constructors of GridLayout class:**
  1. **GridLayout():** creates a grid layout with one column per component in a row.
  2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
  3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Example : GridLayout

```
import java.awt.*;
import javax.swing.*;
public class GridDemo {
JFrame f;
GridDemo() {
f=new JFrame();

   JButton b1=new JButton("1");
   JButton b2=new JButton("2");
   JButton b3=new JButton("3");
   JButton b4=new JButton("4");
   f.add(b1); f.add(b2); f.add(b3);
f.add(b4);
f.setLayout (new GridLayout(2,2));
   //setting grid layout of 2 rows and 2 columns

   f.setSize(300,300);
   f.setVisible(true);
}
public static void main(String[] args)
{
   new GridDemo();
}}
```

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in