UNIT 5 AWT

ABSTRACT WINDOW TOOLKIT)

TOPIC

1. INTRODUCTION TO AWT
2. AWT CONTROLS - BUTTONS, CHECKBOX, CHOICE, LIST AND TEXTFIELD.
3. LAYOUT MANAGERS - FLOW LAYOUT, GRID LAYOUT AND BORDER LAYOUT.
4. USER INTERFACE EVENTS - EVENT CLASSES AND EVENT LISTENER INTERFACES, ADAPTER CLASSES.

JAVA AWT

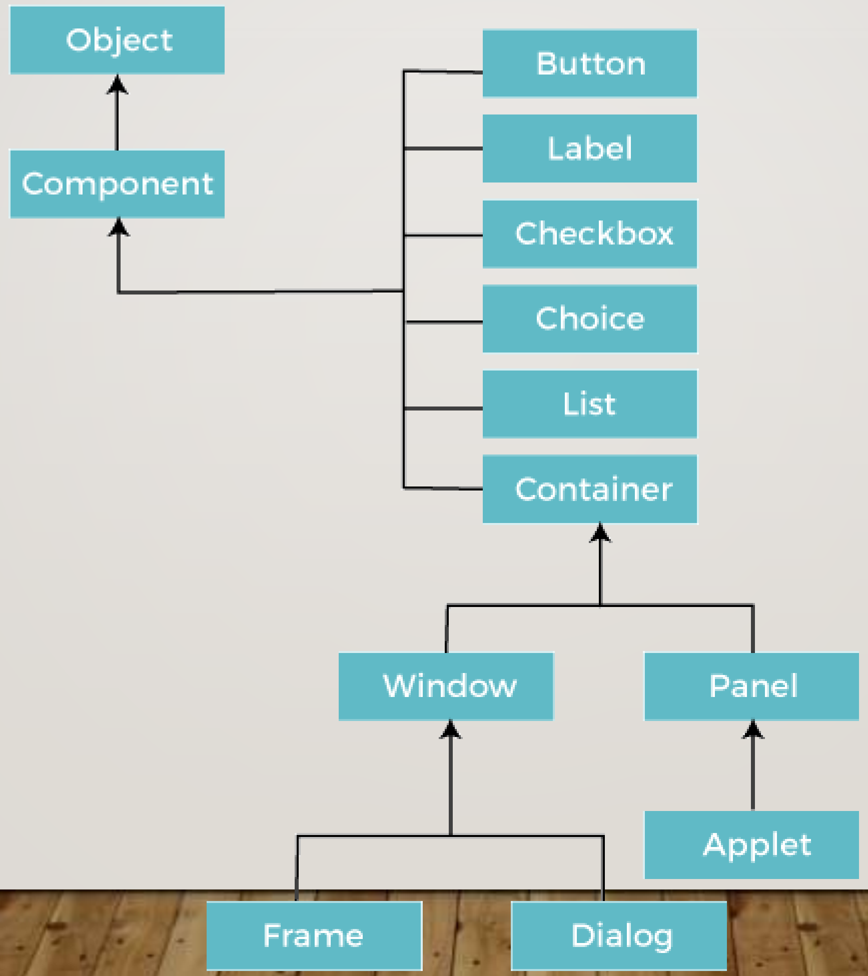
- **Java AWT** (Abstract Window Toolkit) is *an API to develop Graphical User Interface (GUI) or windows-based applications* in Java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).
- The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

WHY AWT IS PLATFORM DEPENDENT?

- Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, ChechBox, button, etc.
- For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.
- In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

JAVA AWT HIERARCHY

- The hierarchy of Java AWT classes are given below.



Components

- All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

Container

- The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as **Frame**, **Dialog** and **Panel**.
- It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

TYPES OF CONTAINERS:

1.There are four types of containers in Java AWT: Window ,Panel , Frame , Dialog

Window

- The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window. We need to create an instance of Window class to create this container.

Panel

- The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

Frame

- The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

USEFUL METHODS OF COMPONENT CLASS

Method	Description
public void add(Component c)	Inserts a component on this component.
public void setSize(int width,int height)	Sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	Defines the layout manager for the component.
public void setVisible(boolean status)	Changes the visibility of the component, by default false.

JAVA AWT EXAMPLE

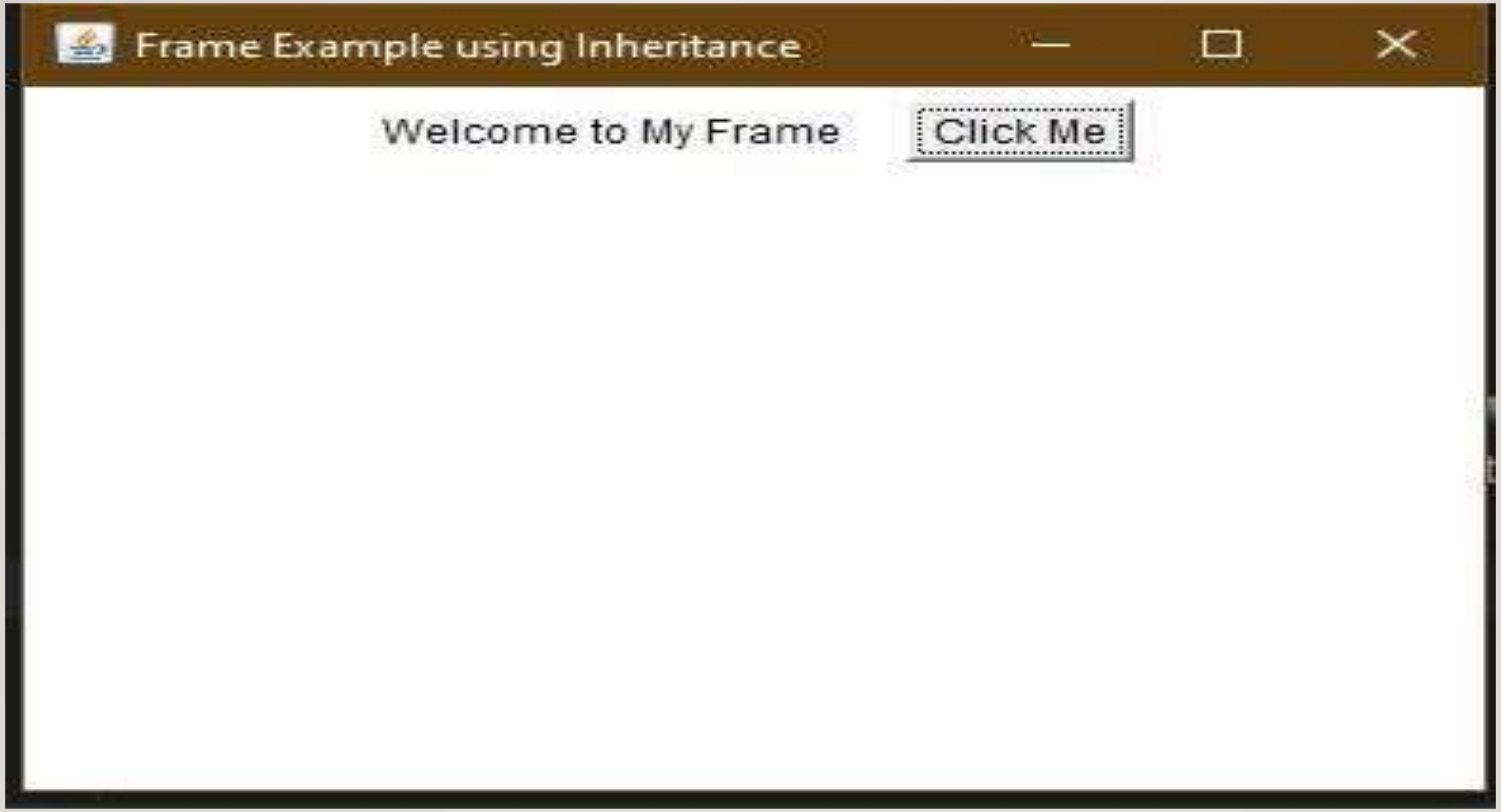
- To create simple AWT example, you need a frame. There are two ways to create a GUI using Frame in AWT.

1. By extending Frame class (**inheritance**)
2. By creating the object of Frame class (**association**)

AWT EXAMPLE BY INHERITANCE

```
import java.awt.*;

public class MyFrame extends Frame {
    public MyFrame() {
        // Set the frame title
        super("Frame Example using Inheritance");

        // Set size and layout
        setSize(400, 300);
        setLayout(new FlowLayout());

        // Add components
        Label label = new Label("Welcome to My Frame");
        Button button = new Button("Click Me");
add(label);
        add(button);

        // Add a close button functionality
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent e) {
                System.exit(0);
            }
        });
    }

    public static void main(String[] args) {
        MyFrame frame = new MyFrame();
        frame.setVisible(true);
    }
}
```

O/P 1

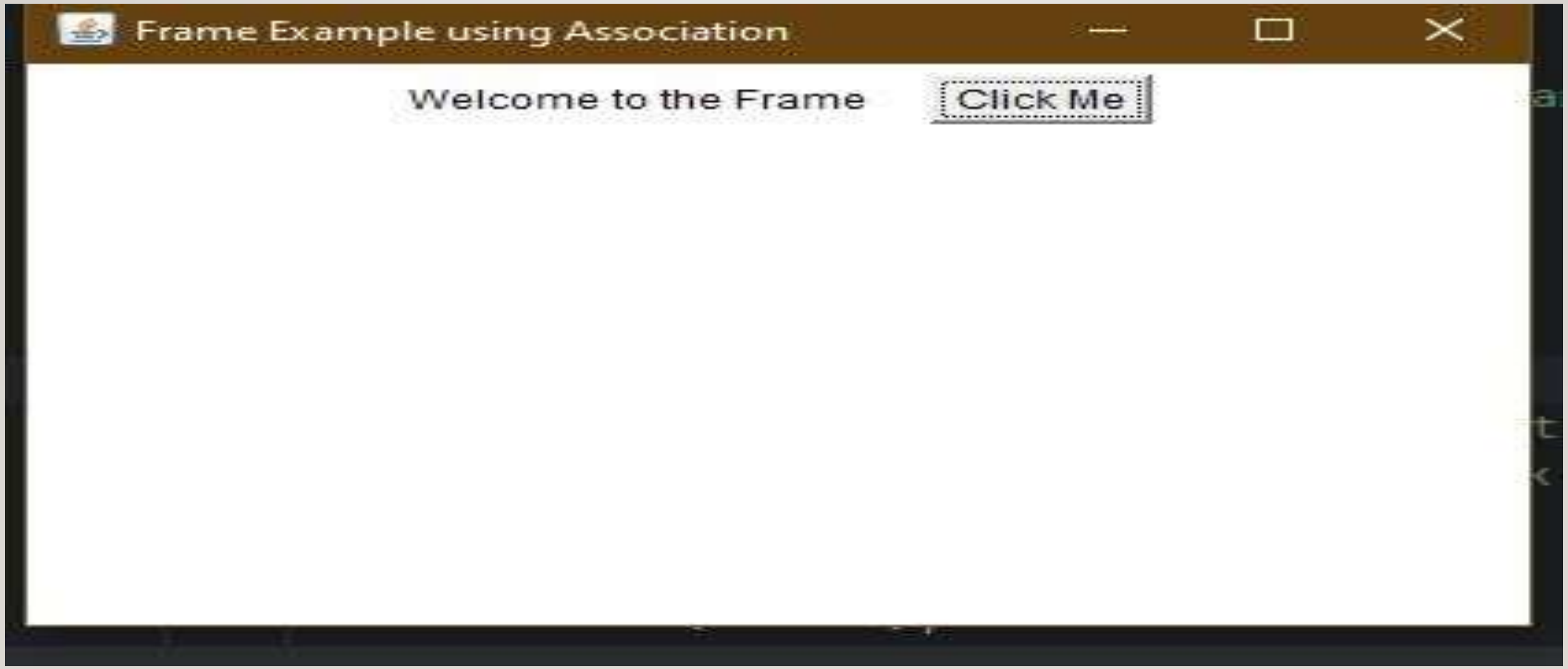

Frame Example using Inheritance

Welcome to My Frame

2. BY CREATING AN OBJECT OF THE FRAME CLASS (ASSOCIATION)

```
import java.awt.*;

public class FrameWithAssociation {
    public FrameWithAssociation() {
        // Create a Frame object
        Frame frame = new Frame("Frame Example using Association");

        // Set size and layout
        frame.setSize(400, 300);
        frame.setLayout(new FlowLayout());

        // Add components
        Label label = new Label("Welcome to the Frame");
        Button button = new Button("Click Me");
frame.add(label);
        frame.add(button);

        // Add a close button functionality
        frame.addWindowListener(new
java.awt.event.WindowAdapter() {
            public void
windowClosing(java.awt.event.WindowEvent e) {
                System.exit(0);
            }
        });

        // Make the frame visible
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        new FrameWithAssociation();
    }
}
```

O/P 2


Frame Example using Association
Welcome to the Frame
Click Me

Parul® University

EVENT AND LISTENER (JAVA 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 Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

STEPS TO PERFORM EVENT HANDLING

- Following steps are required to perform event handling:

1. Register the component with the Listener

- Registration Methods
- For registering the component with the Listener, many classes provide the registration methods. For example:
- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **TextArea**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**

JAVA EVENT HANDLING CODE

- We can put the event handling code into one of the following places:

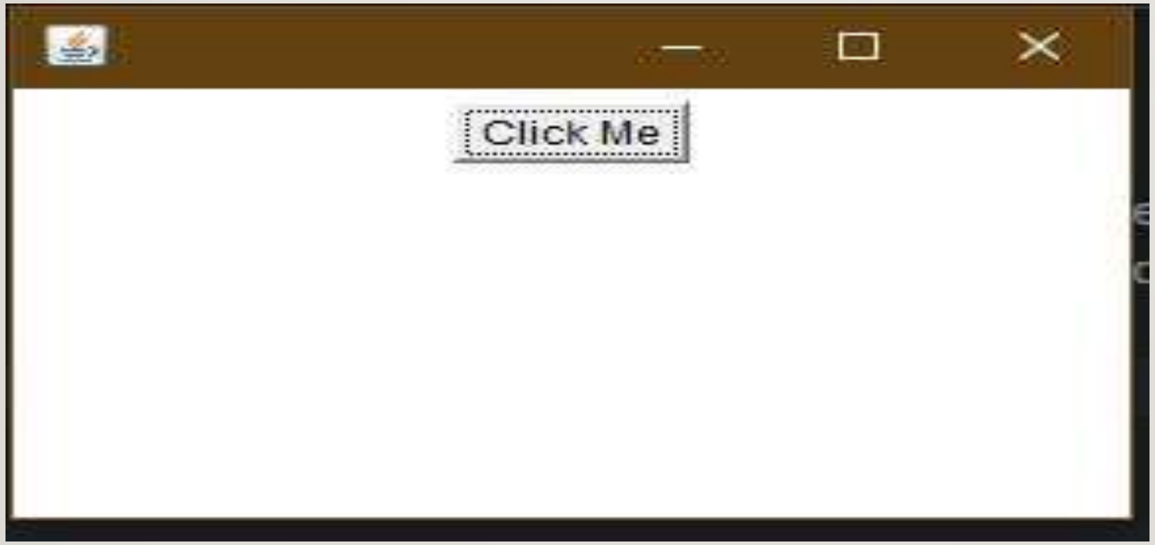
1. Within class
2. Other class
3. Anonymous class

1. Within the Same Class

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

public class EventHandlingWithinClass extends Frame implements ActionListener {
    Button button;

    public EventHandlingWithinClass() {
        // Frame properties
        setLayout(new FlowLayout());
        setSize(300, 200);

        // Create a Button
        button = new Button("Click Me");
        add(button);
```

```
        // Add ActionListener to the button
        button.addActionListener(this);

        // Close the window
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        // Event Handling Logic
        button.setLabel("Clicked!");
    }

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

o/p within class

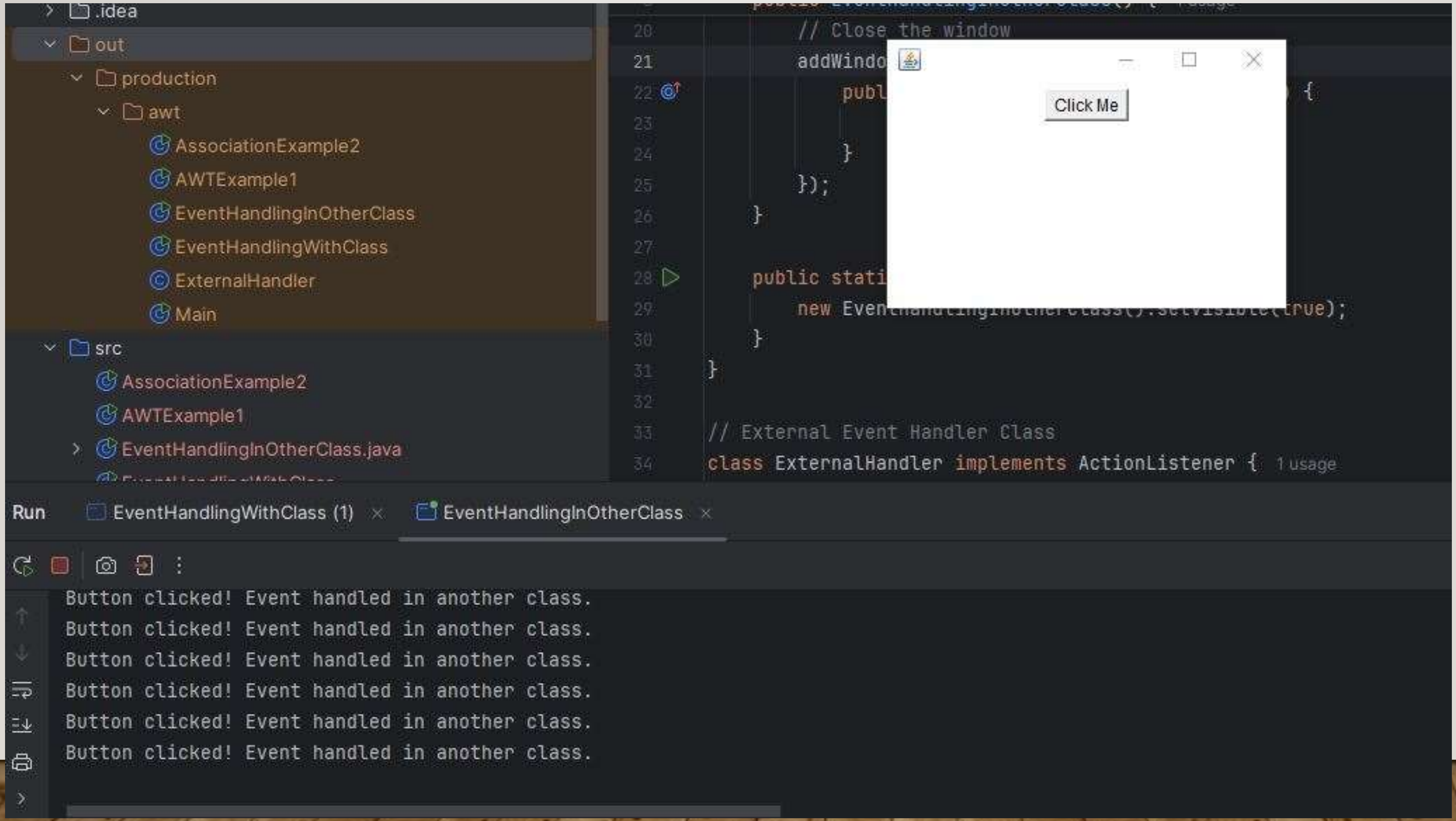


2. IN ANOTHER CLASS (DELEGATION)

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

// Main Class
public class EventHandlingInOtherClass extends Frame {
    Button button;

    public EventHandlingInOtherClass() {
        // Frame properties
        setLayout(new FlowLayout());
        setSize(300, 200);

        // Create a Button
        button = new Button("Click Me");
        add(button);
```

```
        // Add ActionListener from another class
        button.addActionListener(new ExternalHandler());

        // Close the window
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

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

// External Event Handler Class
class ExternalHandler implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button clicked! Event handled in another class.");
    }
}
```

O/P



```
> .idea
v out
  v production
    v awt
        AssociationExample2
        AWTExample1
        EventHandlingInOtherClass
        EventHandlingWithClass
        ExternalHandler
        Main
v src
    AssociationExample2
    AWTExample1
  > EventHandlingInOtherClass.java
```

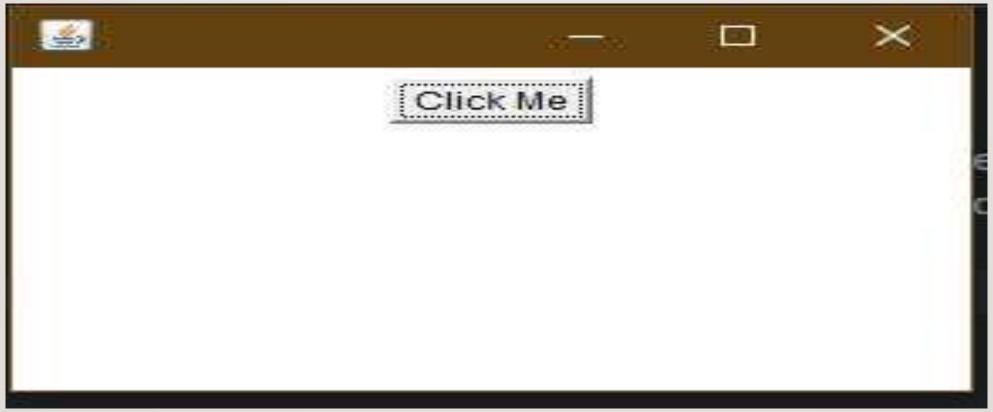
```
        // Close the window
        addWindo
            publ                                    ) {

            }
        });
    }

    public stati
        new Even                              true);
    }
}

// External Event Handler Class
class ExternalHandler implements ActionListener {  1 usage
```

```
Button clicked! Event handled in another class.
Button clicked! Event handled in another class.
Button clicked! Event handled in another class.
Button clicked! Event handled in another class.
Button clicked! Event handled in another class.
Button clicked! Event handled in another class.
```

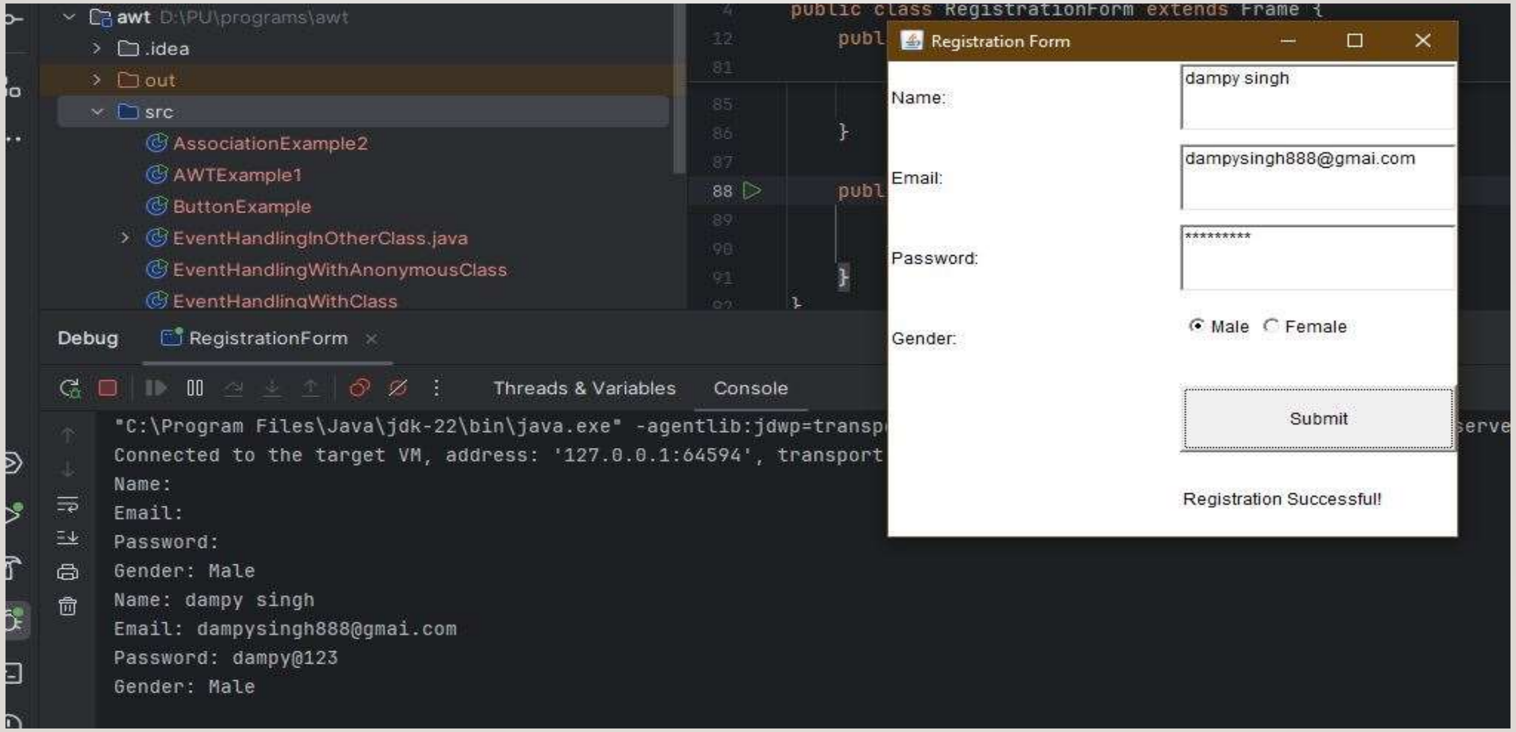
3. USING AN ANONYMOUS CLASS

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

public class EventHandlingWithAnonymousClass extends Frame {
    public EventHandlingWithAnonymousClass() {
        // Frame properties
        setLayout(new FlowLayout());
        setSize(300, 200);

        // Create a Button
        Button button = new Button("Click Me");
        add(button);

        // Add ActionListener using an Anonymous Class
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
button.setLabel("Clicked!");
            }
        });

        // Close the window
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

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

O/P



REGISTARTION FORM



```
public class RegistrationForm extends Frame {
    publ
    }
    publ
    }
}
```

Registration Form

Name:	dampy singh
Email:	dampysingh888@gmai.com
Password:	*********
Gender:	[✓ Male] Female
	Submit
	Registration Successful!

```
"C:\Program Files\Java\jdk-22\bin\java.exe" -agentlib:jdwp=transp
Connected to the target VM, address: '127.0.0.1:64594', transport
Name:
Email:
Password:
Gender: Male
Name: dampy singh
Email: dampysingh888@gmai.com
Password: dampy@123
Gender: Male
```

JAVA AWT BUTTON

- A button is basically a control component with a label that generates an event when pushed. 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.
- **AWT Button Class Declaration**
- public **class** Button **extends** Component **implements** Accessible

JAVA AWT TEXTFIELD

- The object of a **TextField** class is a text component that allows a user to enter a single line text and edit it. It inherits **TextComponent** class, which further inherits **Component** class.
- When we enter a key in the text field (like key pressed, key released or key typed), the event is sent to **TextField**. Then the **KeyEvent** is passed to the registered **KeyListener**. It can also be done using **ActionEvent**; if the ActionEvent is enabled on the text field, then the ActionEvent may be fired by pressing return key. The event is handled by the **ActionListener** interface.
- **AWT TextField Class Declaration**
- public **class** TextField **extends** TextComponent

LAYOUT MANAGERS IN JAVA AWT

- The LayoutManagers are used to arrange components in a particular manner. The **Java LayoutManagers** facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers. There are the following classes that represent the layout managers:

1. FlowLayout

Overview:
•Arranges components in a line, one after another (like words in a paragraph).
•Automatically wraps components to the next line when out of horizontal space.
Key Features:
•Alignment: FlowLayout.LEFT, FlowLayout.CENTER (default), FlowLayout.RIGHT.
•Horizontal and vertical gaps between components can be specified.
•Suitable for simple layouts with a few components.

2. GRIDLAYOUT

- **Overview:**
- Divides the container into a grid of rows and columns.
- All cells in the grid are of equal size.
- Components are added row by row, left to right.
- **Key Features:**
- The grid expands to fit the container's size.
- Rows or columns can be set to zero for dynamic adjustments.
- Equal space distribution among components.

3. BorderLayout

Overview:

•Divides the container into five regions: NORTH, SOUTH, EAST, WEST, and CENTER.

•Each region can contain only one component.

•The CENTER region expands to fill the remaining space.

Key Features:

•Default layout for Frame.

•Regions can be left empty if not needed.

•Components in CENTER region stretch to occupy all available space.

USER INTERFACE EVENTS IN AWT

In Java's Abstract Window Toolkit (AWT), user interface (UI) events are the core mechanism for handling user interactions such as mouse clicks, key presses, and other input actions. These events are managed using **Event Classes**, **Event Listener Interfaces**, and **Adapter Classes**.

- **1. Event Classes**

- Event classes represent different types of user interactions. They are part of the java.awt.event package and are used to encapsulate event information.

1. EVENT CLASSES

In Java's Abstract Window Toolkit (AWT), user interface (UI) events are the core mechanism for handling user interactions such as mouse clicks, key presses, and other input actions. These events are managed using **Event Classes**, **Event Listener Interfaces**, and **Adapter Classes**.

Event Class	Description
ActionEvent	Generated when a button is clicked, or an item in a menu is selected.
MouseEvent	Represents mouse actions like clicks, movement, and drags.
KeyEvent	Represents keyboard actions like key presses and releases.
WindowEvent	Represents actions on a window, such as opening, closing, or minimizing.
FocusEvent	Indicates a change in the focus state of a component.
ItemEvent	Triggered when an item is selected or deselected (e.g., in a checkbox or list).
AdjustmentEvent	Represents changes to an adjustable component, such as a scrollbar.
TextEvent	Represents changes in a text component like a TextField or TextArea.
ComponentEvent	Represents changes in a component's state, such as resizing or movement.
ContainerEvent	Indicates when components are added or removed from a container.

2. EVENT LISTENER INTERFACES

Event listeners are interfaces that define methods to handle specific events. Components register these listeners to listen for events.

Listener Interface	Method(s)	Description
ActionListener	actionPerformed(ActionEvent e)	Handles action events (e.g., button clicks).
MouseListener	mouseClicked, mouseEntered, mouseExited, mousePressed, mouseReleased	Handles mouse events like clicks and entry/exit.
MouseMotionListener	mouseDragged, mouseMoved	Handles mouse motion events.
KeyListener	keyPressed, keyReleased, keyTyped	Handles keyboard events.
WindowListener	windowActivated, windowClosed, windowClosing, etc.	Handles window events.
FocusListener	focusGained, focusLost	Handles focus events.
ItemListener	itemStateChanged(ItemEvent e)	Handles item selection/deselection events.
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent e)	Handles adjustments in components like scrollbars.
TextListener	textValueChanged(TextEvent e)	Handles text changes in text components.
ComponentListener	componentMoved, componentResized, componentShown, componentHidden	Handles component-level changes.
ContainerListener	componentAdded, componentRemoved	Handles addition or removal of components from containers.

3. ADAPTER CLASSES

Adapter classes provide default (empty) implementations of listener methods. They are used to simplify the implementation of listener interfaces when not all methods are needed.

Adapter Class	Implements	Purpose
MouseAdapter	MouseListener, MouseMotionListener	Provides default methods for mouse events.
KeyAdapter	KeyListener	Provides default methods for keyboard events.
WindowAdapter	WindowListener	Provides default methods for window events.
FocusAdapter	FocusListener	Provides default methods for focus events.
ComponentAdapter	ComponentListener	Provides default methods for component-level events.
ContainerAdapter	ContainerListener	Provides default methods for container events.

Thank you

UNIT 5 SWING

SWING

JAVA SWING

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.

JAVA SWING, A GRAPHICAL USER INTERFACE (GUI) TOOLKIT, HAS BEEN A CORNERSTONE OF JAVA DEVELOPMENT FOR DECADES. SINCE ITS INCEPTION, SWING HAS PROVIDED JAVA DEVELOPERS WITH A ROBUST FRAMEWORK FOR CREATING INTERACTIVE, PLATFORM-INDEPENDENT APPLICATIONS. IN THIS ARTICLE, WE'LL DELVE INTO THE FUNDAMENTALS OF JAVA SWING, EXPLORE ITS KEY FEATURES, AND DISCUSS ITS RELEVANCE IN MODERN APPLICATION DEVELOPMENT.

KEY FEATURES OF JAVA SWING

PLATFORM INDEPENDENCE: ONE OF THE PRIMARY ADVANTAGES OF SWING IS ITS PLATFORM INDEPENDENCE. APPLICATIONS DEVELOPED USING SWING CAN RUN ON ANY PLATFORM THAT SUPPORTS JAVA, WITHOUT REQUIRING MODIFICATIONS.

RICH SET OF COMPONENTS: SWING OFFERS A WIDE RANGE OF COMPONENTS THAT CAN BE USED TO CREATE COMPLEX GUIS. DEVELOPERS CAN CHOOSE FROM BASIC COMPONENTS LIKE BUTTONS AND LABELS TO ADVANCED COMPONENTS SUCH AS TABLES, TREES, AND SCROLL PANES.

CUSTOMIZABILITY: ~~SWING COMPONENTS ARE HIGHLY CUSTOMIZABLE, ALLOWING DEVELOPERS TO CONTROL VARIOUS~~ ASPECTS OF THEIR APPEARANCE AND BEHAVIOR. PROPERTIES SUCH AS SIZE, COLOR, FONT, AND LAYOUT CAN BE EASILY ADJUSTED TO MEET SPECIFIC DESIGN REQUIREMENTS.

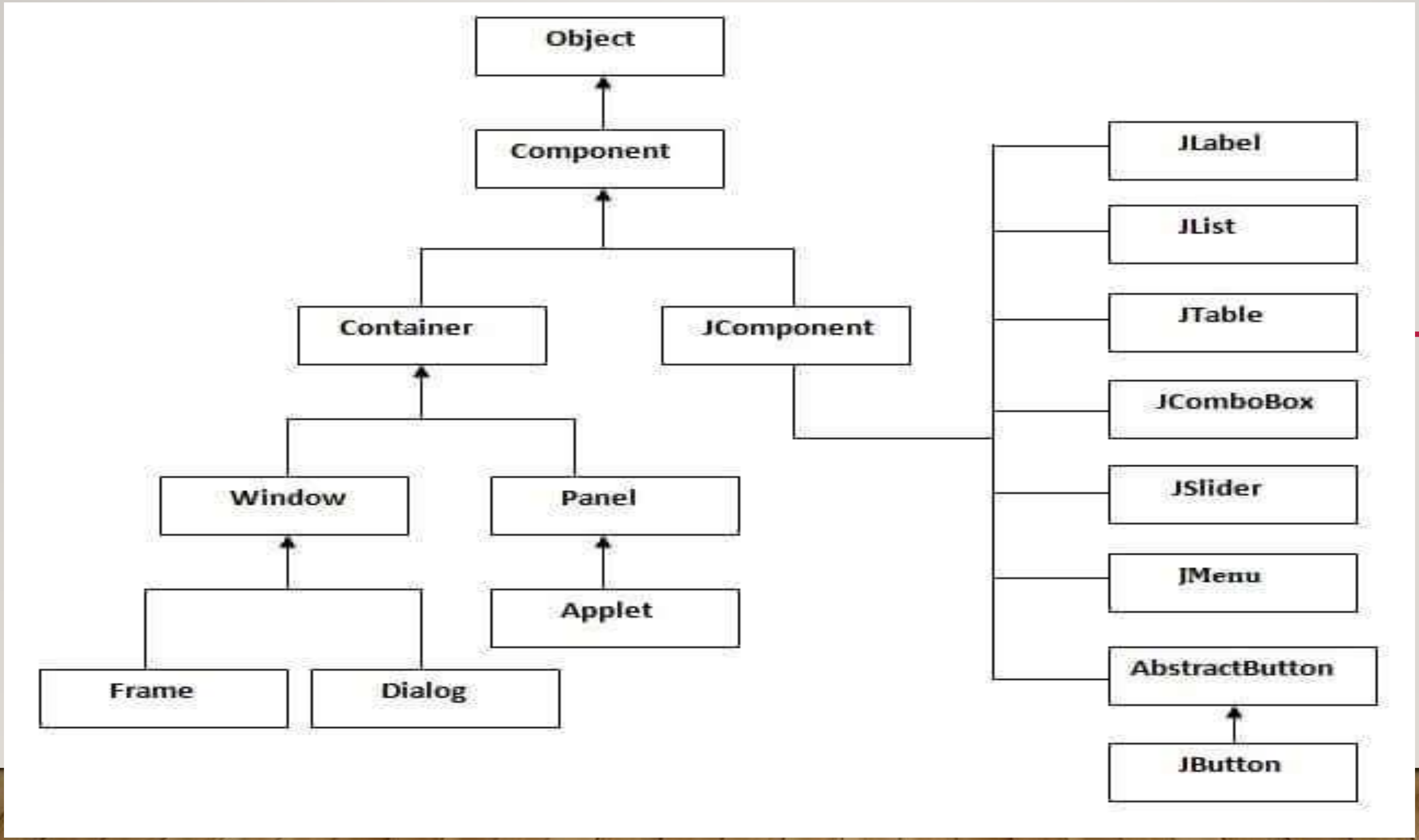
EVENT HANDLING: SWING PROVIDES A ROBUST EVENT HANDLING MECHANISM THAT ALLOWS DEVELOPERS TO RESPOND TO USER INTERACTIONS, SUCH AS BUTTON CLICKS AND MOUSE MOVEMENTS. THIS ENABLES THE CREATION OF INTERACTIVE AND RESPONSIVE APPLICATIONS.

LAYOUT MANAGERS: SWING INCLUDES A SET OF LAYOUT MANAGERS THAT FACILITATE THE ARRANGEMENT OF COMPONENTS WITHIN A CONTAINER. LAYOUT MANAGERS AUTOMATICALLY ADJUST THE POSITION AND SIZE OF COMPONENTS BASED ON THE CONTAINER'S SIZE, ENSURING CONSISTENT BEHAVIOR ACROSS DIFFERENT SCREEN RESOLUTIONS AND DEVICES.

DIFFERENCE BETWEEN AWT AND SWING

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	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**.

HIERARCHY OF JAVA SWING CLASSES



COMMONLY USED METHODS OF COMPONENT CLASS

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

JAVA SWING EXAMPLES

HERE ARE TWO WAYS TO CREATE A FRAME:

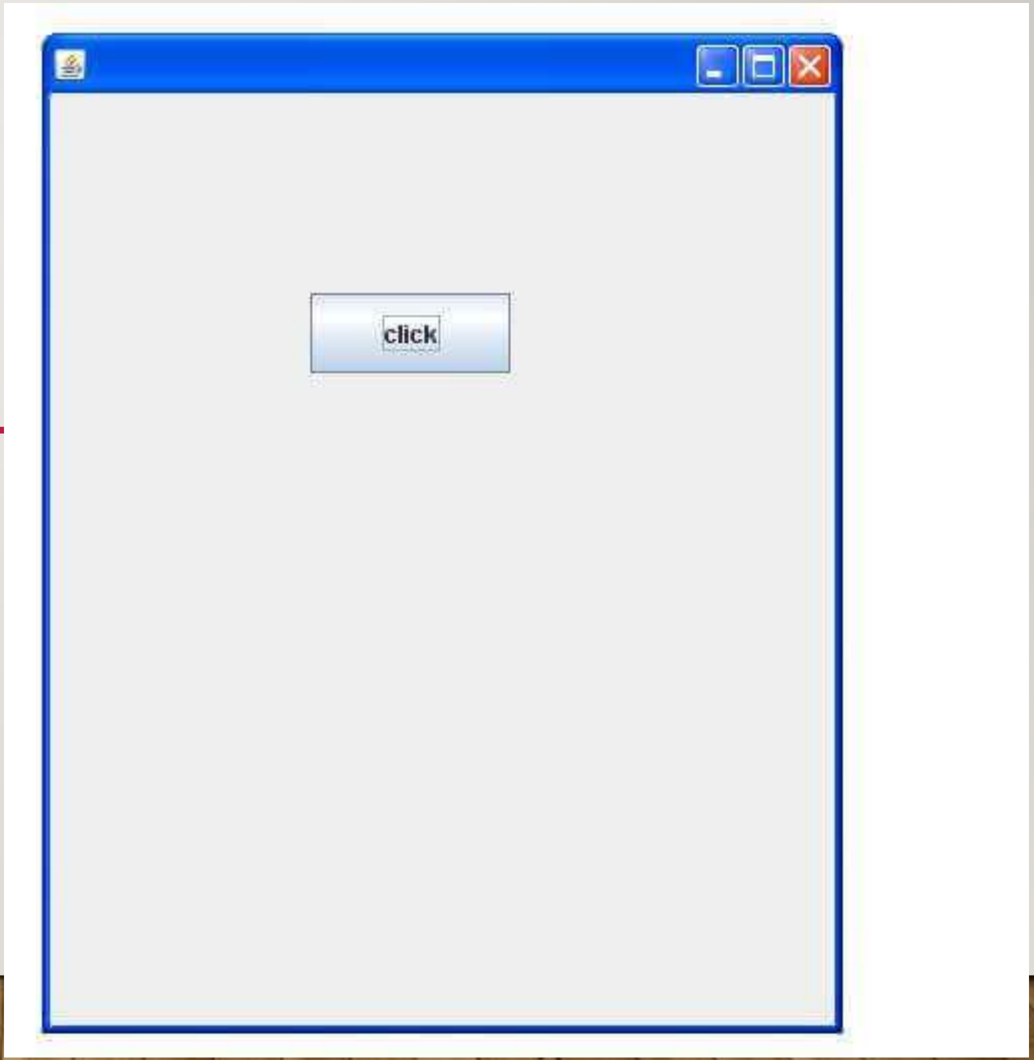
- BY CREATING THE OBJECT OF FRAME CLASS (ASSOCIATION)
- BY EXTENDING FRAME CLASS (INHERITANCE)

WE CAN WRITE THE CODE OF SWING INSIDE THE MAIN(), CONSTRUCTOR OR ANY OTHER METHOD.

JAVA SWING EXAMPLE

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

```
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 OF SWING BY ASSOCIATION INSIDE CONSTRUCTOR

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

```
IMPORT JAVAX.SWING.*;
PUBLIC CLASS DEMO {
JFRAME F;
SIMPLE(){
F=NEW JFRAME();//CREATING INSTANCE OF JFRAME
JBUTTON B=NEW JBUTTON("CLICK");//CREATING INSTANCE OF JBUTTON
B.SETBOUNDS(130,100,100, 40);
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
}
PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
NEW DEMO();
}
}
```

The setBounds(int xaxis, int yaxis, int width, int height)is used in the above example that sets the position of the button.

EXAMPLE OF SWING BY INHERITANCE

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

```
IMPORT JAVAX.SWING.*;
PUBLIC CLASS DEMOSWING EXTENDS JFRAME{//INHERITING JFRAME
JFRAME F;
SIMPLE2(){
JBUTTON B=NEW JBUTTON("CLICK");//CREATE BUTTON
B.SETBOUNDS(130,100,100, 40);
ADD(B);//ADDING BUTTON ON FRAME
SETSIZE(400,500);
SETLAYOUT(NULL);
SETVISIBLE(TRUE);
}
PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
NEW SIMPLE2();
}}
```

Thank you

UNIT 5 APPLET

APPLET

JAVA APPLET

Advantage of applet

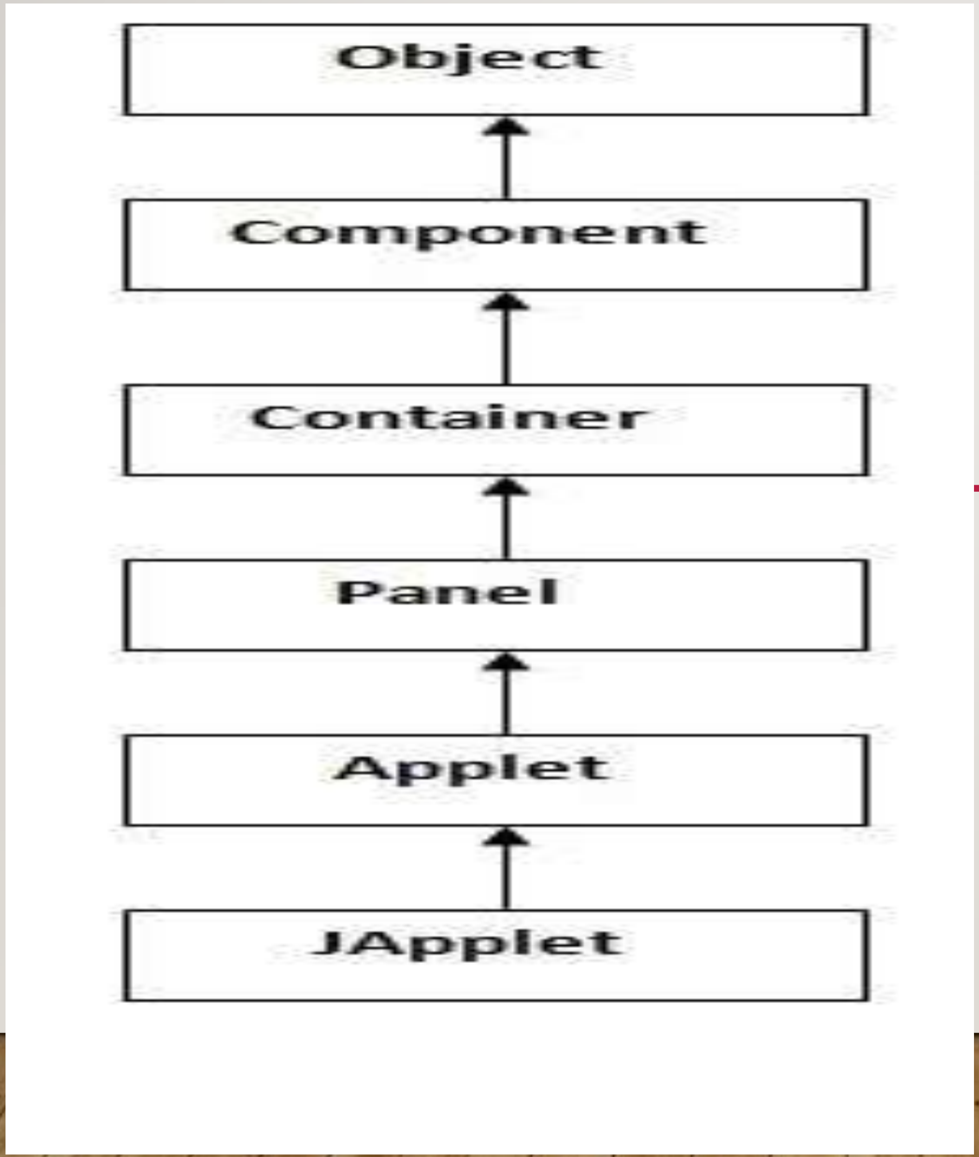
There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including linux, windows, mac os etc.

DRAWBACK OF APPLET

- Plugin is required at client browser to execute applet.

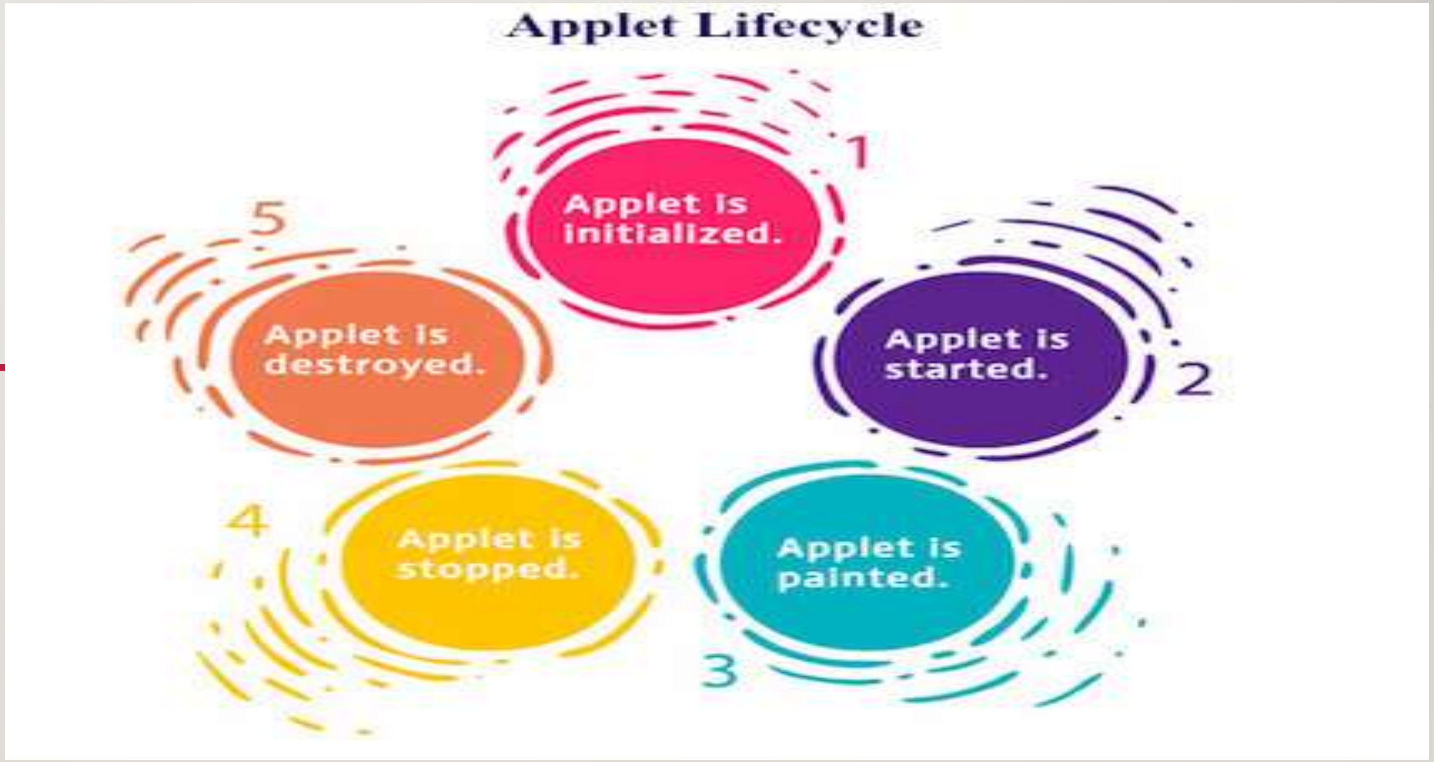
HIERARCHY OF APPLET



As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

LIFECYCLE OF JAVA APPLET

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.



LIFECYCLE METHODS FOR APPLET:

The java.Applet.Applet class 4 life cycle methods and java.Awt.Component class provides 1 life cycle methods for an applet.

Java.Applet.Applet class

For creating any applet java.Applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **PUBLIC VOID INIT():** IS USED TO INITIALIZED THE APPLET. IT IS INVOKED ONLY ONCE.
2. **PUBLIC VOID START**~~**()**: IS INVOKED AFTER THE INIT() METHOD OR BROWSER IS MAXIMIZED. IT IS~~ USED TO START THE APPLET.
3. **PUBLIC VOID STOP():** IS USED TO STOP THE APPLET. IT IS INVOKED WHEN APPLET IS STOP OR BROWSER IS MINIMIZED.
4. **PUBLIC VOID DESTROY():** IS USED TO DESTROY THE APPLET. IT IS INVOKED ONLY ONCE.

JAVA.AWT.COMPONENT CLASS

The component class provides 1 life cycle method of applet.

1. **Public void paint(graphics g):** is used to paint the applet. It provides graphics class object that can be used for drawing oval, rectangle, arc etc.

HOW TO RUN AN APPLET?

There are two ways to run an applet

1. By html file.
2. By appletviewer tool (for testing purpose).

SIMPLE EXAMPLE OF APPLET BY HTML FILE:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
//FIRST.JAVA
IMPORT JAVA.APPLET.APPLET;
IMPORT JAVA.AWT.GRAPHICS;
PUBLIC CLASS FIRST EXTENDS APPLET{

PUBLIC VOID PAINT(GRAPHICS G){
G.DRAWSTRING("WELCOME",150,150);
}

}
```

```
<html>
<body>
<applet code="First.class" width="300" heigh
t="300">
</applet>
</body>
</html>
```

Note: *class must be public because its object is created by Java Plugin software that resides on the browser.*

SIMPLE EXAMPLE OF APPLET BY APPLETVIEWER TOOL:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer first.Java. Now html file is not required but it is for testing purpose only.

```
IMPORT JAVA.APPLET.APPLET;
IMPORT JAVA.AWT.GRAPHICS;
PUBLIC CLASS FIRST EXTENDS APPLET{

PUBLIC VOID PAINT(GRAPHICS G){
G.DRAWSTRING("WELCOME TO APPLET",150,150);
}

}
/*
<APPLET CODE="FIRST.CLASS" WIDTH="300" HEIGHT="300">
</APPLET>
*/
```

To execute the applet by appletviewer tool, write in command prompt

```
c:\>javac First.java
c:\>appletviewer First.java
```

DISPLAYING GRAPHICS IN APPLET

Commonly used methods of graphics class:

1.**Public abstract void drawstring(string str, int x, int y):** is used to draw the specified string.

2.**Public void drawrect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.

3.**Public abstract void fillrect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.

4.**Public abstract void drawoval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.

5.**Public abstract void filloval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.

6.**Public abstract void drawline(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).

7.**Public abstract boolean drawimage(image img, int x, int y, imageobserver observer):** is used draw the specified image.

8.**Public abstract void drawarc(int x, int y, int width, int height, int startangle, int arcangle):** is used draw a circular or elliptical arc.

9.**Public abstract void fillarc(int x, int y, int width, int height, int startangle, int arcangle):** is used to fill a circular or elliptical arc.

10.**Public abstract void setcolor(color c):** is used to set the graphics current color to the specified color.

EXAMPLE OF GRAPHICS IN APPLET:

```
IMPORT JAVA.APPLET.APPLET;
IMPORT JAVA.AWT.*;

PUBLIC CLASS GRAPHICSDEMO EXTENDS APPLET{

PUBLIC VOID PAINT(GRAPHICS G){
G.SETCOLOR(COLOR.RED);
G.DRAWSTRING("WELCOME",50, 50);
G.DRAWLINE(20,30,20,300);
G.DRAWRECT(70,100,30,30);
G.FILLRECT(170,100,30,30);
G.DRAWOVAL(70,200,30,30);

G.SETCOLOR(COLOR.PINK);
G.FILLOVAL(170,200,30,30);
G.DRAWARC(90,150,30,30,30,270);
G.FILLARC(270,150,30,30,0,180);

}
}
```

```
<html>
<body>
<applet code="GraphicsDemo.class" width="300" height="300">
</applet>
</body>
</html>
```

THANK YOU!