CS6308
JAVA PROGRAMMING

V P Jayachitra

Syllabus

MODULE IV GUI	L	T	P	EL
	3	0	4	3
Creating UI, Frames, layout manager, Panels, components, Event Driven Programming				
SUGGESTED ACTIVITIES : • flipped classroom • Practical – Mouse, key events, creating interactive forms using AWT/Swing and adding functionality • EL – Understand AWT and SWING				

Event-Driven Programming

- **Definition:** Event-driven programming is an **event-driven paradigm** in which the flow of the program is determined by events such as **mouse click, key pressed or sensors output or message** from other programs or threads.

- **Program control flow is determined by events or sensor input or user actions or messages**

- **Event driven is a asynchronous process**
  - since components don't communicate directly, or waiting for a response from each other.
  - The services that participate in the communication, simply publish a message and then moves on. Whoever is interested can listen to the event, whenever is useful for them.

- **Asynchronous** is about not blocking threads of execution and it's often more efficient use of resources. It helps minimize congestion on shared resources in the system, this improves scalability, low latency, and high throughput.

Event-Driven Programming

- Event-driven programs do not control the sequence in which input events occur; instead, they are written to react to any reasonable sequence of events.

- Events also do not occur in any particular or predictable order.

- In this control model, the input data govern the particular sequence of operations that is carried out by the program, but the program does not determine the input sequence order.

- Moreover, an event-driven program is usually designed to run indefinitely, or at least until the user selects an exit button.

- Example of an event-driven program:
  - **Graphical user interface (GUI)**
    - Used in desktop , laptop computers and smartphones
  - **Web-based applications**
    - online student registration system, online airline reservation system
  - **Embedded applications**
    - cell phones, car engines, airplane navigation and home security systems

- **Model-view-controller (MVC) design pattern** provide effective support for event-driven programming
  - Languages that support MVC framework or design pattern are Java, Visual Basic, Tcl/Tk, JavaScript, Python, Perl, Ruby, PHP, C#, Swift

EVENT-DRIVEN CONTROL

- **How Event driven programming differs from Traditional view ?**
  - **In general**, statements are **executed in sequential**, from the entry point in main, according to which conditions occur during execution.
  - However, **GUIs** follow a style of programming called event-driven programming must be able to execute statements **according to some events** that may occur at any time while the program in execution.
  - According to Stein (1999), Computation is a function from its input to its output.
  - **Computation** is made up of a sequence of functional steps that produce - at its end - some result as its goal. These steps are combined by temporal sequencing.
  - **Unpredictable modern computations and solutions:** Modern computations are embedded in physical environments where the temporal sequencing of events is unpredictable and (potentially) without an explicit end.
    - To cope with this unpredictability computation should be modeled as *interaction*!

- Computation is a community of persistent entities coupled together by their ongoing interactive behaviour.
  - Example applications under this view includes, robotics, video games, global positioning systems, and home security alarm systems.
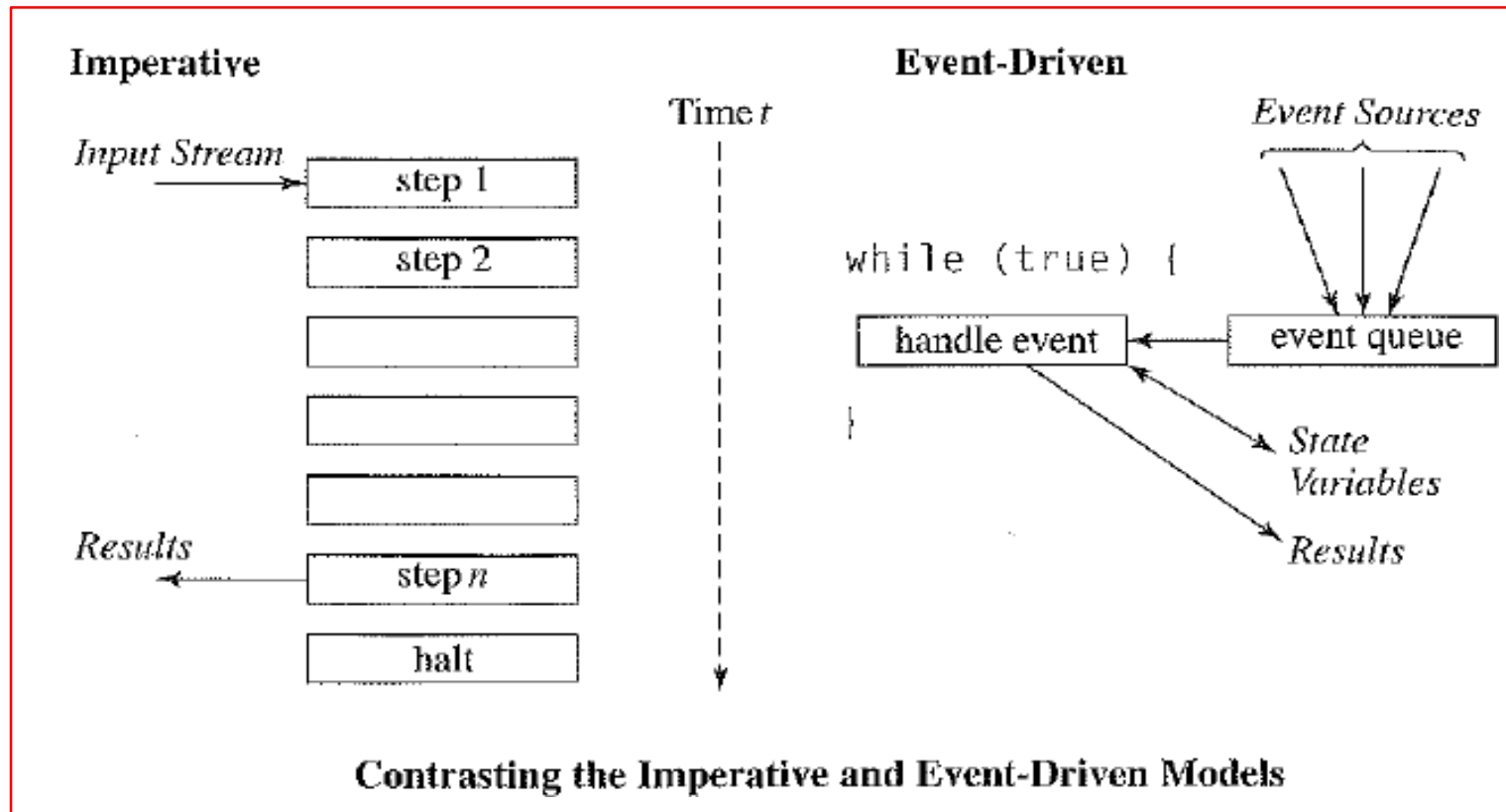
Event-Driven Programming

- The program is designed so that the input order determines the computational design. Entry point ➔ Exit point
  - input is gathered near the beginning of the computation and results are emitted near the end.
- The programmer determines the order of the input data, which in turn determines the computational design.
- The computation consumes input data and produces results in a continuous loop.
- Supported paradigms:
  - **Imperative, object-oriented, functional and logic**
- **Control flow is based on the structure**
  - Program controls the sequence of steps that occur at run time

Event Driven model

- The program prescribe the set of event handlers.

Event Loop

  - The input comes from distinct autonomous *event sources. i.e.,* sensors on a robot or buttons in an interactive application.
- Can't determine Which events happen in which order?
- The program may consult and/or change the value of a variable or produce intermediate *results.*
- Supported paradigms:
  - **Event-driven paradigm**
- **Control flow is based on the internal and external events**
  - Flow of control of program to change according to events that are external to the program itself.

Properties of event-driven programs

- Importantly, two *properties of event-driven programs*:
  - An event-driven program typically has **no preconceived stopping point**, such as reaching the end of file in reading data.
  - The explicit read-evaluate-print loop present in traditional input-controlled programs **does not explicitly** appear in event-driven programs.

Imperative

Input Stream

step 1

step 2

Results

step n

halt

Time t

Event-Driven

Event Sources

while (true) {

handle event ← event queue

}

State Variables

Results

Contrasting the Imperative and Event-Driven Models

Model-View-Controller

- Gamma *et al.,* 1995 discussed many Object-oriented design pattern, and, model-view-controller (MVC), is particularly useful in the design of event-driven GUI applications.

- **Definition**: In MVC the model is the actual object being implemented, the controller is the input mechanism (buttons, menus, combo boxes, etc.) and the view is the output representation of the model.

- MVC pattern it is good practice to decouple the model, the view, and the controller as much as possible.

- The MVC framework supports separation of logic and keeps application tasks separate, such as **UI, business logic, and input**.
    - **Easily manage complexity**
    - **Supports Test Driven Development**

Introduction to MVC (Model-View-Controller)
MVC is a software architectural pattern that separates an application into three interconnected components: Model, View, and Controller. This separation helps manage complexity, improves organization, and makes it easier to maintain and scale applications. Below is a brief overview of each component:

MVC Architecture

- **Model**
  - Domain layer (database instances)
  - ***Manage application data and business logic***
  - Accepts request from controller and update its content accordingly
- **View**
  - <u>UI</u>, display results to user
  - Accepts request from controller and generate output using model
- **Controller**
  - Control logic , Handles **user interaction**.
 - Receive user inputs, instruct model what to do and view what to display
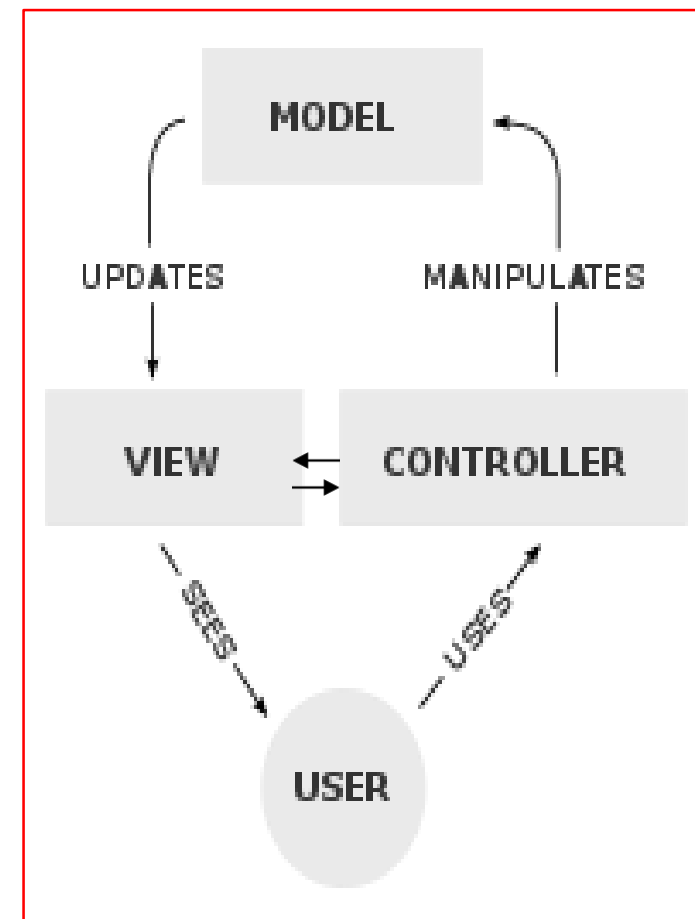
Source: Wikipedia

Thread

- **Multithreading**: allows applications to handle activities <mark>concurrently</mark> leading to <mark>efficient</mark> execution.
- Solution for concurrency
- Multiple independent execution streams
- **Shared state**
- **Synchronization**(locks, conditions)
- synchronous I/O operation
- Only use threads where true CPU concurrency is needed. Scalable performance on multiple CPUs
- Use threads only for performance-critical kernels.
- **Apache** rely on multithreading
- important for high-end servers (e.g. **databases**).
- Problem:
  - Deadlock
  - Threads break abstraction, i.e can't design modules independently
  - Locking increases complexity and reduces performance

Event

- **Event loop**: uses <mark>indefinite loop</mark> to receive requests and process them
  - When multiple requests are made, the first is processed while the rest are blocked (until the first is complete).
  - Handling more and more concurrent client's request is very easy.
- <mark>no concurrency, no preemption, no synchronization, no deadlock.</mark>
- One execution stream i.e one handler for each event(button press, release)
- Support <mark>Callback mechanism</mark>
- **Asynchronous** I/O operation
- Use events, not threads, for GUIs, distributed systems, low-end servers
- Use events as primary development tool (both GUIs and distributed systems).
- **Node.js** based on asynchronous I/O eventing model to create high performance web applications easily. Single threaded event loop model.
- Facebook's chat server is being powered by **Erlang**, which uses same model as node.

Event Handlers
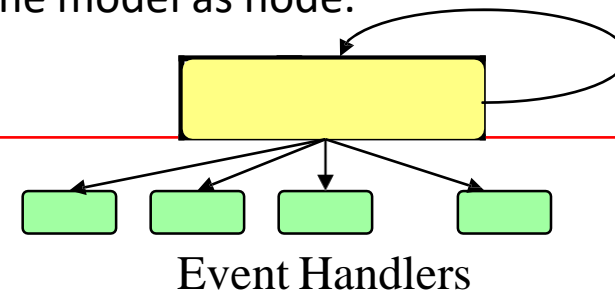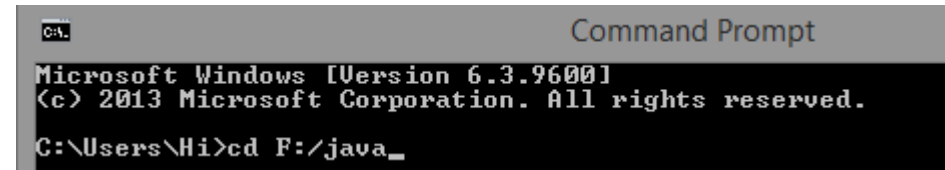
Graphical User Interface

UI User Interface

- ## Command Line Interface (CUI)
  - Type commands using keyboard to interact with computers.
  - Non-graphical UI

- ## Graphical User Interface(GUI)
  - Click readymade commands or icons using mouse to interact with computer.

GUI

A *graphical user interface (GUI)* application is a program that runs inside its own window and communicates with users using graphical elements such as buttons and menus.
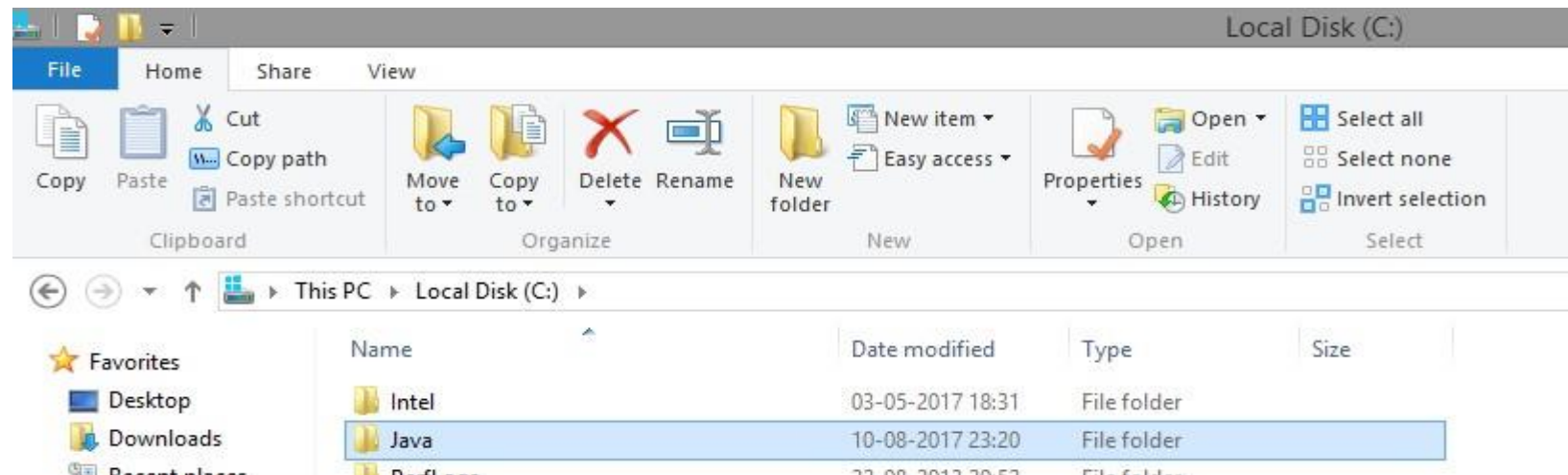
- Graphical User Interface

- G-U-I or Gooey

- Its an interface between human and computer interaction.

- Includes graphical elements such as icons, buttons, windows or visual indicators to interact with electronic devices.

- Introduced by **Steve Jobs** for Apple Macintosh in 1984, Windows 1 in 1985, Windows 95 for Microsoft office, Nokia and Apple iphone 2007,ipad in 2010, Windows 8 and 10 in 2012 and 2015, Holograms and Virtual reality now.

GUI in Java

- JFC <mark>Java Foundation Class</mark>
  - API that provides features for building GUI, Graphics and interactivity to Java applications.
- AWT <mark>Abstract Window Toolkit</mark>
- SWING
- JavaFX

AWT vs Swing

AWT

- Heavy-weight components
- Platform specific
  - Rely on native GUI
  - need Paint() of specific platform to draw
- Introduced in java 1.1
- Simple applications
- Package java.awt

SWING

- Light-weight components
- Platform independent
  - Donot rely on native GUI except for java.awt.Window and java.awt.Panel
- Introduced in java 2
- Complex applications
- All classes begins with 'J'
  - i.e., JButton, JTextField etc.,
- Package javax.swing

Swing classes
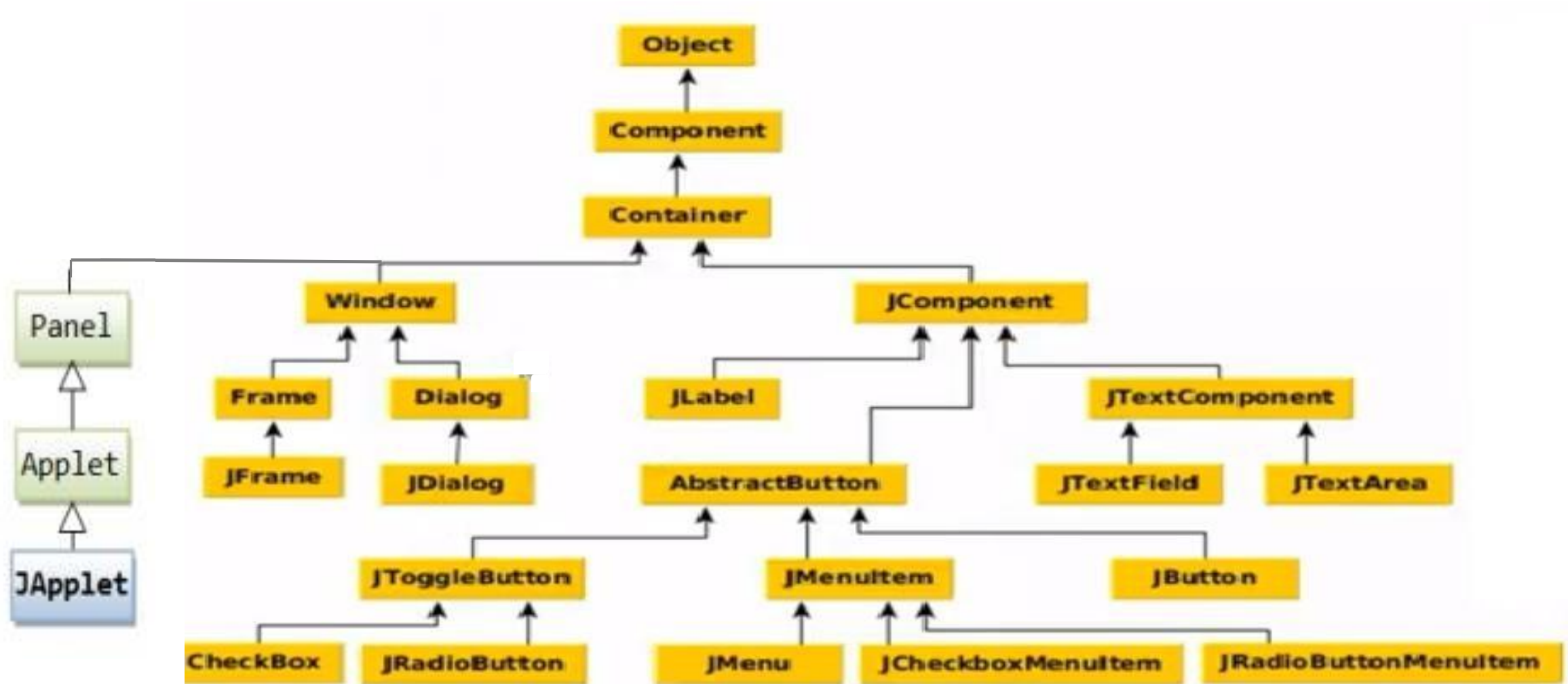
AWT
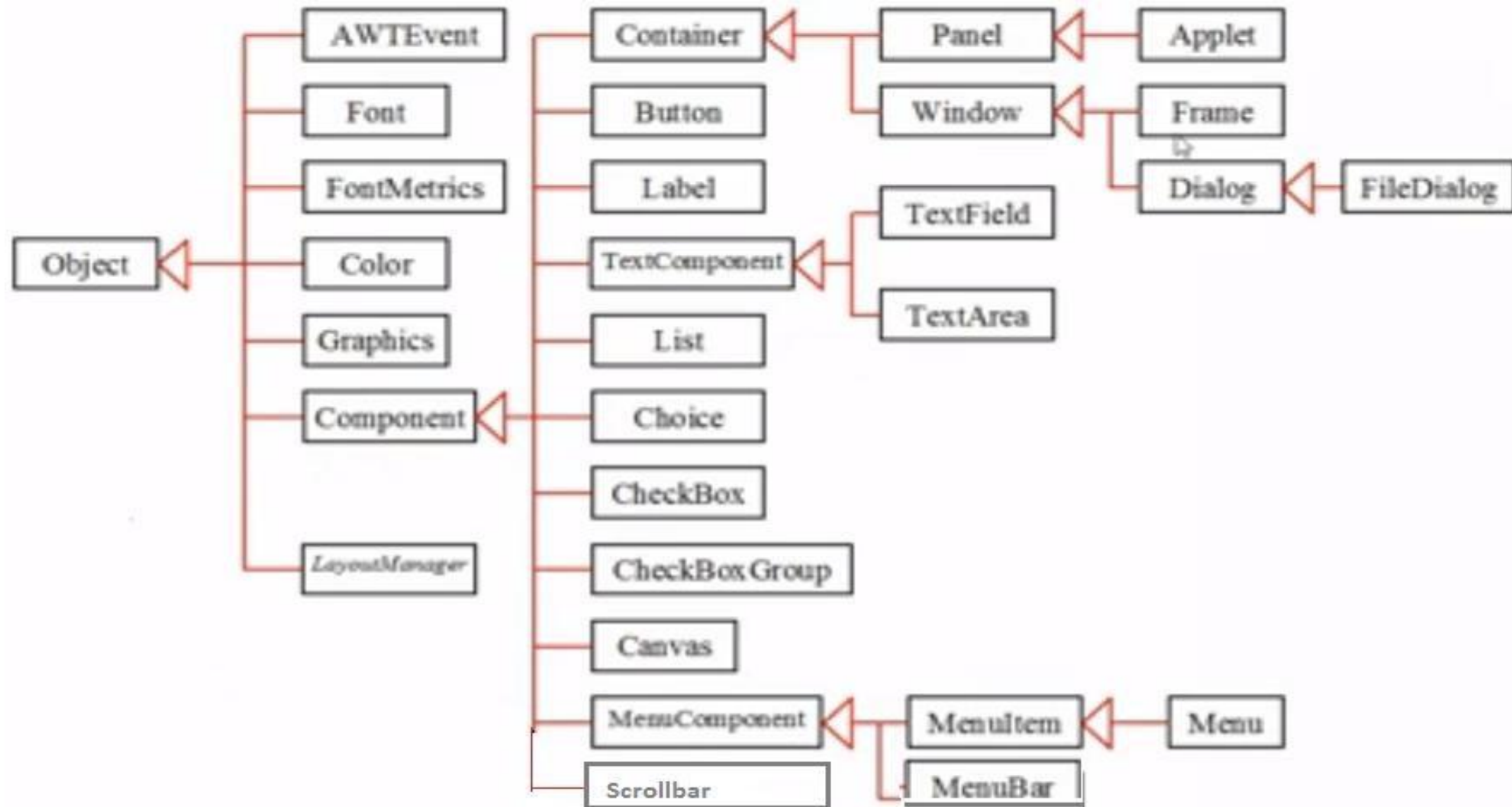
Ref : google.com/image

Events in Java:Event Handling Model

Name

- Event object
- Event source
- Event listener
- Event handling methods

```
s   -> event source
l   -> event listener of type XXX
s.addXXX(l)
```

Element

- ActionEvent
- Button
- ActionListener
- actionPerformed(..)

```
b = new Button("Hello!");
add(b);
b.addActionListener(this);
```

- The types of **events that can occur in Java** are defined by the subclasses of the predefined abstract class **AWTEvent**.
- Every event source in an interaction can generate an event that is a member of one of these classes.
- For instance, if a **button is an event source**, it generates **events** that are **members of the ActionEvent** class.
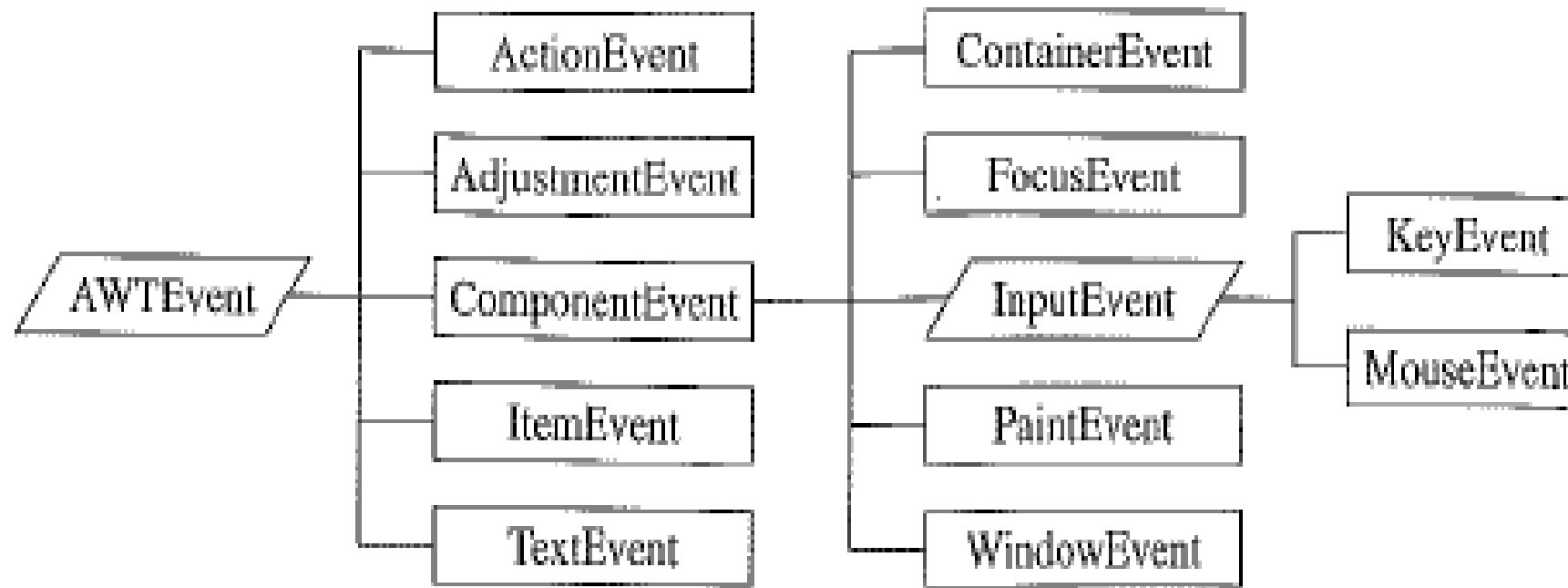
Java Class AWTEvent and Its Subclasses

Event Class	: Description
ActionEvent	button is pressed; list item is double-clicked; menu item is selected;
AdjustmentEvent	scroll bar is manipulated. [String getActionCommand( ) , long getWhen( )]
ComponentEvent	component is hidden, moved, resized, visible.
ContainerEvent	component is added to a container or removed from a container.
FocusEvent	component gains keyboard focus or loses keyboard focus.
InputEvent	Abstract superclass for all component input event classes.
ItemEvent	check box or list item is clicked , selected or deselected. [int getStateChange( ), Object getItem( )]
KeyEvent	input is received from the keyboard.
MouseEvent	mouse is dragged, moved, clicked, pressed, or released; mouse enters or exits a component.
MouseWheelEvent	mouse wheel is moved
MouseWheelEvent	text area value or text field value is changed.
WindowEvent	window is activated, closed, deactivated, opened, or quit.

The objects themselves that can be **event sources** in the Swing framework are subclasses of the abstract class **Jcomponent**

for example, that any **button** that can be selected by the user in an interaction is declared as an **instance** of the **JButton** class.

AbstractButton

JButton

JMenuItem

JToggleButton

JComboBox

JComponent

JLabel

JPanel

JPopupMenu

JTable

JTextArea

JTextComponent

JTextField
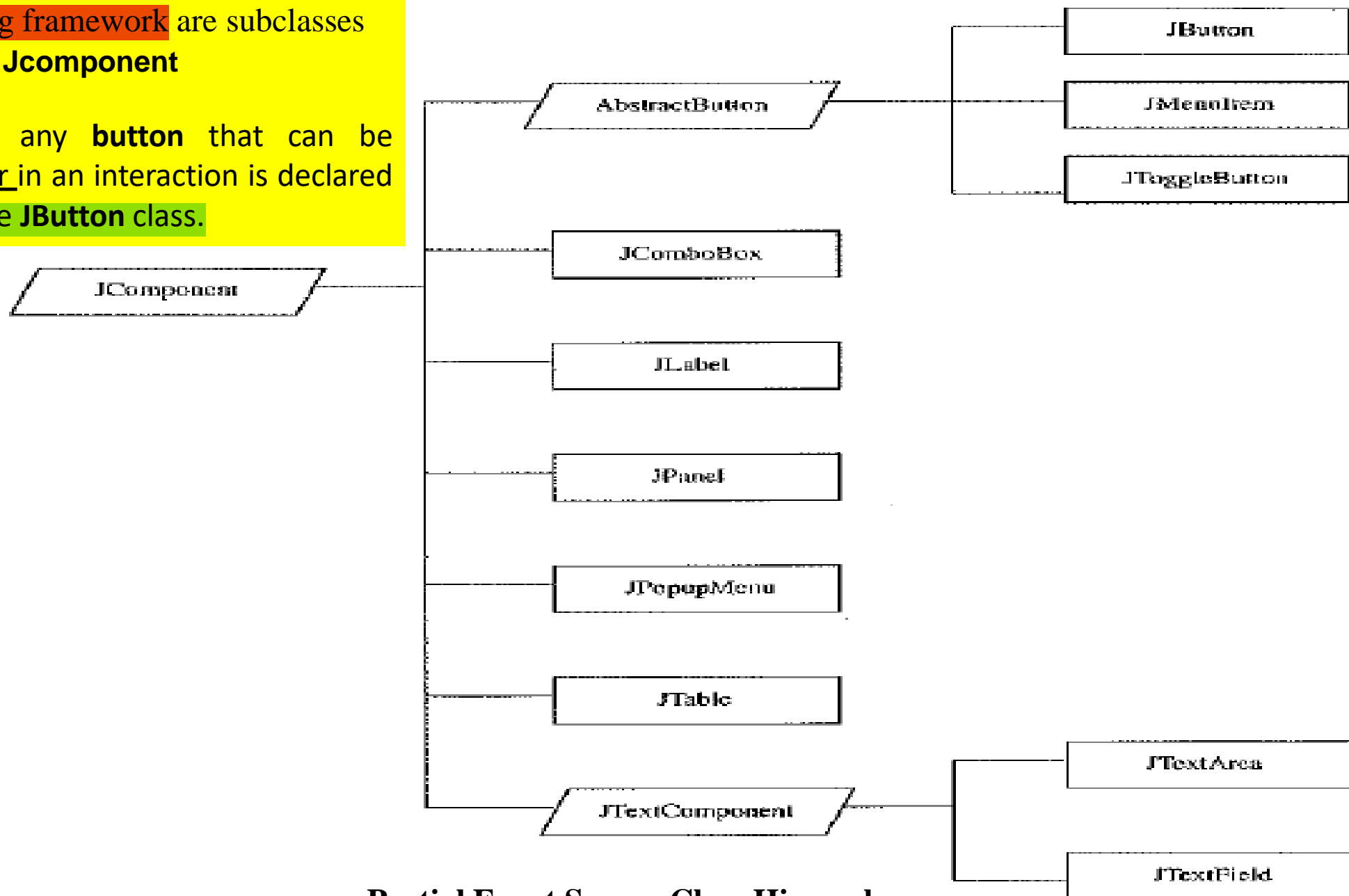
Partial Event Source Class Hierarchy

For a **program to handle an event**, it must be equipped with appropriate "listeners" that will recognize when a particular event, such as **a mouse click**, has occurred on an object that is an **event source**.
For example, to equip a **button** so that the program can "detect" an occurence of that button's selection, the button needs to invoke its **addActionListener** method.
If this is not done, button events will not be detected by the program.

```
                              ┌─ ActionListener
                              ├─ AdjustmentListener
                              ├─ ComponentListener
                              ├─ ContainerListener
                              ├─ FocusListener
EventListener ────────────────┼─ ItemListener
                              ├─ KeyListener
                              ├─ MouseListener
                              ├─ MouseMotionListener
                              ├─ TextListener
                              └─ WindowListener
```
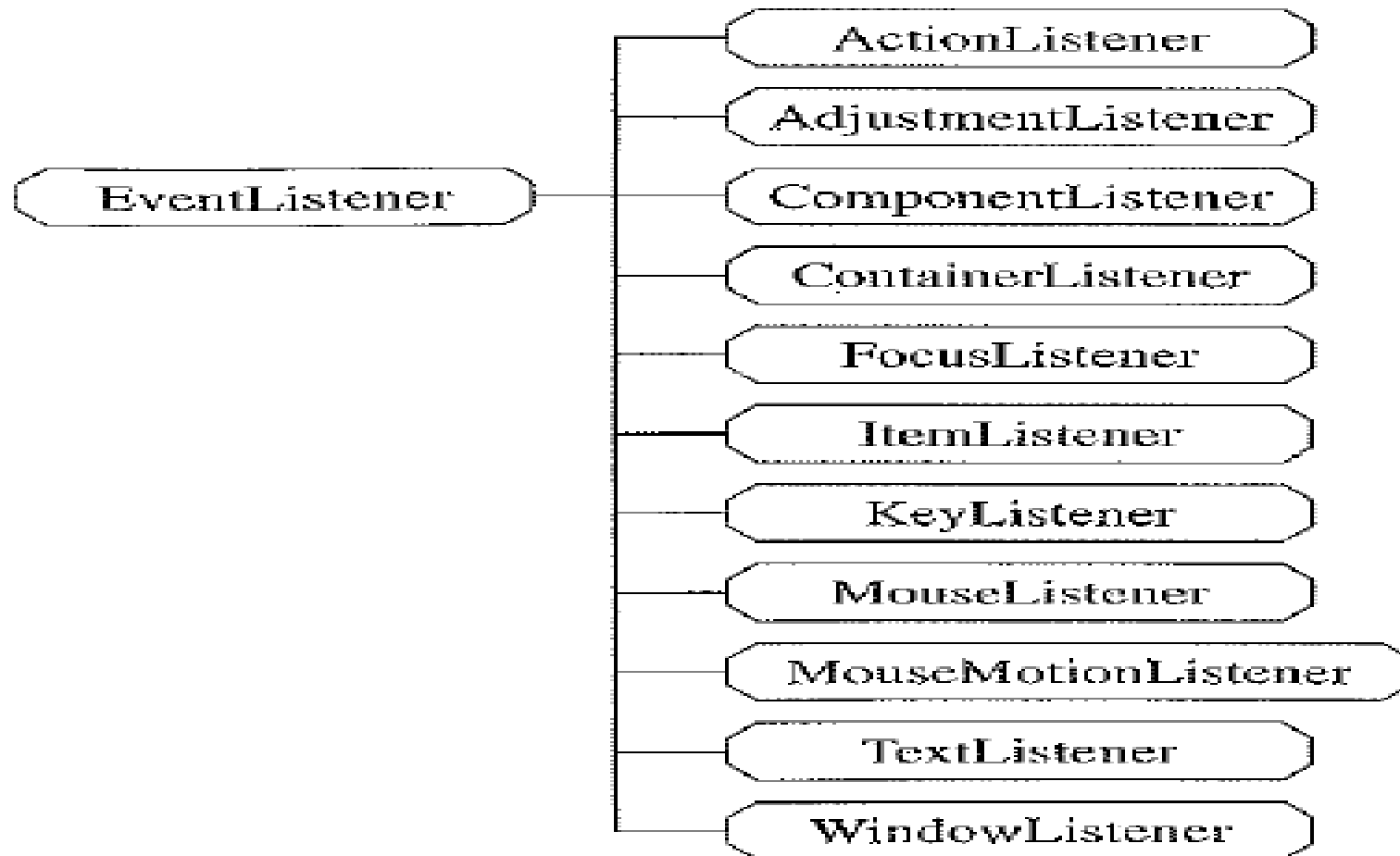
Java EventListener Classes

Components and Their Event Handlers

actionPerformed!!!

Widget	Listener	Interface Methods
JButton	ActionListener	actionPerformed(ActionEvent e)
JComboBox	ActionListener	actionPerformed(ActionEvent e)
JLabel		
	MouseListener	mouseClicked(MouseEvent e)
		mouseEntered(MouseEvent e)
		mouseExited(MouseEvent e)
		mousePressed(MouseEvent e)
		mouseReleased(MouseEvent e)
	MouseMotion-Listener	mouseDragged(MouseEvent e)
		mouseMoved(MouseEvent e)
JTextArea	ActionListener	actionPerformed(ActionEvent e)
JTextField	ActionListener	actionPerformed(ActionEvent e)

the interfcae method used!!

Four different kinds of user-initiated events can be handled by:

- **Mouse motion events** (handled by the **MouseMotionlistener**).
- **Mouse events** (handled by the **Mouselistener**).
- **Button and text field selections** (handled by the **ActionListener**).
- **Selections from a combo box** (handled by the **ActionListener**).

- Importantly, the program cannot know, or predict, the order in which different events will occur, or the number of times each one will be repeated;
- Program has to be prepared for all possibilities. That is the essence of event-driven programming.
- Another important difference is that the programmer writes each method to accomplish its task and exit.
- The loop that awaits events and dispatches them is part of the framework that Java provides.

Containers and Components
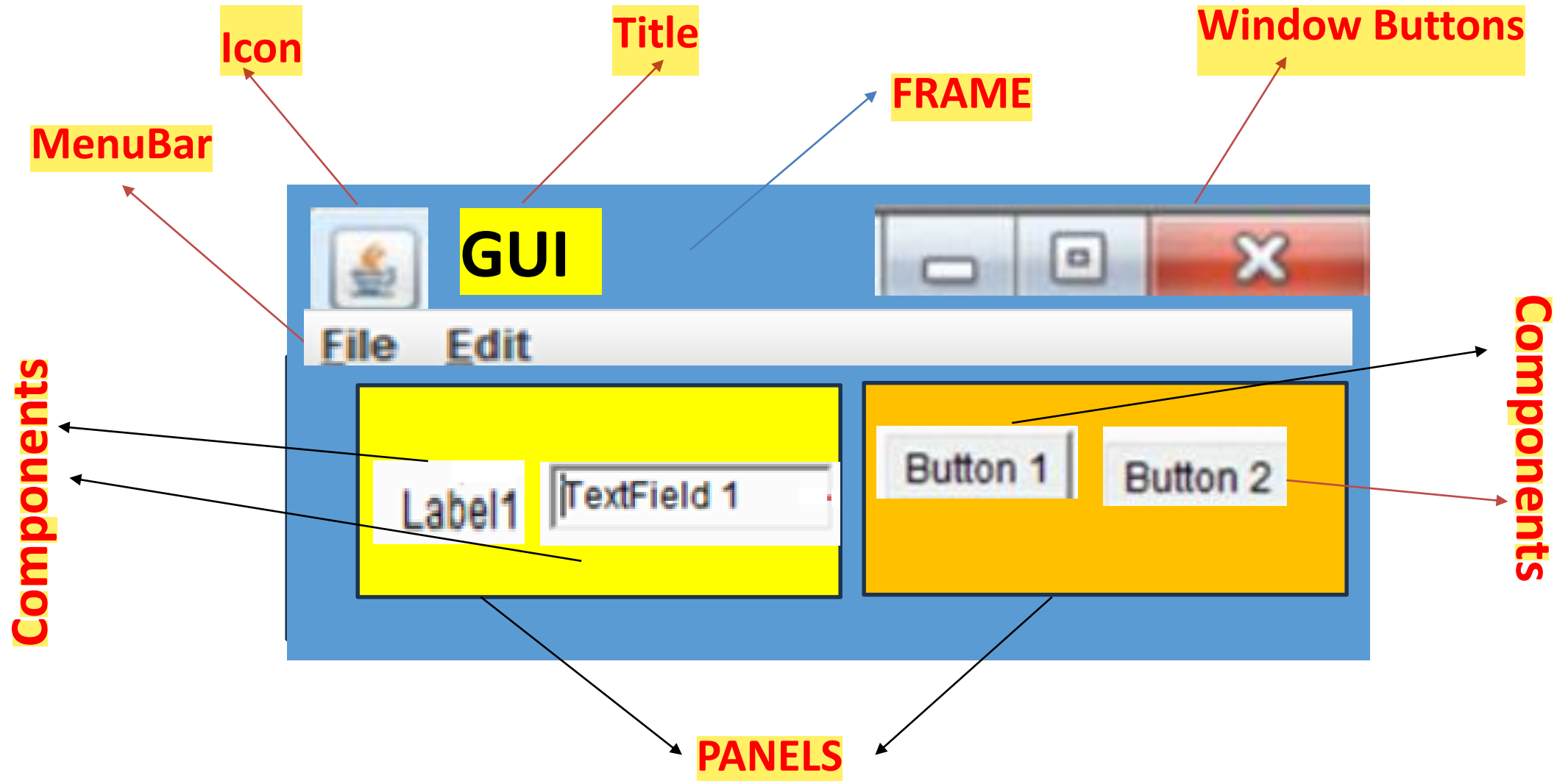
Containers

Frame: Main window to hold components.

Panel: Sub-container for organizing components.

Dialog: Pop-up window for user interactions.

ScrollPane: Adds scrollbars to larger components.

Frame: Outer-level component

Main window and a top-level container to hold components.

A Frame has

- A title bar
    - icon, a title, minimize/maximize/close buttons, and the optional menu bar

- A display area.
    - Hold sub-containers, components

- A method to add the components
    - add(Component C)

Panel: Sub-container

A light-weight container used to group and layout atomic components

- Organizes the components
- Has a method to add the components within the panel
  - add(Component C)

Dialog: Sub-container

Dialog is a pop-up window used for interacting with users, often to gather input, display messages, or warning actions. Organizes the components

- Types of Dialogs
  - Input Dialog: Prompts the user for input
  - Information Dialog: Displays information or alerts.
  - Warning Dialog: Displays warning message to the user.

```
JOptionPane.showMessageDialog(null, "This is an informational message", "Information", JOptionPane.INFORMATION_MESSAGE);
JOptionPane.showMessageDialog(null, "An error occurred!", "Error", JOptionPane.ERROR_MESSAGE);
                                        JOptionPane.showMessageDialog(null, "This is a warning", "Warning", JOptionPane.WARNING_MESSAGE);
```

SYNTAX:
JOptionPane.showMessageDialog(Component parentComponent, Object message, String title, int messageType);

```
String name = JOptionPane.showInputDialog("Enter your name:");     JOptionPane.
System.out.println("Hello, " + name + "!");
```

Parameters:
parentComponent: The parent component for the dialog. Can be null if you don't want it to have a parent.
message: The message to display (could be a String, Icon, or any other object).
title: The title of the message box (typically shown in the title bar).
messageType: The type of message (like INFORMATION_MESSAGE, ERROR_MESSAGE, etc.).

Dialog:

Input message

Information message

Input Dialog   ✕

?   **Please enter a value:**

    [   ]

     OK   Cancel

User Input   ✕

ⓘ   You entered: Jayachitra

     OK

Warning message

User Input   ✕

⚠   No input was provided.

     OK
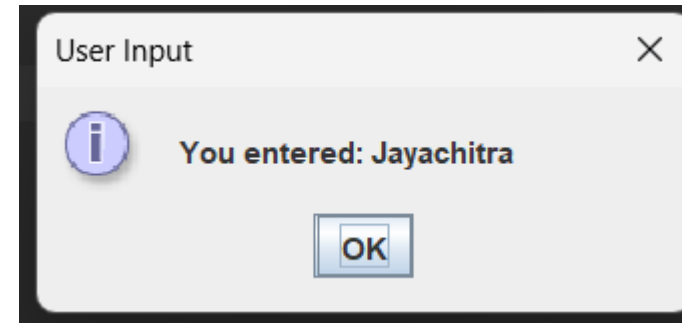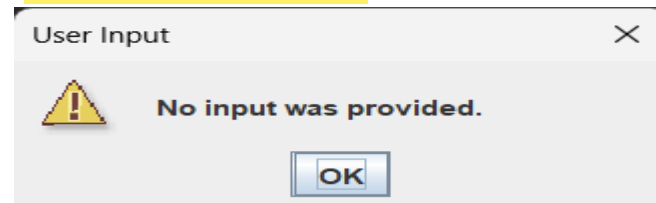
Syntax:
    String userInput = JOptionPane.showInputDialog(
    Component parentComponent, // Parent component for the dialog (can be null)
    Object message,   // The message or prompt to display
    String title   // Title for the dialog window
    int messageType   // Type of message (e.g., JOptionPane.QUESTION_MESSAGE)
    );

```java
import java.awt.FlowLayout;
import java.awt.Color;


public class addgui{   //MAIN CLASS!!
    public static void main(String args[])
        {
         addition n =new addition();
        }
}
```

C:\Program Files\Java\jdk1.8.0_131\bin>javac f:/java/addgui.java

C:\Program Files\Java\jdk1.8.0_131\bin>java -cp f:/java addgui

```java
import java.awt.*;
import java.awt.event.*;
import java.swing.*;
public class <classname> extends JPanel implements <listeners> {
<instance variable declarations>
public <classname> ( ) {
<code to initialize the GUI> }
<event handlers>
}
```
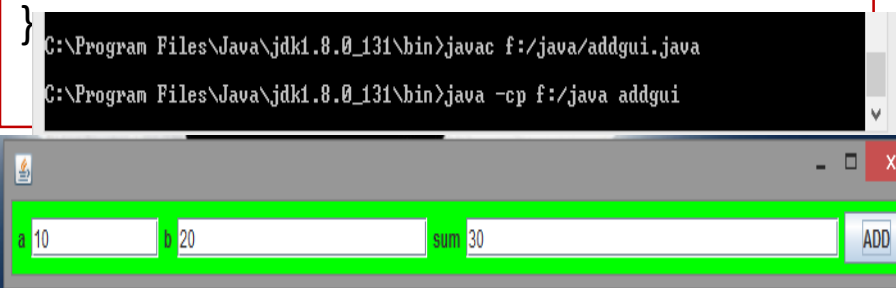
IMPORTANT!!!!!!!!!

Overall Structure of a GUI Java Application

```java
//GUI example program to add two numbers using swing class
 class addition extends JFrame implements ActionListener{
        JTextField a,b,sum;
        JButton  add;
        JLabel al,bl,suml;
  addition()
  {
    al=new JLabel("a");         a=new JTextField(10);
    bl=new JLabel("b");         b=new JTextField(20);
    suml=new JLabel("sum"); sum=new JTextField(30);
    add=new JButton("ADD");
    add(al);add(a);add(bl); add(b);add(suml); add(sum); add(add);
    setLayout(new FlowLayout());
    setSize(400,400);
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    getContentPane().setBackground(Color.GREEN);
    add.addActionListener(this);
    pack();        }
    public void actionPerformed (ActionEvent e){
        int num1=Integer.parseInt(a.getText());
        int num2=Integer.parseInt(b.getText());
        int totalSum=num1 + num2;
        sum.setText(Integer.toString(totalSum));
        }}
```

Dialog:

```java
import javax.swing.*;
public class BasicGUIComponents extends JFrame {
    BasicGUIComponents() {
        String userInput = JOptionPane.showInputDialog(this, "Please enter a value:", "Input Dialog", JOptionPane.QUESTION_MESSAGE);
        if (userInput != null) {
            JOptionPane.showMessageDialog(this, "You entered: " + userInput, "User Input", JOptionPane.INFORMATION_MESSAGE);
        } else {
            JOptionPane.showMessageDialog(this, "No input was provided.", "User Input", JOptionPane.WARNING_MESSAGE);
        }
    }

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

JOptionPane!!!

Input message

Input Dialog

? Please enter a value:

[   ]

OK   Cancel

Information message

User Input

(i) You entered: Jayachitra

OK

Warning message

User Input

⚠ No input was provided.

OK

```java
import javax.swing.*;
import java.awt.*;

public class BasicGUIComponents extends JFrame {
    BasicGUIComponents() {
        setSize(400,200);
        setLayout(new FlowLayout());
        JTextArea textArea = new JTextArea(5, 30);
        textArea.setText("This is an example of a JTextArea inside a JScrollPane.\n"
            + "You can add more lines of text to see the scrollbars in action.\n"
            + "Scroll down to view more text...\n");
        add(textArea);
    }

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

JTextArea:

JTextArea is a component in Java's Swing library that provides a multi-line area for text input and display. It's commonly used in applications that require text editing or displaying large amounts of text, such as text editors, logs, or chat applications.

This is an example of a JTextArea inside a JScrollPane.
You can add more lines of text to see the scrollbars in action.
Scroll down to view more text...

```java
import javax.swing.*;
import java.awt.*;

public class BasicGUIComponents extends JFrame {
    BasicGUIComponents() {
        setSize(400,200);
        setLayout(new FlowLayout());
        JTextArea textArea = new JTextArea(5, 30);
        textArea.setText("This is an example of a JTextArea inside a JScrollPane.\n"
            + "You can add more lines of text to see the scrollbars in action.\n"
            + "Scroll down to view more text...\n");
    JScrollPane scrollPane = new JScrollPane(textArea);
     add(scrollPane);
    }

    public static void main(String[] args) {
        new BasicGUIComponents().setVisible(true);
    }
}
```
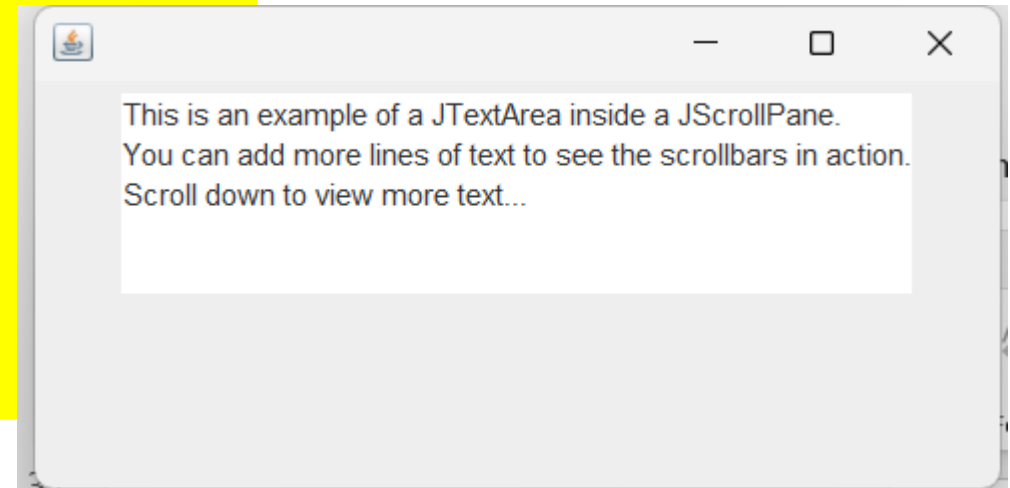
Wraps the JTextArea: The JScrollPane acts as a container that provides a scrollable view of the JTextArea. This means that if the content of the JTextArea exceeds its visible area, the JScrollPane will automatically provide vertical and/or horizontal scrollbars to allow users to navigate through the text.

Dynamic Sizing: The JScrollPane manages the size of the JTextArea based on the content. As more text is added to the JTextArea, the scrollbars will appear when necessary, making it easier to read or edit large amounts of text without changing the layout of the entire GUI.
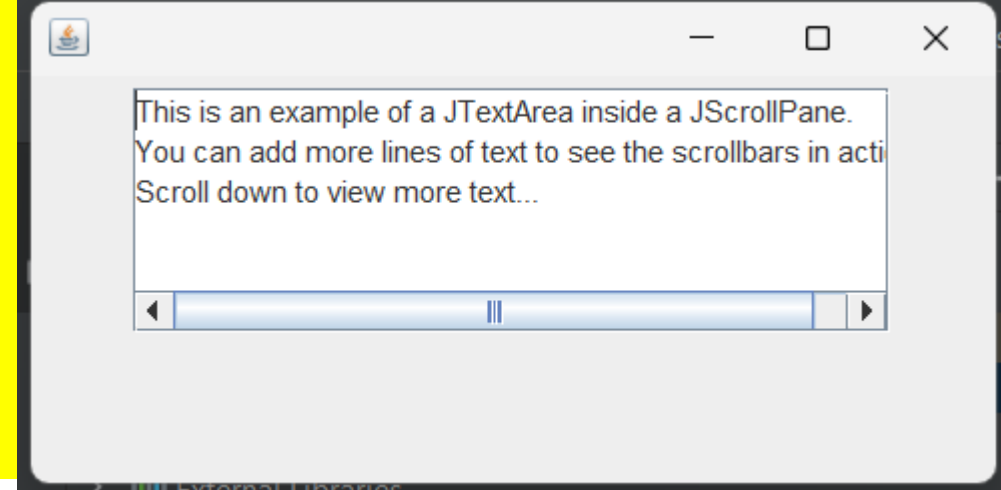
The line JScrollPane scrollPane = new JScrollPane(textArea);
 creates a JScrollPane that contains the JTextArea (referred to as textArea).

Components

- **Button:** Triggers actions when clicked.

- **Label:** Displays non-interactive text/images.

- **TextField:** For single-line text input.

- **TextArea:** For multi-line text input.

- **Checkbox:** Select/deselect options.

- **Radio Button:** Single selection from a group.

- **Combo Box:** Drop-down list for selections.

- **List Box**: Displays a list of items from which users can select one or more options.

- **CheckBoxGroup:** allows users to select multiple options simultaneously.

CheckBoxGroup: In Java AWT (Abstract Window Toolkit), a CheckboxGroup is used to group a set of checkboxes. However, it allows users to select only one checkbox from the group at a time, similar to radio buttons. This means that when one checkbox in the group is selected, the previously selected checkbox is automatically deselected.

Enter your name here — TextField

Click Me! — Button

This is Label — Label

Red (Red, Green, Blue) — Choice

☑ one ☐ two ☐ three — CheckBox

○ Alpha ○ Beta ○ Charlie — CheckBoxGroup

Mercury / Venus / Earth / Mars / Jupiter / Saturn / Uranus / Neptune — List

LABEL:

JLabel(): Creates an empty label.

JLabel(String text): Creates a label with the specified text.

JLabel(String text, Icon icon, int horizontalAlignment): Creates a label with specified text, icon, and horizontal alignment.

JLabel(Icon icon): Creates a label with the specified icon.

void setText(String text): Sets the text of the label.

String getText(): Returns the text of the label.

void setIcon(Icon icon): Sets the icon displayed on the label.

Icon getIcon(): Returns the icon displayed on the label.

Alignments: SwingConstants.LEFT, SwingConstants.CENTER, SwingConstants.RIGHT

Example: setHorizontalAlignment(SwingConstants.CENTER);   Note!!!

        setVerticalAlignment(SwingConstants.CENTER);

void setFont(Font font):

void setForeground(Color color):

void setBackground(Color bg):

LABEL:

```java
import javax.swing.*;
import java.awt.*;

public class GUILabel {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JLabel Example"); //not extending frame hence creating a class!!
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new FlowLayout());
        JLabel label1 = new JLabel("Hello, World!");
        label1.setFont(new Font("Arial", Font.BOLD, 16));
        label1.setForeground(Color.BLUE);
        label1.setOpaque(true);
        label1.setBackground(Color.LIGHT_GRAY);
        label1.setHorizontalAlignment(SwingConstants.CENTER);
        ImageIcon icon = new ImageIcon("c:/pictures/Image1.png");
        JLabel label2 = new JLabel("Label with Icon", icon, SwingConstants.LEFT);
        frame.add(label1);
        frame.add(label2);
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

TextField

JTextField(): Creates an empty text field.
JTextField(int columns): Creates a text field with the specified number of columns.
JTextField(String text): Creates a text field initialized with the specified text.
JTextField(String text, int columns): Creates a text field with the specified text and number of columns.

void setText(String text): Sets the text of the text field.
String getText(): Returns the current text in the text field.

void setColumns(int columns): Sets the number of columns in the text field
int getColumns(): Returns the number of columns.

void setEditable(boolean editable): Sets whether the text field is editable or not.
boolean isEditable(): Returns true if the text field is editable.

void selectAll(): Selects all text in the text field.
void select(int start, int length): Selects a portion of the text.

void setBackground(Color bg): Sets the background color.
void setForeground(Color fg): Sets the text color.

TEXTFIELD:

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class GUITextFileld extends JFrame {
    private JLabel lblCount;
    private JTextField tfCount;
    private JButton btnCount;
    private int count = 0;
    public GUITextFileld() {
        setLayout(new FlowLayout());
        lblCount = new JLabel("Counter");
        add(lblCount);
        tfCount = new JTextField(count + "", 10);
        tfCount.setEditable(false);        // Set to read-only
        add(tfCount);
        btnCount = new JButton("Count");
        add(btnCount);
        btnCount.addActionListener(new BtnCountListener()); // Add ActionListener
        setTitle("Swing Counter");
        setSize(300, 100);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);        // Show the JFrame    }
```

```java
    public static void main(String[] args) {   // MAIN FUCNTION!!!
        GUITextFileld app = new GUITextFileld();
    }
    // inner class to handle the "Count" button-click
    private class BtnCountListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent evt) {
            ++count;
            tfCount.setText(count + ""); // Convert int to String
        }
    }
}
```


Swing Counter — Counter: 0 — Count

Swing Counter — Counter: 4 — Count
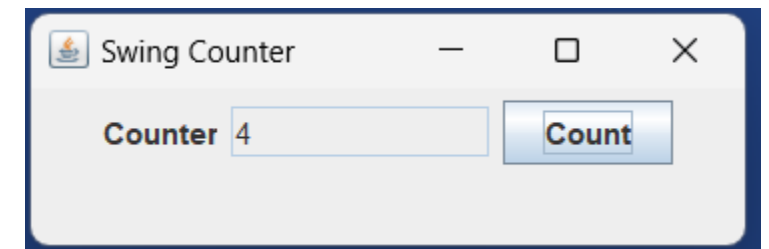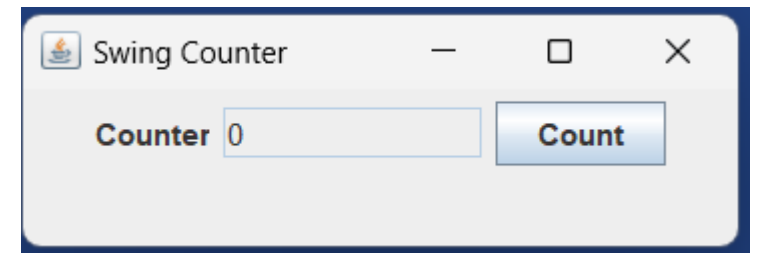
Button

JJButton(): Creates an empty text field.
JButton(String text): Creates a button with the specified text.
JButton(Icon icon): Creates a button with the specified icon.
Button(String text, Icon icon): Creates a button with the specified text and icon.

void setText(String text): Sets the text displayed on the button.
String getText(): Returns the text displayed on the button.
void setIcon(Icon icon): Sets the icon displayed on the button.
Icon getIcon(): Returns the icon displayed on the button..

void setBackground(Color color): Sets the background color of the button.
void setForeground(Color color): Sets the text color of the button.

void setEnabled(boolean enabled): Enables or disables the button.
boolean isEnabled(): Returns true if the button is enabled.

void addActionListener(ActionListener l): Adds an action listener to the button to handle click events.
void removeActionListener(ActionListener l): Removes an action listener from the button.

void setToolTipText(String text): Sets the tool tip text to be displayed when the mouse hovers over the button.

```java
import javax.swing.*;        //different buttons is executed on same actionListener
import java.awt.*;
import java.awt.event.*;
public class GUIButton extends JFrame {
    private JButton buttonWithTextAndIcon; // JButton with text and icon
    private JButton buttonEmpty;         // Empty JButton
    private JButton buttonDisabled;       // Disabled JButton
    public GUIButton() {
        setTitle("JButton Example");       setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());
        ImageIcon icon = new ImageIcon("c:/pictures/Image2.png");
            buttonWithTextAndIcon = new JButton("Click Me!", icon); // Button with text and icon
        buttonWithTextAndIcon.setBackground(Color.LIGHT_GRAY);
        buttonWithTextAndIcon.setForeground(Color.BLUE);
        buttonWithTextAndIcon.setToolTipText("This button does something!");
        buttonWithTextAndIcon.addActionListener(new ButtonClickListener());
        buttonEmpty = new JButton(); // Create an empty JButton
        buttonEmpty.setText("Empty Button");
        buttonEmpty.setBackground(Color.GREEN);
        buttonEmpty.setToolTipText("This is an empty button.");
        buttonEmpty.addActionListener(new ButtonClickListener());
        buttonDisabled = new JButton("Disabled Button"); // Create a disabled JButton
        buttonDisabled.setEnabled(false); // Disable the button
        buttonDisabled.setToolTipText("This button is disabled.");
        add(buttonWithTextAndIcon);
        add(buttonEmpty);
        add(buttonDisabled);        setVisible(true);    }
```

```java
    private class ButtonClickListener implements ActionListener {
        @Override  //ActionListener
        public void actionPerformed(ActionEvent e) {
            JButton sourceButton = (JButton) e.getSource();
            JOptionPane.showMessageDialog(null,
sourceButton.getText() + " clicked!");
        }   }
    public static void main(String[] args) {        MAIN METHOD!!
        new GUIButton();
}}
```

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class GUIButton extends JFrame {
    private JButton buttonWithTextAndIcon; // JButton with text and icon
    private JButton buttonEmpty;          // Empty JButton
    private JButton buttonDisabled;        // Disabled JButton
    public GUIButton() {
        setTitle("JButton Example");      setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());
        ImageIcon icon = new ImageIcon("c:/pictures/Image2.png");
            buttonWithTextAndIcon = new JButton("Click Me!", icon); // Button with text and icon
        buttonWithTextAndIcon.setBackground(Color.LIGHT_GRAY);
        buttonWithTextAndIcon.setForeground(Color.BLUE);
        buttonWithTextAndIcon.setToolTipText("This button does something!");
        buttonWithTextAndIcon.addActionListener(new ButtonClickListener());
        buttonEmpty = new JButton(); // Create an empty JButton
        buttonEmpty.setText("Empty Button");
        buttonEmpty.setBackground(Color.GREEN);
        buttonEmpty.setToolTipText("This is an empty button.");
        buttonEmpty.addActionListener(new ButtonClickListener());
        buttonDisabled = new JButton("Disabled Button"); // Create a disabled JButton
        buttonDisabled.setEnabled(false); // Disable the button
        buttonDisabled.setToolTipText("This button is disabled.");
        add(buttonWithTextAndIcon);
        add(buttonEmpty);
        add(buttonDisabled);        setVisible(true);    }
```
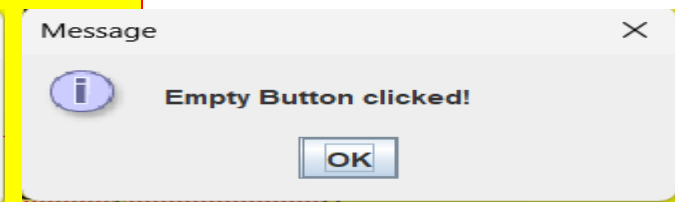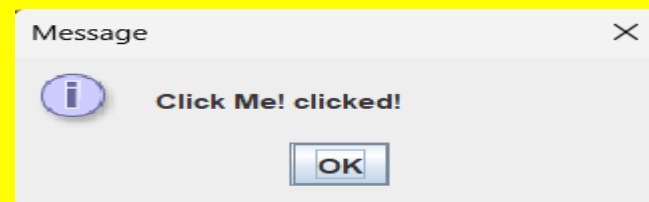
```java
private class ButtonClickListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        JButton sourceButton = (JButton) e.getSource();
        JOptionPane.showMessageDialog(null,
sourceButton.getText() + " clicked!");
    }   }
    public static void main(String[] args) {
        new GUIButton();
}}
```

```java
import javax.swing.*;  //Program that simply sums over an accumulated value
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class GUITextListener  extends JFrame {
    private JTextField tfInput;
    private JTextField tfOutput;
    private int sum = 0;    // Accumulated sum, initialized to 0
    public GUITextListener() {
        setLayout(new GridLayout(2, 2));
        add(new JLabel("Enter an Integer:"));
        tfInput = new JTextField(10);
        add(tfInput);
        tfInput.addActionListener(new TFInputListener());
        add(new JLabel("The Accumulated Sum is:"));
        tfOutput = new JTextField(10);
        tfOutput.setEditable(false);
        add(tfOutput);
        setTitle("Swing Accumulator");
        setSize(350, 120);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String[] args) {
        new GUITextListener();    }
```

```java
// Inner class to handle the input TextField
    private class TFInputListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent evt) {
            try {
                int numberIn = Integer.parseInt(tfInput.getText());
                sum += numberIn;
                tfInput.setText("");
                tfOutput.setText(String.valueOf(sum));
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(null, "Please enter a valid integer.", "Input Error", JOptionPane.ERROR_MESSAGE);
                tfInput.setText("");
            }
        }
    }
}
```

When the user types an integer in the input text field and presses Enter, an action listener (TFInputListener) is triggered. This listener: Tries to parse the entered value as an integer. Adds the parsed integer to the accumulated sum.

Clears the input field and updates the output text field (tfOutput) to show the new accumulated sum.

Swing Accumulator

Enter an Integer: 7

The Accumulated Sum is: 6

Swing Accumulator

Enter an Integer:

The Accumulated Sum is: 13

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class GUITextListener  extends JFrame {
    private JTextField tfInput;
    private JTextField tfOutput;
    private int sum = 0;     // Accumulated sum, initialized to 0
    public GUITextListener() {
        setLayout(new GridLayout(2, 2));
        add(new JLabel("Enter an Integer:"));
        tfInput = new JTextField(10);
        add(tfInput);
        tfInput.addActionListener(new TFInputListener());
        add(new JLabel("The Accumulated Sum is:"));
        tfOutput = new JTextField(10);
        tfOutput.setEditable(false);
        add(tfOutput);
        setTitle("Swing Accumulator");
        setSize(350, 120);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String[] args) {
        new GUITextListener();     }
```
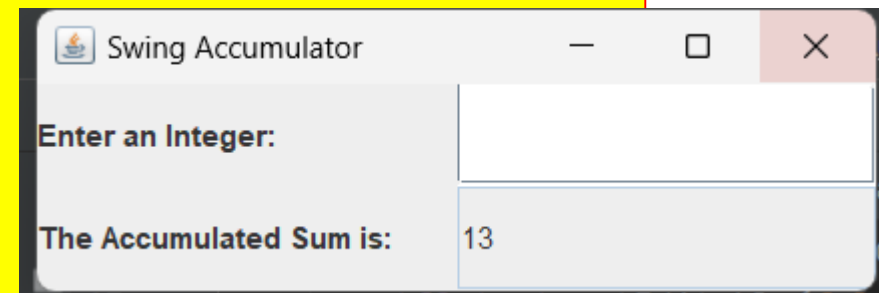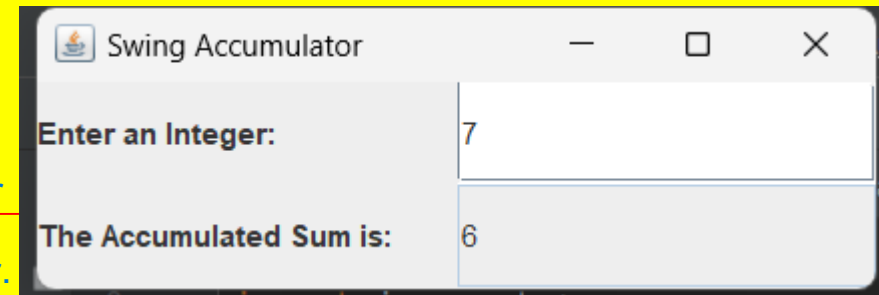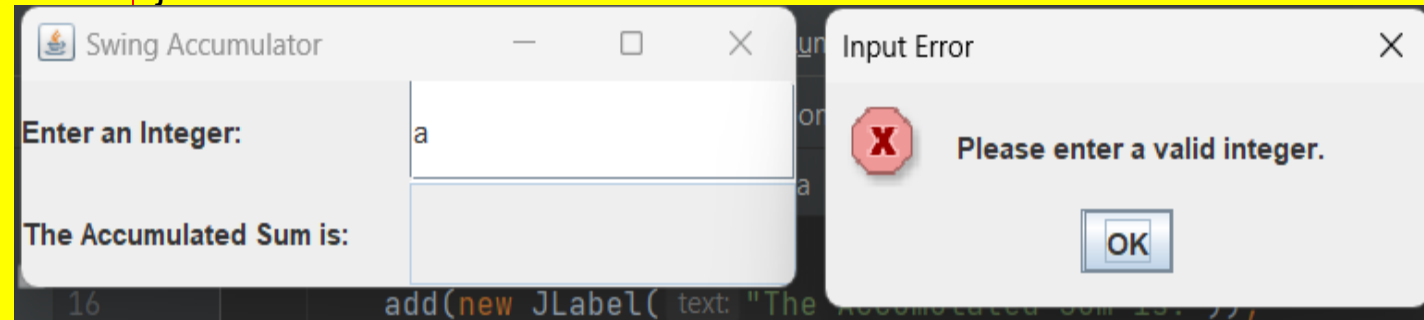
```java
// Inner class to handle the input TextField
    private class TFInputListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent evt) {
            try {
                int numberIn = Integer.parseInt(tfInput.getText());
                sum += numberIn;
                tfInput.setText("");
                tfOutput.setText(String.valueOf(sum));
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(null, "Please enter a
valid integer.", "Input Error", JOptionPane.ERROR_MESSAGE);
                tfInput.setText("");
            }
        }
    }
}
```

Swing Accumulator

Enter an Integer: a

The Accumulated Sum is:

Input Error

Please enter a valid integer.

OK

```java
import java.awt.*;      //COUNTER PROGRAM!
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;

public class UPDOWN extends Frame {
    private TextField tCount;
    private Button bUp, bDown, bReset;
    private int count = 0;
    public UPDOWN() {
        setLayout(new FlowLayout());
        add(new Label("Counter"));
        tCount = new TextField("0", 10);
        tCount.setEditable(false);
        add(tCount);
        bUp = new Button("Count Up");
        add(bUp);
        bUp.addActionListener(e-> {
                ++count;
            tCount.setText(count + "");
        }
    );
                                   1
```
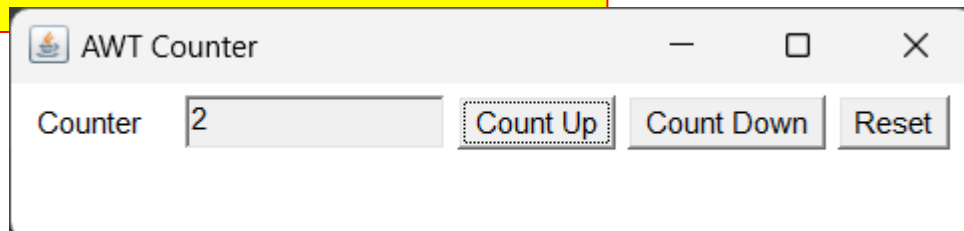
```java
bDown = new Button("Count Down");
    add(bDown);
    bDown.addActionListener(e-> {
        count--;
        tCount.setText(count + "");
    }          );
    bReset = new Button("Reset");
    add(bReset);
    bReset.addActionListener(e->{
        count = 0;
        tCount.setText("0");
    }
);
                        2
```

```java
        setTitle("AWT Counter");
        setSize(400, 100);
        setVisible(true);
                    3
```

```java
    public static void main(String[] args) {
        new UPDOWN();   MAIN METHOD!
    }           5
```

```java
// anonymous inner class
addWindowListener(new WindowListener() {
    public void windowClosing(WindowEvent evt) {
        System.exit(0);
        }
        //  provide an methods for compilation
    public void windowOpened(WindowEvent e) { }
    public void windowClosed(WindowEvent e) { }
    public void windowIconified(WindowEvent e) { }
    public void windowDeiconified(WindowEvent e) { }
    public void windowActivated(WindowEvent e) { }
    public void windowDeactivated(WindowEvent e) { }
    });        }                4
```

AWT Counter

Counter [2] [Count Up] [Count Down] [Reset]

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

public class UPDOWN extends JFrame {
    private JTextField tCount;
    private JButton bUp, bDown, bReset;
    private int count = 0;
    public UPDOWN() {
        setLayout(new FlowLayout());
        add(new JLabel("Counter"));
        tCount = new JTextField("0", 10);
        tCount.setEditable(false);
        add(tCount);
        bUp = new JButton("Count Up");
        add(bUp);
        bUp.addActionListener(e-> {
                ++count;
            tCount.setText(count + "");
        }
        );
```
1

```java
bDown = new JButton("Count Down");
    add(bDown);
    bDown.addActionListener(e-> {
        count--;
        tCount.setText(count + "");
    }       );
    bReset = new JButton("Reset");
    add(bReset);
    bReset.addActionListener(e->{
        count = 0;
        tCount.setText("0");
    }
);
```
2

```java
        setTitle("AWT Counter");
        setSize(600, 100);
        setVisible(true);
```
3

```java
    public static void main(String[] args) {
        new UPDOWN();
    }
```
5

```java
    // anonymous inner class
    addWindowListener(new WindowListener() {
      public void windowClosing(WindowEvent evt) {
        System.exit(0);
        }
        //  provide an methods for compilation
    public void windowOpened(WindowEvent e) { }
    public void windowClosed(WindowEvent e) { }
    public void windowIconified(WindowEvent e) { }
    public void windowDeiconified(WindowEvent e) { }
    public void windowActivated(WindowEvent e) { }
    public void windowDeactivated(WindowEvent e) { }
        });      }
```
4

AWT Counter   —   □   ×

Counter   -3   Count Up   Count Down   Reset

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

public class UPDOWN extends JFrame {
    private JTextField tCount;
    private JButton bUp, bDown, bReset;
    private int count = 0;
    public UPDOWN() {
        setLayout(new FlowLayout());
        add(new JLabel("Counter"));
        tCount = new JTextField("0", 10);
        tCount.setEditable(false);
        add(tCount);
        bUp = new JButton("Count Up");
        add(bUp);
        bUp.addActionListener(e-> {
                ++count;
            tCount.setText(count + "");
        }
        );
```
1

```java
        bDown = new JButton("Count Down");
        add(bDown);
        bDown.addActionListener(e-> {
            count--;
            tCount.setText(count + "");
        }        );
        bReset = new JButton("Reset");
        add(bReset);
        bReset.addActionListener(e->{
            count = 0;
            tCount.setText("0");
        }
        );
```
2

```java
        setTitle("AWT Counter");
        setSize(600, 100);
        setVisible(true);
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        //addWindowListener(new AdapterClass());
```
3

```java
    public static void main(String[] args) {
        new UPDOWN();
    }
```
5

```java
    private class AdapterClass extends WindowAdapter{
        @Override
        public void windowClosing(WindowEvent evt) {
            System.exit(0);
        }
    }
    public static void main(String[] args) {
        new UPDOWN();
    }
}
```
4

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

public class UPDOWN extends JFrame {
    private JTextField tCount;
    private JButton bUp, bDown, bReset;
    private int count = 0;
    public UPDOWN() {
        setLayout(new FlowLayout());
        add(new JLabel("Counter"));
        tCount = new JTextField("0", 10);
        tCount.setEditable(false);
        add(tCount);
        bUp = new JButton("Count Up");
        add(bUp);
        bUp.addActionListener(e-> {
                ++count;
            tCount.setText(count + "");
        }
        );
```
1

```java
        bDown = new JButton("Count Down");
        add(bDown);
        bDown.addActionListener(e-> {
                count--;
                tCount.setText(count + "");
            }        );
        bReset = new JButton("Reset");
        add(bReset);
        bReset.addActionListener(e->{
                count = 0;
                tCount.setText("0");
            }
        );
```
2

```java
        setTitle("AWT Counter");
        setSize(600, 100);
        setVisible(true);
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        addWindowListener(new AdapterClass());
```
3

```java
    public static void main(String[] args) {
        new UPDOWN();
    }
```
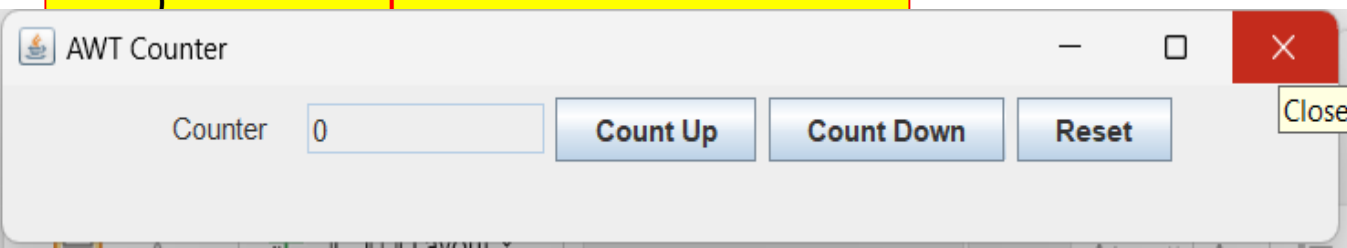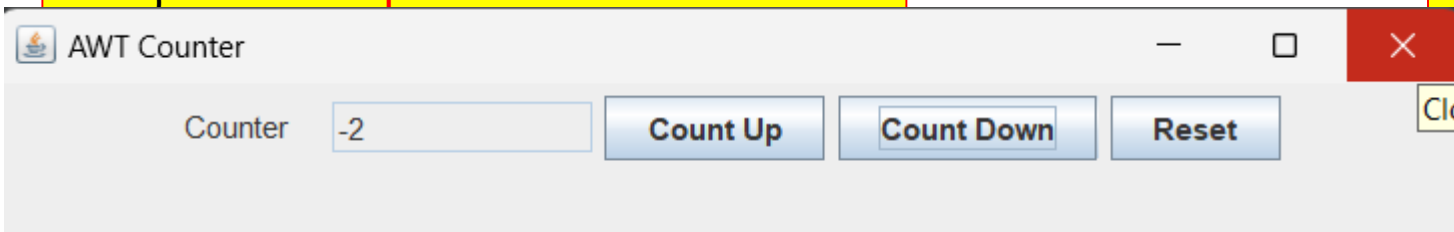*5*

```java
    private class AdapterClass extends WindowAdapter{
        @Override
        public void windowClosing(WindowEvent evt) {
            System.exit(0);
        }
    }
    public static void main(String[] args) {
        new UPDOWN();
    }
```
4

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

public class UPDOWN extends JFrame {
    private JTextField tCount;
    private JButton bUp, bDown, bReset;
    private int count = 0;
    public UPDOWN() {
        setLayout(new FlowLayout());
        add(new JLabel("Counter"));
        tCount = new JTextField("0", 10);
        tCount.setEditable(false);
        add(tCount);
        bUp = new JButton("Count Up");
        add(bUp);
        bUp.addActionListener(e-> {
                ++count;
            tCount.setText(count + "");
        }
    );            1
```

```java
bDown = new JButton("Count Down");
    add(bDown);
    bDown.addActionListener(e-> {
        count--;
        tCount.setText(count + "");
    }     );
    bReset = new JButton("Reset");
    add(bReset);
    bReset.addActionListener(e->{
        count = 0;
        tCount.setText("0");
    }
);
            2
```

```java
setTitle("AWT Counter");
setSize(600, 100);
setVisible(true);

        3
```

```java
public static void main(String[] args) {
    new UPDOWN();
}            5
```

```java
// anonymous inner class
addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent evt) {
        System.exit(0);  // Terminate the program
    }
});            4
```

AWT Counter — □ ✕

Counter 0   Count Up   Count Down   Reset

Close

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

public class UPDOWN extends JFrame {
  private JTextField tCount;
  private JButton bUp, bDown, bReset;
  private int count = 0;
  public UPDOWN() {
    setLayout(new FlowLayout());
    add(new JLabel("Counter"));
    tCount = new JTextField("0", 10);
    tCount.setEditable(false);
    add(tCount);
    bUp = new JButton("Count Up");
    add(bUp);
    bUp.addActionListener(e-> {
            ++count;
        tCount.setText(count + "");
      }
    );                 1
```

```java
bDown = new JButton("Count Down");
    add(bDown);
    bDown.addActionListener(e-> {
        count--;
        tCount.setText(count + "");
      }      );
    bReset = new JButton("Reset");
    add(bReset);
    bReset.addActionListener(e->{
        count = 0;
        tCount.setText("0");
      }
    );

              2
```

```java
setTitle("AWT Counter");
 setSize(600, 100);
 setVisible(true);
          3
```
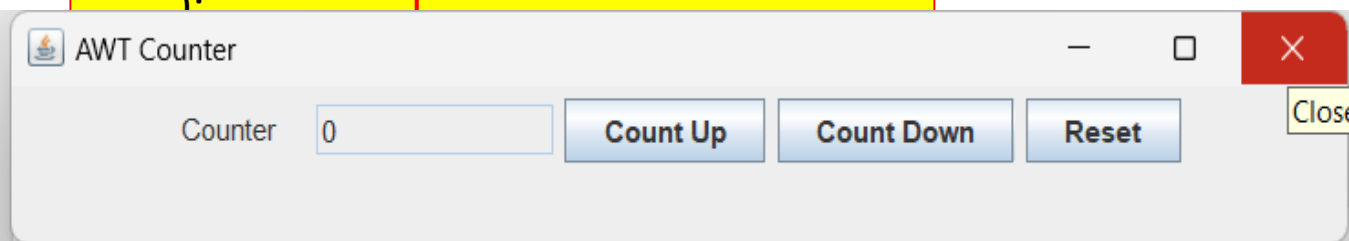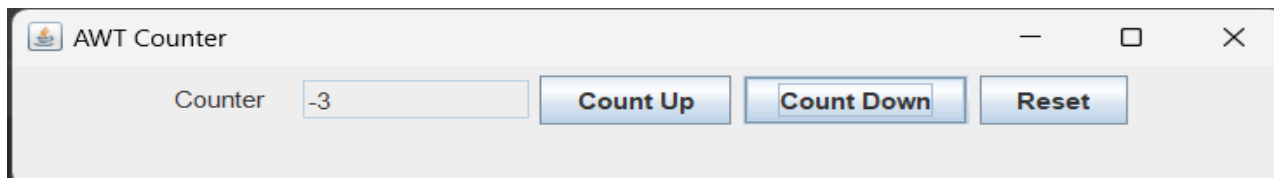
```java
public static void main(String[] args) {
    new UPDOWN();
  }              5
```

```java
// anonymous inner class
 addWindowListener(new WindowAdapter() {
     @Override
     public void windowClosing(WindowEvent evt) {
       System.exit(0);  // Terminate the program
     }
   });              4
```

AWT Counter

Counter   -3 | Count Up | Count Down | Reset

```
/*private class ButtonsListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        String btnType = e.getActionCommand();
        if (btnType.equals("Count Up")) {
            ++count;
        } else if (btnType.equals("Count Down")) {
            --count;
        } else {
            count = 0;
        }
        tCount.setText(count + "");
    }
  }*/
```

```
/* private class ButtonsListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        Button source = (Button)e.getSource();
        if (source == bUp) {
            ++count;
        } else if (source == bDown) {
            --count;
        } else {
            count = 0;
        }
        tCount.setText(count + "");
    }
  }*/
```

```java
private JButton submit;
private JTextField Name;
private JTextField Email;
private JTextArea ProjectDescription;
private JList<String> Skills;
private JComboBox Experience;
private JCheckBox UG;
private JRadioButton FullTime;
private JRadioButton PartTime;
private JScrollPane jp;
public GUI(){
    submit=new JButton("submit");
    Name=new JTextField(10);
    Email=new JTextField(10);
    ProjectDescription=new JTextArea(5,20);
    jp=new JScrollPane(ProjectDescription);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(500,400);
    setVisible(true);
    setLayout(new GridLayout(8,2));
    String[] str={"Python", "Java", "Go"};
    String[] str1={"1", "2", "3"};
    Skills=new JList<>(str);
    Experience=new JComboBox(str1);
    UG=new JCheckBox("yes");

    FullTime=new JRadioButton("1");
    PartTime=new JRadioButton("2");
    ButtonGroup bg=new ButtonGroup();
    bg.add(FullTime);
    bg.add(PartTime);
    submit.addActionListener(new add());
    add(new Label("NAME"));
    add(Name);
    add(new Label("EMAIL"));
    add(Email);
    add(new Label("Project Description"));
    //add(ta1);
    add(ProjectDescription);
    add(new Label("Skills"));
    add(Skills);
    add(new Label("Experience"));
    add(Experience);
    add(UG);

    add(FullTime);
    add(PartTime);
    add(submit);

}
```
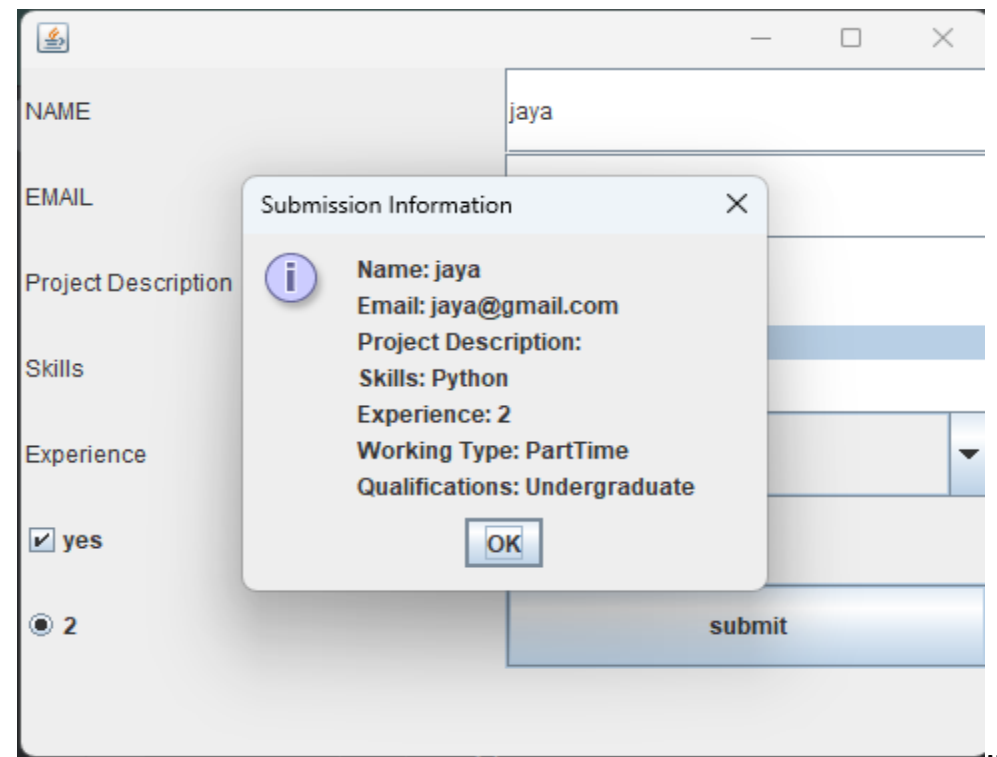
```java
private class add implements ActionListener{
    public void actionPerformed(ActionEvent e){
        String SName= Name.getText();
        String SEmail=Email.getText();
        String SProject=ProjectDescription.getText();
        String SSkill=(String)Skills.getSelectedValue();
        String SExperience=(String)Experience.getSelectedItem();
        String SWorkingType=FullTime.isSelected()? "FullTime":"PartTime";
        String SUG="none";
        if (UG.isSelected()) {
            SUG="Undergraduate";
        }
        String About = String.format(
            "Name: %s\nEmail: %s\nProject Description: %s\nSkills: %s\nExperience: %s\nWorking Type: %s\nQualifications: %s",
            SName, SEmail, SProject, SSkill, SExperience, SWorkingType, SUG
        );

        // Display the message in a dialog                    WARNING,INFORMATION,QUESTION
        JOptionPane.showMessageDialog(GUI.this, About, "Submission Information", JOptionPane.INFORMATION_MESSAGE);
    }
}
    public static void main(String[] args){
        new GUI();   }}
```

The actionPerformed Method
- An actionPerformed method must have only one parameter, ActionEvent
- public void actionPerformed(ActionEvent e) {
  if (e.getActionCommand().equals("click") . . .
  }

```java
import java.awt.*;            // AWT components, and layout managers
import java.awt.event.*; // event-handlers
import javax.swing.*; // Swing components
```

```java
public class SampleGUI extends JFrame {
    protected Container contentPanel;
    SampleGUI() {
    // Access default content pane, AWT container
    contentPanel = this.getContentPane();
    // add Button
    JButton button=new JButton("Hello GUI !");
    contentPanel.add(button);
    // close window
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    //resize frame to arrange components tightly
    pack();
    setTitle("GUI");  //set frame title
    setSize(250, 70); //set Frame size
    //display frame
    setVisible(true);
    button.addActionListener(new AddCount(button)) }  }
```

```java
public class SampleGUI extends JFrame {
    protected JPanel contentPanel;
    SampleGUI() {
    // set contentpanel to swing JPanel
    contentPanel = new JPanel();
    this.setContentPane(contentPanel);
    JButton button=new JButton("Hello GUI !");
    contentPanel.add(button );
    // close window
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    //resize frame to arrange components tightly
    pack();
    setTitle("GUI"); //set frame title
    setSize(250, 70); //set Frame size
    //display frame
    setVisible(true);
    button.addActionListener(new AddCount(button)) }  }
```

```java
public class SampleGUI extends JFrame {
    protected JPanel contentPanel;
    SampleGUI() {
        // set contentpanel to swing JPanel
        contentPanel = new JPanel();
        this.setContentPane(contentPanel);
        JButton button=new JButton("Hello GUI !");
        contentPanel.add(button );
        // close window
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //resize frame to arrange components tightly
        pack();
        setTitle("GUI");   //set frame title
        setSize(250, 70); //set Frame size
        //display frame
        setVisible(true);
        button.addActionListener(new
        AddCount(button)) }  }
```

```java
public class  AddCount implements ActionListener
{
        private int count;
        AddCount(JLabel label) {
        this.label = label;
        count = 0; }
        // When button clicked
        public void actionPerformed(ActionEvent e) {
            // increment counter
        count++;
            // update button's text
    button.setText("I am clicked " + count + " times");
                        }
}
```

Labels, TextAreas, and Textfields

Component

- **JTextArea** is an object on the screen which is named and can hold _multiline text messages_.
- It is a _scrollable object_, so the object can hold more text than is visible on the screen at one time.
- an _ActionEvent_ is raised when the user types the retum/enter key.
- Initialization requires the number of lines of text and the number of characters per line to be specified.

```
JTextArea <variable1>;
<variable1> =new JTextArea(<lines>. <chars>);
add(<variable1>);
```

Example:

```
JTextArea echoArea;
echoArea =new JTextArea(5, 40);
add(echoArea);
```

//place a 5 line by 40 character textarea

A **JTextField** is an object which holds a _single line of text_. Either one can be used to gather text from the keyboard or merely to display messages from the application.

```
JTextField <variable>;
JTextField typing;
typing= new JTextField(40);
add( typing);
```

Event handler

When the user types in a JTextField or JTextArea and hits the return key, the application can handle that event by writing additional code in its actionPerformed event handler:

```
public void actionPerformed (ActionEvent e) {
if (e.getSource() ==<variable>) {
String s= <variable>.getText();
<action> } }
```

Here, <variable> refers to the name of the JTextArea or JTextArea vmiable that was declared and placed in the frame by the constructor, like typing in the example above.
A better solution is to use an **internal class** that listens specifically to the JTextFi el d typing:
In this case, the handler need not check for the source of the triggering event; it must be the user pressing the return (or enter) key in the JTextFiel typing.

```
private class TextHandler implements ActionListener{
public void actionPerformed (ActionEvent e) {
String s= echoArea.getText( );
echoArea.setText(echoArea.getText() + s + "\n");     }}
```

A JLabel is an object whose string value can be placed inside a frame to label another object, such as a JTextField.
add(new JLabel(HFahrehheit")};

Buttons

A *button* is a named object on the screen which can be selected by a mouse click.

Because any number of variables can be declared with class JButton and placed in the application, button handlers are usually implemented via a separate class, rather than by the application itself.

> **JButton** <variable> = new **JButton**(<string>);
> **add**(<variable>);
> <variable>. **addActionListene**r(<listener>) ;

The class which handles the user selecting the button must implement the ActionListener interface. The listener class must implement an actionPerformed method to handle the button selection event:

```
JButton clearButton =new JButton("Clear");
add(clearButton);
public void actionPerformed (ActionEvent e) {
if (e.getSource() =<variable>) {
<action>  }}
```

```
clearButton.addActionListener(new ClearButtonHandler( ) ) ;

private class ClearButtonHandler implements ActionListener {
public void actionPerformed(ActionEvent e) {
repaint();  }}
```

Note that the **repaint method** is a **JPanel method**; thus, this class works as written only if it is an inner class to the application class which **extends JPanel**. However, An external class requires writing a constructor which takes the application as an argument.

Note that using an external class makes it more difficult for a listener class to determine which button was selected since the

For the program to **handle mouse clicks**, the **MouseListener interface** must be implemented, either by *the application itself or by a mouse handler class*.

If the listener is being handled directly by the class, then the class must specify the listener in its implements clause and activate the listener, usually inside the constructor:

Application itself handle mouse clicks
```
 public class MyApplication extends JPanel implements MouseListener
{
public MyApplication( ) {
        …
        addMouseListener(this);
        …}}
```

Separate class to handle mouse clicks
```
MyApplication extends JPanel {
public MyApplication( ) {
        …
        addMouseListener(new MouseHandler());
        … }
private class MouseHandler implements MouseListener
{           …}
```

The alternative is to use a separate class to handle mouse clicks. Commonly, this class is an inner class to the application, so that the mouse handler has access to all of the application's methods, particularly the graphics context:

An advantage to using a separate class is that Java provides a MouseAdapter class. This means that the separate class can extend the MouseAdapter class (which an application cannot do), oveniding exactly the methods for which actions are to be provided. In most instances, this is usually only the mouseClicked or mousePressed method.

```
public class MyApplication extends JPanel {
public MyApplication( ){ …
addMouselistener(new MouseHandler()); …}
private class MouseHandler extends MouseAdapter {
public void mouseClicked(MouseEvent e) { <action>
}}
```

Capture the x-y pixel coordinates where that event occurs on the frame.
```
public void mouseClicked(MouseEvent e) {
        x = e.getX();
        y = e.getY();
}
```

Combo Boxes

A *combo box* is an object on the screen that offers several options to be selected by the user; it is like a pull-down menu, but it can be placed anywhere within the application.

```
JComboBox <variable> ;
JComboBox combo;
```

```
<variable>= new JComboBox( );
<variable>.addItem(<stringl>);
…
add(<variable>);
<variable>. addItemListener(<listener));
```

```
private class ComboHandler implements ActionListener {
public void actionPerformed (ActionEvent e) {
String s =(String)combo.getItem( );
if (s.equals(<stringl>)) {
        <actionl> in response to a selection of <stringl> }
else if (s.equals(<string2>)){
        <action2> in response to a selection of <string2> }
   } }
```

```
combo= new JComboBox( );
combo.addItem("North");
combo.addItem("East");
combo.addItem("South");
combo.addItem("West");
add(combo);
combo.addItemListener(new ComboHandler());
```

Mouse Motion

```
addMouseMotionListener(<listener>);
```

```
public void mouseDragged(MouseEvent e) { }
public void mouseMoved(MouseEvent e) { }
```

Simple GUI Interface

Consider the design of a simple interface, in which the user can insert rectangles and place texts in arbitrary locations of the frame. The user should be able to accomplish this as simply as possible, and so providing buttons, menus, and text typing areas, as well as handling mouse-click actions on the screen is essential

combo =>User can select Nothing, Rectangle, or Message from this menu.
echoArea=> A text area for reporting the most recent event that has occured.
typing=> A text field for entering user input.