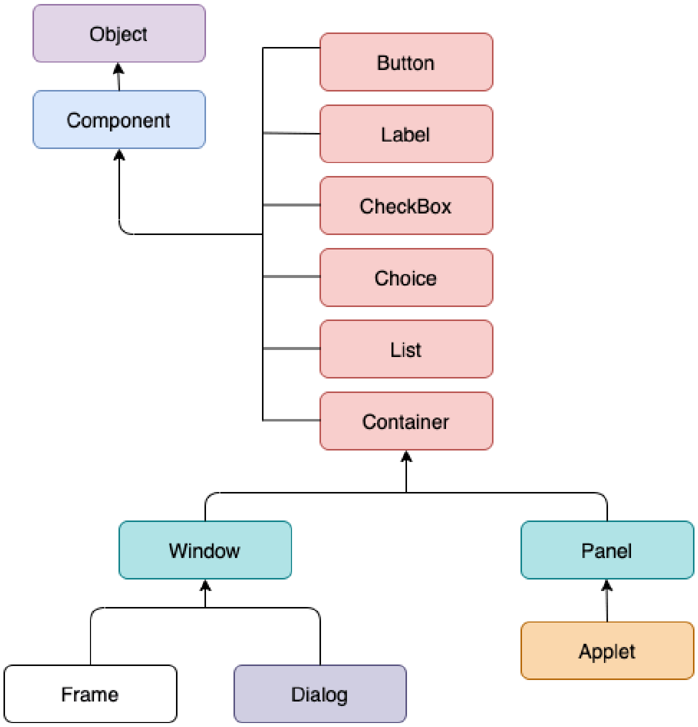
Unit - IV Event handling using Abstract Window Toolkit (AWT) & Swings Components

Is an API used to create **Graphical User Interface** (GUI).
Java AWT components are platform-dependent.
The java.awt package contains AWT API classes such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List, and so on.



- Component Class

  The component class stands at the top of the AWT hierarchy, is an abstract class that contains all the properties of the component visible on the screen. The Component object contains information about the currently selected foreground and background color. It also has information about the currently selected text color.

- Container

  AWT provides containers like panels, frames, and dialogues to organize and group components in the Application. It is a subclass of the Component class.

  Window: A window can be defined as a container that doesn't contain any border or menu bar. It creates a top-level view. However, we must have a frame, dialog, or another window for creating a window. The Window class extends the Container class, which means it can contain other components, such as buttons, labels, and text fields.

  Frame: The Frame is the container that contains the title bar and border and can have menu bars.

  Panel: The panel can be defined as a container that can be used to hold other components. However, it doesn't contain the title bar, menu bar, or border.

  Panel is a container class. It is a lightweight container that can be used for grouping other components together within a window or a frame.

- Layout Managers

  Layout Managers are responsible for arranging data in the containers some of the layout managers are BorderLayout, FlowLayout, etc.

- Event Handling

  AWT allows the user to handle the events like mouse clicks, key presses, etc. using event listeners and adapters.

Java AWT (Abstract Window Toolkit) provides a set of classes to create graphical user interfaces (GUIs). Below are some common AWT components you can use to build GUI applications:

1. Button (Button)

- A button that can trigger an action when clicked.

```
Button button = new Button("Click Me");
button.setBounds(50, 50, 100, 30);
add(button);
```

2. Label (Label)

- A simple text element to display information.

```
Label label = new Label("Hello, AWT!");
label.setBounds(50, 100, 150, 30);
add(label);
```

3. TextField (TextField)

- A single-line text input field for user input.

```
TextField textField = new TextField();
textField.setBounds(50, 150, 200, 30);
add(textField);
```

4. TextArea (TextArea)

- A multi-line text input field.

```
TextArea textArea = new TextArea();
textArea.setBounds(50, 200, 250, 100);
add(textArea);
```

5. Checkbox (Checkbox)

- A checkable box that can be either checked or unchecked.

```
Checkbox checkbox = new Checkbox("Accept Terms");
checkbox.setBounds(50, 350, 150, 30);
add(checkbox);
```

6. Choice (Choice)

- A drop-down list for selecting one option from many.

```
Choice choice = new Choice();
choice.setBounds(50, 400, 150, 30);
choice.add("Option 1");
choice.add("Option 2");
choice.add("Option 3");
add(choice);
```

7. List (List)

- A list of items from which the user can select one or more.

```
List list = new List();
list.setBounds(50, 450, 150, 100);
list.add("Item 1");
list.add("Item 2");
list.add("Item 3");
add(list);
```

8. Panel (Panel)

- A container for grouping other components.

```
Panel panel = new Panel();
panel.setBounds(50, 600, 250, 150);
panel.setBackground(Color.GRAY);
add(panel);
```

9. Menu (Menu) and MenuItem (MenuItem)

- Used for creating menus and items within the menu.

```
Menu menu = new Menu("File");
MenuItem item = new MenuItem("Open");
menu.add(item);
MenuBar menuBar = new MenuBar();
menuBar.add(menu);
setMenuBar(menuBar);
```

10. Frame (Frame)

- A top-level window for the GUI application.

```
Frame frame = new Frame("AWT Example");
frame.setSize(400, 400);
frame.setLayout(null);  // Absolute positioning
frame.setVisible(true);
```

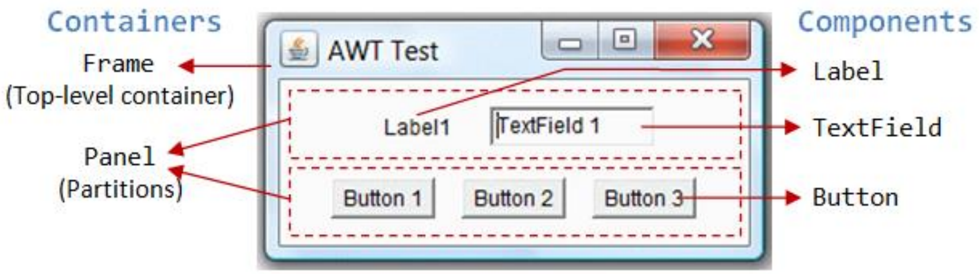
Example Program

```
import java.awt.*;
public class AWTExample {
  public static void main(String[] args) {
    Frame frame = new Frame("AWT Components Example");
    // Create and position components

    Button button = new Button("Click Me");
    button.setBounds(50, 50, 100, 30);
Label label = new Label("Enter your name:");
    label.setBounds(50, 100, 150, 30);
    TextField textField = new TextField();
    textField.setBounds(50, 150, 200, 30);
// Add components to the frame
    frame.add(button);
  frame.add(label);
    frame.add(textField);
// Set frame properties
    frame.setSize(400, 400);
    frame.setLayout(null);
    frame.setVisible(true);
     // Close the frame when the user closes the window
    frame.addWindowListener(new java.awt.event.WindowAdapter()
 {
      public void windowClosing(java.awt.event.WindowEvent
windowEvent) {         System.exit(0);       }    });
  }
}
```

AWT components are part of the java.awt package and can be used to build simple user interfaces.

Containers

Such as Frame and Panel, are used to hold components in a specific layout (such as FlowLayout or GridLayout)



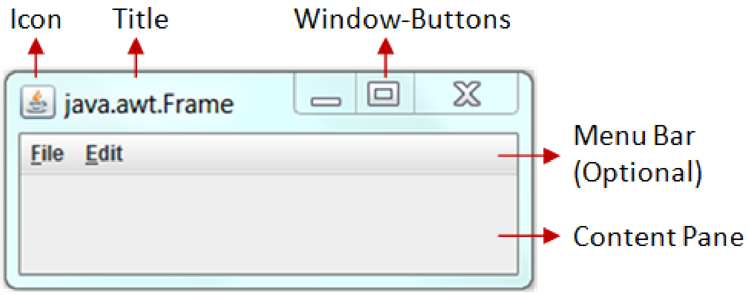
- In the above figure, there are three containers: a Frame and two Panels. A Frame is the *top-level container* of an AWT program.
- A Frame has a title bar (containing an icon, a title, and the minimize/maximize/close buttons), an optional menu bar and the content display area.
- A Panel is a *rectangular area* used to group related GUI components in a certain layout.
- There are five components: a Label (providing description), a TextField (for users to enter text), and three Buttons (for user to trigger certain programmed actions).

In a GUI program, a component must be kept (or added) in a container. You need to identify a container to hold the components. Every container has a method called add(Component c).

```
Panel pnl = new Panel ();           // Panel is a container
Button btn = new Button ("Press"); // Button is a component
pnl.add(btn);                       // The Panel container adds a Button component
```

Frame:

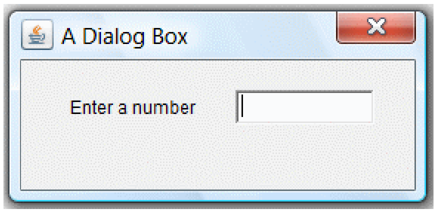
1. A Frame provides the "main window" for your GUI application. It has a title bar (containing an icon, a title, the minimize/maximize/restore-down and close buttons), an optional menu bar, and the content display area.



2. To write a GUI program, we typically start with a subclass extending from java.awt.Frame to inherit the main window

Dialog

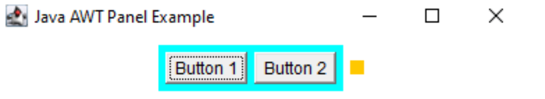
1. An AWT Dialog is a *"pop-up window"* used for interacting with the users. A Dialog has a title-bar (containing an icon, a title and a close button) and a content display area, as illustrated.



Panel

1. Panel class is a fundamental component for creating graphical user interfaces.
2. It offers a straightforward way to organize and group various GUI elements.

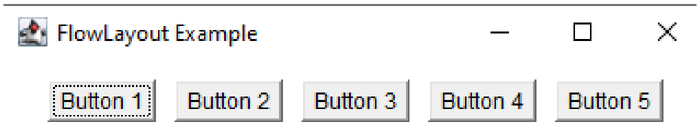
```
import java.awt.*;
import java.awt.event.*;
public class PanelExample
{
        public static void main(String[] args)
        {
    Frame frame = new Frame("Java AWT Panel Example");
    Panel panel1 = new Panel();
    Panel panel2 = new Panel();

  // Set the layout manager for panel1
    panel1.setLayout(new FlowLayout());

  // Add components to panel1
    Button button1 = new Button("Button 1");
    Button button2 = new Button("Button 2");
    panel1.add(button1);
    panel1.add(button2);

  // Set background colors for panels
    panel1.setBackground(Color.CYAN);
    panel2.setBackground(Color.orange);

  // Add panels to the frame
    frame.add(panel1);
    frame.add(panel2);
    frame.setSize(400, 200);
    frame.setLayout(new FlowLayout());
    frame.setVisible(true);

    frame.addWindowListener(new WindowAdapter()
    {
       public void windowClosing(WindowEvent we)
       {            System.exit(0);     }
    });
 }
}
```



Layout Manager

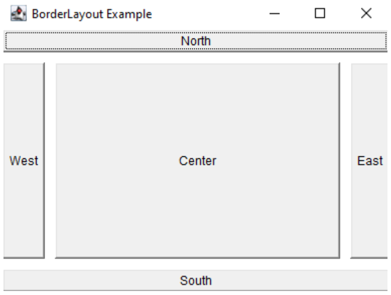
- Layout managers define how components are arranged within a container.
- Java provides several layout managers to suit various design needs.
- The Layout managers enable us to control the way in which visual components are arranged in the GUI forms by determining the size and position of components within the containers.

1. FlowLayout
   - ➢ FlowLayout is a simple layout manager that arranges components in a row, left to right, wrapping to the next line as needed.
   - ➢ It is ideal for scenarios where components need to maintain their natural sizes and maintain a flow-like structure.



```
import java.awt.*;
import java.awt.event.*;
public class FlowLayoutExample {
   public static void main(String[] args) {
      Frame frame = new Frame("FlowLayout Example");
      frame.setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));
      Button btn1 = new Button("Button 1");
      Button btn2 = new Button("Button 2");
      Button btn3 = new Button("Button 3");
      Button btn4 = new Button("Button 4");
      Button btn5 = new Button("Button 5");
      frame.add(btn1);
      frame.add(btn2);
      frame.add(btn3);
      frame.add(btn4);
      frame.add(btn5);
      frame.setSize(400, 200);
      frame.setVisible(true);
      frame.addWindowListener(new WindowAdapter(){
             public void windowClosing(WindowEvent e)
             {
                          frame.dispose();
             }
             });
   }
}
```

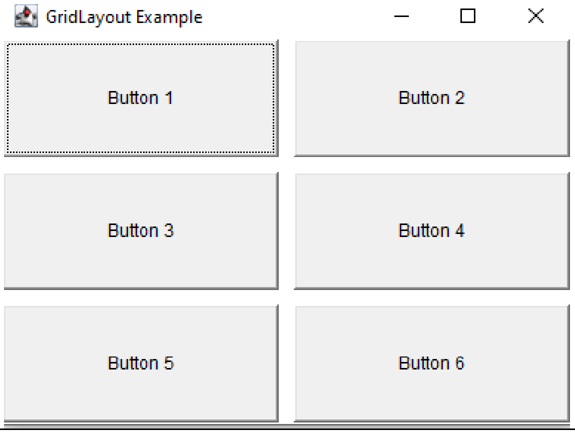
2. BorderLayout
   - ➢ BorderLayout divides the container into five regions: NORTH, SOUTH, EAST, WEST, and CENTER.
   - ➢ Components can be added to these regions, and they will occupy the available space accordingly.
   - ➢ This layout manager is suitable for creating interfaces with distinct sections, such as a title bar, content area and status bar.



```
import java.awt.*;
import java.awt.event.*;
public class BorderLayoutExample {
  public static void main(String[] args) {
        Frame frame = new Frame("BorderLayout Example");
    frame.setLayout(new BorderLayout(10, 10)); // 10px horizontal & vertical gap
    Button btnNorth = new Button("North");
    Button btnSouth = new Button("South");
    Button btnEast = new Button("East");
    Button btnWest = new Button("West");
    Button btnCenter = new Button("Center");
    frame.add(btnNorth, BorderLayout.NORTH);
    frame.add(btnSouth, BorderLayout.SOUTH);
    frame.add(btnEast, BorderLayout.EAST);
    frame.add(btnWest, BorderLayout.WEST);
    frame.add(btnCenter, BorderLayout.CENTER);
    frame.setSize(400, 300);
    frame.setVisible(true);
    frame.addWindowListener(new WindowAdapter()
    {
        public void windowClosing(WindowEvent e)
        {
          frame.dispose();
         }
    });
  }
}
```

3. GridLayout

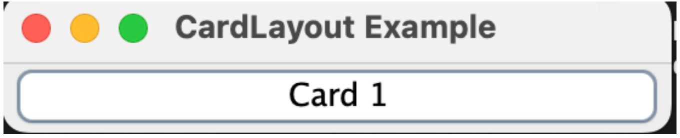
- GridLayout arranges components in a grid with a specified number of rows and columns. Each cell in the grid can hold a component.
- This layout manager is ideal for creating a uniform grid of components, such as a calculator or a game board.



```
import java.awt.*;
import java.awt.event.*;
public class GridLayoutExample {
  public static void main(String[] args) {
    Frame frame = new Frame("GridLayout Example");
    frame.setLayout(new GridLayout(3, 2, 10, 10));
    Button btn1 = new Button("Button 1");
    Button btn2 = new Button("Button 2");
    Button btn3 = new Button("Button 3");
    Button btn4 = new Button("Button 4");
    Button btn5 = new Button("Button 5");
    Button btn6 = new Button("Button 6");
    frame.add(btn1);
    frame.add(btn2);
    frame.add(btn3);
    frame.add(btn4);
    frame.add(btn5);
    frame.add(btn6);
    frame.setSize(400, 300);
    frame.setVisible(true);
    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        frame.dispose();
      }
    });
  }
}
```

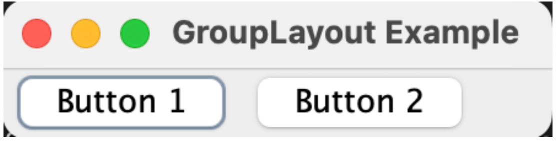
4. CardLayout

- CardLayout allows components to be stacked on top of each other, like a deck of cards.
- Only one component is visible at a time, and you can switch between components using methods like next() and previous().
- This layout is useful for creating wizards or multi-step processes.



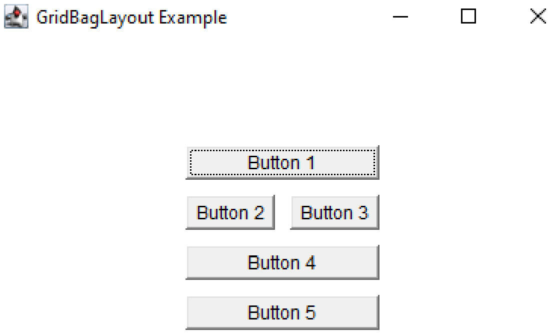
5. GroupLayout

- GroupLayout is a versatile and complex layout manager that provides precise control over the positioning and sizing of components.
- It arranges components in a hierarchical manner using groups.
- GroupLayout is commonly used in GUI builders like the one in NetBeans IDE.



6. GridBagLayout

- GridBagLayout is a powerful layout manager that allows you to create complex layouts by specifying constraints for each component.
- It arranges components in a grid, but unlike GridLayout, it allows components to span multiple rows and columns and have varying sizes.



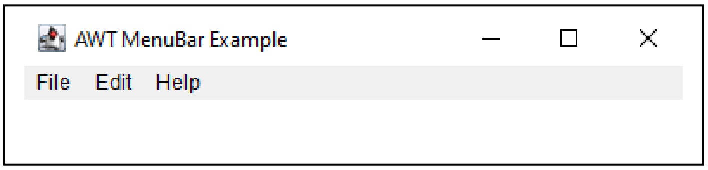
```
import java.awt.*;
import java.awt.event.*;
public class GridBagLayoutExample {
  public static void main(String[] args) {
    Frame frame = new Frame("GridBagLayout Example");
    GridBagLayout gridBagLayout = new GridBagLayout();
    GridBagConstraints gbc = new GridBagConstraints();
    frame.setLayout(gridBagLayout);
    Button btn1 = new Button("Button 1");
    Button btn2 = new Button("Button 2");
    Button btn3 = new Button("Button 3");
    Button btn4 = new Button("Button 4");
    Button btn5 = new Button("Button 5");
    gbc.gridx = 0; gbc.gridy = 0;
    gbc.gridwidth = 2; // Span 2 columns
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.insets = new Insets(5, 5, 5, 5); // Padding
    frame.add(btn1, gbc);

    gbc.gridx = 0; gbc.gridy = 1; // Button 2 (Row 1, Col 0)
    gbc.gridwidth = 1; // Reset span to 1
    frame.add(btn2, gbc);

    gbc.gridx = 1; gbc.gridy = 1; // Button 3 (Row 1, Col 1)
    frame.add(btn3, gbc);
    gbc.gridx = 0; gbc.gridy = 2;   // Button 4 (Row 2, Col 0, spans 2 columns)
    gbc.gridwidth = 2; // Span 2 columns
    gbc.fill = GridBagConstraints.HORIZONTAL;
    frame.add(btn4, gbc);

    // Button 5 (Row 3, Col 0, spans 2 columns)
    gbc.gridx = 0; gbc.gridy = 3;
    gbc.gridwidth = 2;
    frame.add(btn5, gbc);
    frame.setSize(400, 300);
    frame.setVisible(true);
    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        frame.dispose();
      }
    });
  }
}
```

Menu-Bar, Menus & File Dialog

Menu Bar & Menus

- ➢ The MenuBar class provides menu bar bound to a frame and is platform specific.
- ➢ MenuItem is a class that represents a simple labeled menu item within a menu.
- ➢ It can be added to a menu using the Menu.add(MenuItem mi) method.
  - ❖ MenuBar (MenuBar) → the top-level menu bar.
  - ❖ Menu (Menu) → a drop-down menu inside the menu bar.
  - ❖ MenuItem (MenuItem) →Individual selectable items inside a menu.
  - ❖ CheckboxMenuItem (CheckboxMenuItem) →A menu item with a checkbox.
  - ❖ Separators (addSeparator()) →Used to separate menu items visually.

```
import java.awt.*;
import java.awt.event.*;
public class MenuBarExample {
  public static void main(String[] args) {
    Frame frame = new Frame("AWT MenuBar Example");
    MenuBar menuBar = new MenuBar();
    Menu fileMenu = new Menu("File");
    Menu editMenu = new Menu("Edit");
    Menu helpMenu = new Menu("Help");

    MenuItem newItem = new MenuItem("New");
    MenuItem openItem = new MenuItem("Open");
    MenuItem saveItem = new MenuItem("Save");
    MenuItem exitItem = new MenuItem("Exit");
    exitItem.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        frame.dispose();          }      });
    // Add Menu Items to "File" Menu
    fileMenu.add(newItem);
    fileMenu.add(openItem);
    fileMenu.add(saveItem);
    fileMenu.addSeparator(); // Adds a separator line
    fileMenu.add(exitItem);
    // Create Menu Items for "Edit" Menu
    MenuItem cutItem = new MenuItem("Cut");
    MenuItem copyItem = new MenuItem("Copy");
    MenuItem pasteItem = new MenuItem("Paste");
    // Add Menu Items to "Edit" Menu
    editMenu.add(cutItem);
    editMenu.add(copyItem);
    editMenu.add(pasteItem);
    // Create Menu Items for "Help" Menu
    MenuItem aboutItem = new MenuItem("About");
    // Add Menu Item to "Help" Menu
    helpMenu.add(aboutItem);
    // Add Menus to MenuBar
    menuBar.add(fileMenu);
    menuBar.add(editMenu);
    menuBar.add(helpMenu);
    // Set MenuBar to Frame
    frame.setMenuBar(menuBar);
    // Set Frame properties
    frame.setSize(400, 300);
    frame.setVisible(true);
    // Add Window Listener to close the Frame on clicking close button
    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        frame.dispose();
      }      });
  }
}
```



File Dialog

- Java's FileDialog class is part of the AWT (Abstract Window Toolkit) package, specifically designed for creating a file dialog box.
- File Dialog control represents a dialog window from which the user can select a file.
- This dialog box serves as an intermediary between the user and the underlying file system, allowing users to browse through directories, select files, and perform operations on the selected files.
  - static int LOAD -- This constant value indicates that the purpose of the file dialog window is to locate a file from which to read.
  - static int SAVE -- This constant value indicates that the purpose of the file dialog window is to locate a file to which to write.

```
import java.awt.*;
import java.awt.event.*;
public class AWTFileDialogExample {
  public static void main(String[] args) {
    Frame frame = new Frame("AWT FileDialog Example");
    Button open = new Button("Open File");
    Button save = new Button("Save File");
    frame.setLayout(new FlowLayout());
    frame.add(open);
    frame.add(save);
    open.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        // Create FileDialog for opening files
        FileDialog fileDialog = new FileDialog(frame, "Open File", FileDialog.LOAD);
        fileDialog.setVisible(true);
        // Get selected file
        String fileName = fileDialog.getFile();
        String directory = fileDialog.getDirectory();
        if (fileName != null) {               System.out.println("Selected File: " + directory + fileName);           }
      }      });
    // Add Action Listener for Save File button
    save.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        // Create FileDialog for saving files
        FileDialog fileDialog = new FileDialog(frame, "Save File", FileDialog.SAVE);
        fileDialog.setVisible(true);
        // Get selected file
        String fileName = fileDialog.getFile();
        String directory = fileDialog.getDirectory();
        if (fileName != null) {              System.out.println("File to Save: " + directory + fileName);           }
      }      });
    frame.setSize(400, 200);
    frame.setVisible(true);
    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        frame.dispose();
      }
    });
  }
}
```

Swing

- Swing is a Java Foundation Classes [JFC] library and an extension of the Abstract Window Toolkit [AWT].
- Java Swing offers much-improved functionality over AWT, new components, expanded components features, and excellent event handling.
- AWT has a limited implementation, not quite capable of providing the components required for developing complex GUIs required in modern commercial applications.
- Some features
  - Swing is Provided to Design Graphical User Interfaces
  - Swing is an Extension library to the AWT (Abstract Window Toolkit)
  - Swing can be used to build (Develop) The Standalone swing GUI Apps as Servlets and Applets
  - Swing Supports a Pluggable look and feel and Swing provides more powerful rich components like tables, lists, Scrollpanes, Colourchooser, tabbed pane, etc.
  - Platform Independent: Applications developed using Swing can run on any platform that supports Java, without requiring modifications.

- 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.
-

Difference between Java Swing and Java AWT

Java AWT	Java Swing
Java AWT is an API to develop GUI applications in Java.	Swing is a part of Java Foundation Classes and is used to create various applications.
Components of AWT are heavy weighted.	The components of Java Swing are lightweight.
Components are platform dependent.	Components are platform independent.
Execution Time is more than Swing.	Execution Time is less than AWT.
AWT components require java.awt package.	Swing components requires javax.swing package.

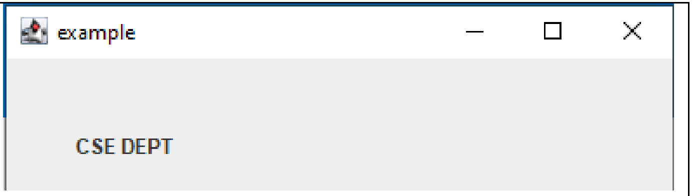
Swing Components

- All the components in swing like JButton, JComboBox, JList, JLabel are inherited from the JComponent class which can be added to the container classes.
- Containers are the windows like frame and dialog boxes.
- Basic swing components are the building blocks of any gui application.
- Methods like setLayout override the default layout in each container.
- Containers like JFrame and JDialog can only add a component to itself.

JLabel Class

- It is used for placing text in a container. It also inherits JComponent class.

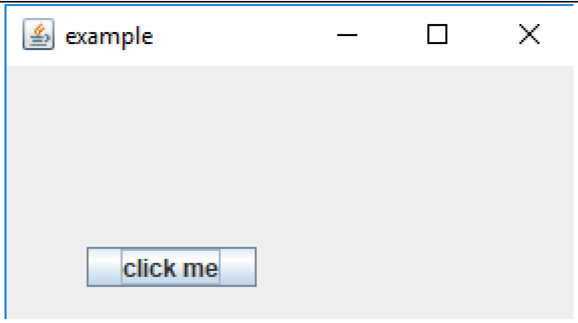
```
import javax.swing.*;
public class textFieldexample{
public static void main(String args[]) {
            JFrame a = new JFrame("example");
            JLabel b1;
            b1 = new JLabel("CSE DEPT");
            b1.setBounds(40,40,90,20);
            a.add(b1);
            a.setSize(400,400);
            a.setLayout(null);
            a.setVisible(true);
}
}
```



JButton Class

- It is used to create a labelled button.
- Using the ActionListener it will result in some action when the button is pushed.
- It inherits the AbstractButton class and is platform independent.

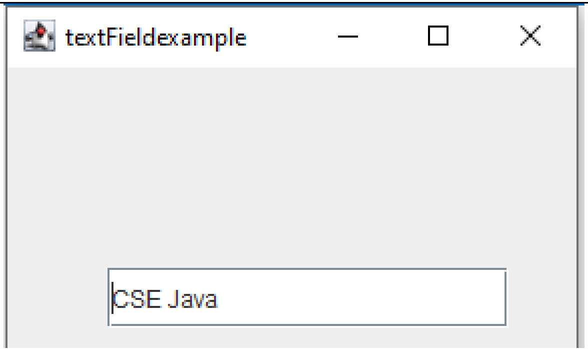
```
import javax.swing.*;
public class example{
public static void main(String args[]) {
JFrame a = new JFrame("example");
JButton b = new JButton("click me");
b.setBounds(40,90,85,20);
a.add(b);
a.setSize(300,300);
a.setLayout(null);
a.setVisible(true);
}
}
```



JTextField Class

- It inherits the JTextComponent class and it is used to allow editing of single line text.

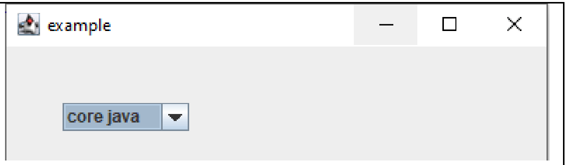
```
import javax.swing.*;
public class textFieldexample{
public static void main(String args[]) {
JFrame a = new JFrame("textFieldexample");
JTextField b = new JTextField("CSE Java");
b.setBounds(50,100,200,30);
a.add(b);
a.setSize(300,300);
a.setLayout(null);
a.setVisible(true);
}
}
```



JComboBox Class

- It inherits the JComponent class and is used to show pop up menu of choices.

```
import javax.swing.*;
public class textFieldexample{
JFrame a;
textFieldexample(){
a = new JFrame("example");
String courses[] = { "core java","advance java", "java
servlet"};
JComboBox c = new JComboBox(courses);
c.setBounds(40,40,90,20);
a.add(c);
a.setSize(400,400);
a.setLayout(null);
a.setVisible(true);
}
public static void main(String args[]){
 new textFieldexample();   }
}
```



JRadioButton

- We use the JRadioButton class to create a radio button.
- Radio button is use to select one option from multiple options.
- It is used in filling forms, online objective papers and quiz.
- We add radio buttons in a ButtonGroup so that we can select only one radio button at a time.
- We use “ButtonGroup” class to create a ButtonGroup and add radio button in a group.

Steps to Group the radio buttons together.

Create a ButtonGroup instance by using "ButtonGroup()" Method.

ButtonGroup G = new ButtonGroup()

Now add buttons in a Group "G", with the help of "add()" Method.

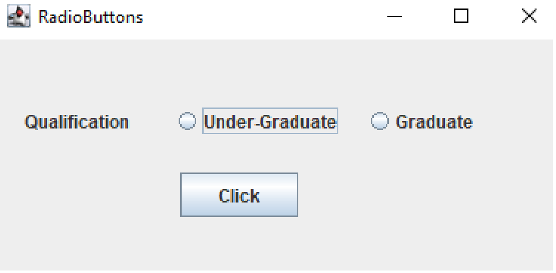
G.add(Button1);
G.add(Button2);

isSelected() :

it will return a Boolean value true or false, if a JRadioButton is selected it Will return true otherwise false.

JRadioButton.isSelected();

```
import javax.swing.*;
class Demo extends JFrame {
        JRadioButton jRadioButton1;
        JRadioButton jRadioButton2;
        JButton jButton;
        ButtonGroup G1;
        JLabel L1;
        public Demo()
        {

        this.setLayout(null);
        jRadioButton1 = new JRadioButton();
        jRadioButton2 = new JRadioButton();
        jButton = new JButton("Click");
        G1 = new ButtonGroup();
        L1 = new JLabel("Qualification");
        jRadioButton1.setText("Under-Graduate");
        jRadioButton2.setText("Graduate");

        jRadioButton1.setBounds(120, 30, 120, 50);
        jRadioButton2.setBounds(250, 30, 80, 50);
        jButton.setBounds(125, 90, 80, 30);
        L1.setBounds(20, 30, 150, 50);
        this.add(jRadioButton1);
        this.add(jRadioButton2);
        this.add(jButton);
        this.add(L1);
        G1.add(jRadioButton1);
        G1.add(jRadioButton2);
   }
}
class radioButton {
        public static void main(String args[])
        {
                Demo f = new Demo();
                f.setBounds(100, 100, 400, 200);
                f.setTitle("RadioButtons");
                f.setVisible(true);
        }
}
```



JCheckBox

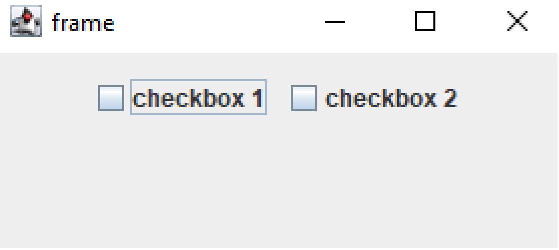
- ➢ JCheckBox can be selected or deselected.
- ➢ It displays it state to the user.
- ➢ JCheckBox inherits JToggleButton class.
- ➢ Constructor of the class are :

1. **JCheckBox()** : creates a new checkbox with no text or icon
2. **JCheckBox(Icon i)** : creates a new checkbox with the icon specified

3. **JCheckBox(Icon icon, boolean s)** : creates a new checkbox with the icon specified and the boolean value specifies whether it is selected or not.

- Methods to add Item Listener to checkbox.
  1. **addActionListener(ItemListener l):** adds item listener to the component
  2. **itemStateChanged(ItemEvent e)** : abstract function invoked when the state of the item to which listener is applied changes
  3. **getSource()** : Returns the component that fired the item event.
  4. **setText(String s)** :sets the text of the checkbox to the given text
  5. **getText()** : returns the text of the checkbox

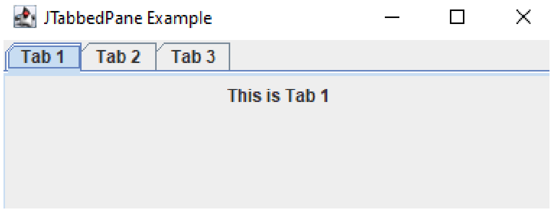
```
import java.awt.*;
import javax.swing.*;
class checkBox extends JFrame {
 static JFrame f ;
 public static void main(String[] args) {
          f = new JFrame("frame");
          f.setLayout(new FlowLayout());
          JCheckBox c1 = new JCheckBox("checkbox 1");
          JCheckBox c2 = new JCheckBox("checkbox 2");
          JPanel p = new JPanel();
          p.add(c1);
          p.add(c2);
          f.add(p);
          f.setSize(300, 300);
          f.show();
 } }
```



JTabbedPane

- A GUI (Graphical User Interface) component in the Java Swing library that allows you to create a tabbed pane interface.
- A tabbed pane is a container that can store and organize multiple components into distinct tabs.
- It contains a collection of tabs.
- When you click on a tab, only data related to that tab will be displayed.
- Constructor
  1. **JTabbedPane ():** The JTabbedPane class in Java Swing has a default, no-argument constructor called JTabbedPane(). This constructor initializes an empty tabbed pane with no tabs and no initial content when a JTabbedPane is created.
- Methods
  1. **addTab(String title, Component component)** ; Creates a new tab with the given title and content.
  2. **removeTabAt(int index):** Removes the tab at the given index.
  3. **getTabCount():**Returns the number of tabs present in the JTabbedPane.
  4. **setSelectedIndex(int index):** Sets the chosen tab to the index given.
  5. **getSelectedIndex():**Returns the index of the currently selected tab.

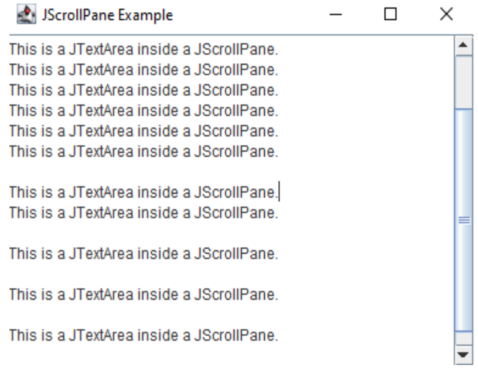
```
import javax.swing.*;
import java.awt.*;
public class tabbedPanes {
  public static void main(String[] args) {
    JFrame frame = new JFrame("JTabbedPane Example");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 300);
    JTabbedPane tabbedPane = new JTabbedPane();
    JPanel panel1 = new JPanel();
    panel1.add(new JLabel("This is Tab 1"));
    JPanel panel2 = new JPanel();
    panel2.add(new JLabel("This is Tab 2"));
    JPanel panel3 = new JPanel();
    panel3.add(new JLabel("This is Tab 3"));
    tabbedPane.addTab("Tab 1", panel1);
    tabbedPane.addTab("Tab 2", panel2);
    tabbedPane.addTab("Tab 3", panel3);
    frame.add(tabbedPane);
    frame.setVisible(true);
  }}
```



JScrollPane

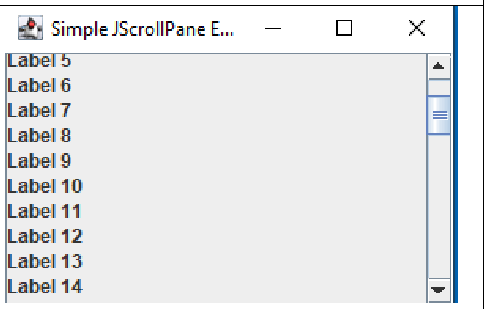
- A component in the Java Swing library that provides a scrollable view of another component, usually a JPanel or a JTextArea.
- It provides a scrolling functionality to the display for which the size changes dynamically.
- It is useful to display the content which exceeds the visible area of the window.
- Methods of JScrollPane
    1. **void setVerticalScrollBarPolicy(int vertical):** Sets the vertical scrollbar policy
    2. **void setHorizontalScrollBarPolicy(int horizontal):** Sets the horizontal scrollbar policy
    3. **void setColumnHeaderView(Component comp):** sets the column header for the JScrollPane
    4. **void setRowHeaderView(Component comp):** sets the rowheader for the JScrollPane
    5. **setCorner(String key, Component corner):** It is used to set a component to be displayed in one of the corners of the scroll pane
    6. **Component getCorner(String key):** It is used to retrieve the component that has been previously set in one of the corners of the scroll pane using the setCorner method
    7. **void setViewportView(Component comp):** It is used to set the component that will be displayed in the viewport of the scroll pane

```
import javax.swing.*;
import java.awt.*;
public class scrollPanes {
   public static void main(String[] args) {
JFrame frame = new JFrame("JScrollPane Example");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 frame.setSize(400, 300);
 JTextArea textArea = new JTextArea(10, 30);
 textArea.setText("This is a JTextArea inside a ScrollPane.\n".repeat(10));
  JScrollPane scrollPane = new JScrollPane(textArea);
scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
      frame.add(scrollPane, BorderLayout.CENTER);
      frame.setVisible(true);
   }
}
```



```
This is a JTextArea inside a JScrollPane.
This is a JTextArea inside a JScrollPane.
This is a JTextArea inside a JScrollPane.
This is a JTextArea inside a JScrollPane.
This is a JTextArea inside a JScrollPane.
This is a JTextArea inside a JScrollPane.

This is a JTextArea inside a JScrollPane.
This is a JTextArea inside a JScrollPane.

This is a JTextArea inside a JScrollPane.

This is a JTextArea inside a JScrollPane.

This is a JTextArea inside a JScrollPane.
```

```
import javax.swing.*;
public class scrollPanes_ {
 public static void main(String[] args) {
    JFrame frame = new JFrame("Simple JScrollPane Example");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(300, 200);

    JPanel panel = new JPanel();
    panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

    for (int i = 1; i <= 50; i++) {

      JLabel label = new JLabel("Label " + i);
      panel.add(label);

    }

    JScrollPane scrollPane = new JScrollPane(panel);
    frame.add(scrollPane);
    frame.setVisible(true);
 }
}
```

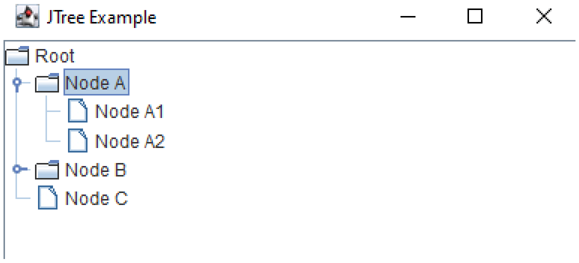


```
Label 5
Label 6
Label 7
Label 8
Label 9
Label 10
Label 11
Label 12
Label 13
Label 14
```

JTree

- The JTree is a type of GUI (Graphic User Interface) that displays information in a hierarchical way.
- Representing relationships among elements in a tree-like structure.

```
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;

public class tREE {
   public static void main(String[] args) {
 JFrame frame = new JFrame("JTree Example");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setSize(400, 300);

  DefaultMutableTreeNode root = new
DefaultMutableTreeNode("Root");

       // Create Child Nodes
  DefaultMutableTreeNode nodeA = new
DefaultMutableTreeNode("Node A");
  DefaultMutableTreeNode nodeB = new
DefaultMutableTreeNode("Node B");
  DefaultMutableTreeNode nodeC = new
DefaultMutableTreeNode("Node C");
       // Create Sub-Nodes
  DefaultMutableTreeNode nodeA1 = new
DefaultMutableTreeNode("Node A1");
  DefaultMutableTreeNode nodeA2 = new
DefaultMutableTreeNode("Node A2");

  DefaultMutableTreeNode nodeB1 = new
DefaultMutableTreeNode("Node B1");

      // Add Sub-Nodes to Parent Nodes
       nodeA.add(nodeA1);
       nodeA.add(nodeA2);
       nodeB.add(nodeB1);

       // Add Child Nodes to Root
       root.add(nodeA);
       root.add(nodeB);
       root.add(nodeC);

       // Create a JTree with the root node
       JTree tree = new JTree(root);

       // Add JTree to a ScrollPane
       JScrollPane scrollPane = new JScrollPane(tree);
       frame.add(scrollPane);
       frame.setVisible(true);
   }
}
```

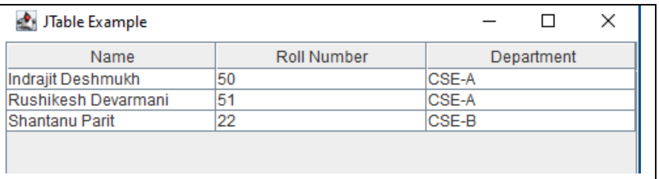


JTable

- The JTable class is a part of Java Swing Package and is generally used to display or edit two-dimensional data that is having both rows and columns.
- It is similar to a spreadsheet. This arranges data in a tabular form.
- methods in JTable:
  1. **addColumn(TableColumn []column) :** adds a column at the end of the JTable.
  2. **clearSelection() :** Selects all the selected rows and columns.

3. **editCellAt(int row, int col) :** edits the intersecting cell of the column number col and row number row programmatically, if the given indices are valid and the corresponding cell is editable.
4. **setValueAt(Object value, int row, int col) :** Sets the cell value as 'value' for the position row, col in the JTable.

```
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
public class Tables {
        JFrame f;
        JTable j;
        Tables()
{
        f = new JFrame();
        f.setTitle("JTable Example");
        String[][] data = {
                { "Indrajit Deshmukh", "50", "CSE-
A" },
                { "Rushikesh Devarmani", "51",
"CSE-A" },
                { "Shantanu Parit", "22", "CSE-B" }
                      };
        String[] columnNames = { "Name", "Roll
Number", "Department" };
        j = new JTable(data, columnNames);
        j.setBounds(30, 40, 200, 300);
        JScrollPane sp = new JScrollPane(j);
        f.add(sp);
        f.setSize(500, 200);
        f.setVisible(true);
        }
        public static void main(String[] args)
        {
                new Tables();
        }
}
```



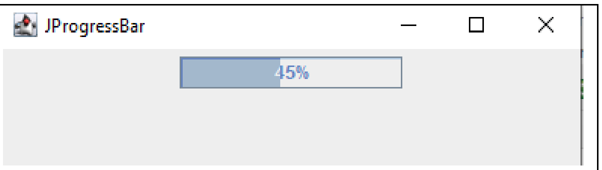
JTable Example

Name	Roll Number	Department
Indrajit Deshmukh	50	CSE-A
Rushikesh Devarmani	51	CSE-A
Shantanu Parit	22	CSE-B

JProgressBar

- JProgressBar is a part of Java Swing package.
- JProgressBar visually displays the progress of some specified task.
- JProgressBar shows the percentage of completion of specified task.
- The progress bar fills up as the task reaches it completion.
- In addition to show the percentage of completion of task, it can also display some text.
- methods of JProgressBar are :
  1. **int getMaximum()** : returns the progress bar's maximum value.
  2. **int getMinimum()** : returns the progress bar's minimum value.
  3. **String getString()** : get the progress bar's string representation of current value.
  4. **void setMaximum(int n)** : sets the progress bar's maximum value to the value n.
  5. **void setMinimum(int n)** : sets the progress bar's minimum value to the value n.
  6. **void setValue(int n)** : set Progress bar's current value to the value n.
  7. **void setString(String s)** : set the value of the progress String to the String s.

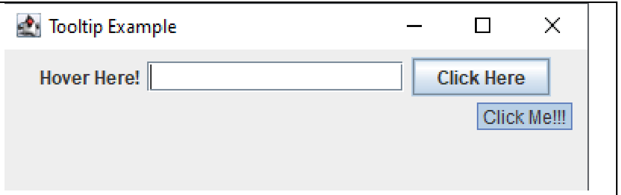
```
import javax.swing.*;
import java.awt.*;
public class ProgressBAR {
   public static void main(String[] args) {
      // Create a JFrame
      JFrame frame = new JFrame("JProgressBar");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(400, 200);
      frame.setLayout(new FlowLayout());
     // Create a JProgressBar
     JProgressBar progressBar = new JProgressBar(0, 100);
      progressBar.setStringPainted(true); // Show percentage
     // Add ProgressBar to Frame
      frame.add(progressBar);
      frame.setVisible(true);
     // Simulate Progress Update
      for (int i = 0; i <= 100; i++) {
         try {
            Thread.sleep(50); // Simulating time delay
         } catch (InterruptedException e) {
           System.out.println(e);          }
         progressBar.setValue(i);
      }
   }
}
```



ToolTip

- When the cursor enters the boundary of that component a popup appears and text is displayed.
- We can add tooltip text to almost all the components of Java Swing
  1. **setToolTipText(String s):** method sets the tooltip of the component to the specified string s.
  2. **getToolTipText()** : returns the tooltip text for that component .
  3. **getToolTipText(MouseEvent e):** returns the same value returned by getToolTipText(). Multi-part components such as JTabbedPane, JTable, and JTree override this method to return a string associated with the mouse event location.
  4. **getToolTipLocation(MouseEvent e) :** Returns the location (in the receiving component's coordinate system) where the upper left corner of the component's tool tip appears.

```
import javax.swing.*;
import java.awt.*;
public class toolTip {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Tooltip Example");

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(400, 200);
      frame.setLayout(new FlowLayout());
      // Create a Label with Tooltip
      JLabel label = new JLabel("Hover Here!");
      label.setToolTipText("This is a label tooltip");
      // Create a Text Field with Tooltip
      JTextField textField = new JTextField(15);
      textField.setToolTipText("Enter your text here!!");
      // Create a Button with Tooltip
      JButton button = new JButton("Click Here");
      button.setToolTipText("Click Me!!!");
      frame.add(label);
      frame.add(textField);
      frame.add(button);
      frame.setVisible(true);
   }
}
```



Event Handling

The Delegation Event Model

- The delegation event model, which defines standard and consistent mechanisms to generate and process events.
- Its concept is quite simple: a **source** generates an event and sends it to one or more listeners.
- The **listener** simply waits until it receives an event.
- Once received, the listener processes the event and then returns.
- In the delegation event model, listeners must register with a source in order to receive an event notification
- An event is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a graphical user interface

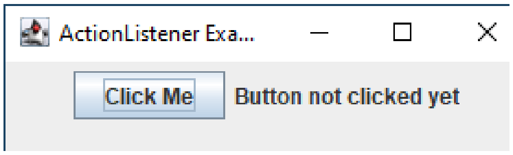
Event Classes

ActionEvent Class

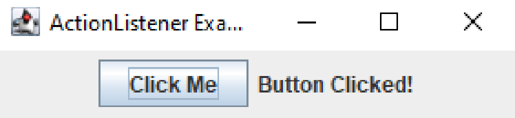
- This class is defined in java.awt.event package.
- An event that indicates that a component-defined action occurred like a button click or selecting an item from the menu-item list or the item of a list is double-clicked.
- **ActionListener** interface is used.
- **actionPerformed()** method is used

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class actionListener {
   public static void main(String[] args) {
      JFrame frame = new JFrame("ActionListener Example");

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(300, 200);
      frame.setLayout(new FlowLayout());

      JButton button = new JButton("Click Me");
      JLabel label = new JLabel("Button not clicked yet");

      button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
            label.setText("Button Clicked!");
         }
      });
      frame.add(button);
      frame.add(label);
      frame.setVisible(true);
   }
}
```

Before Button click



After Button click

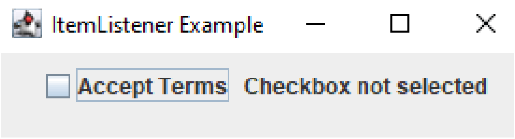


ItemEvent Class

- An event that indicates whether an item was selected or not.
- Used to handle **checkbox, radio button, and combo box selection events**.
- **ItemListener** interface is used.
- **itemStateChanged()** method is used

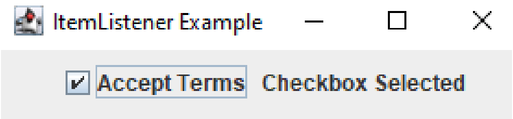
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
public class itemListener {
   public static void main(String[] args) {
      JFrame frame = new JFrame("ItemListener Example");
```

Before click on checkbox



```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(300, 200);
    frame.setLayout(new FlowLayout());
    JCheckBox checkBox = new JCheckBox("Accept Terms");
    JLabel label = new JLabel("Checkbox not selected");

    checkBox.addItemListener(new ItemListener() {
        public void itemStateChanged(ItemEvent e) {
        if (e.getStateChange() == ItemEvent.SELECTED) {
            label.setText("Checkbox Selected");
        } else {
            label.setText("Checkbox Deselected");
        }
      }
    });
    frame.add(checkBox);
    frame.add(label);
    frame.setVisible(true);
  }
}
```

After click on checkbox

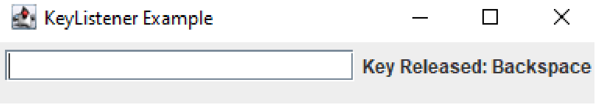


Checkbox Selected

KeyEvent Class

- An event that occurs due to a sequence of keypresses on the keyboard.
- Used to handle **keyboard events** (key press, release, and type) in Swing applications.
- KeyListener interface is used.
- keyTyped(), keyPressed(), keyReleased() methods are used.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
public class keyPressEvent {
  public static void main(String[] args) {
    JFrame frame = new JFrame("KeyListener Example");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 200);
    frame.setLayout(new FlowLayout());

     JTextField textField = new JTextField(20);
    JLabel label = new JLabel("Press a key...");

    textField.addKeyListener(new KeyListener()
    {
      public void keyTyped(KeyEvent e)
      {
       label.setText("Key Typed: " + e.getKeyChar());
      }
      public void keyPressed(KeyEvent e)
      {
        label.setText("Key Pressed: " +
KeyEvent.getKeyText(e.getKeyCode()));
      }
      public void keyReleased(KeyEvent e) {
        label.setText("Key Released: " +
KeyEvent.getKeyText(e.getKeyCode()));   }
    });
    frame.add(textField);
    frame.add(label);
    frame.setVisible(true);
  }
}
```

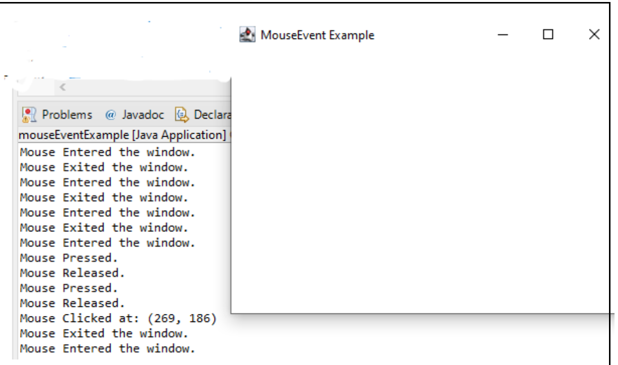


Key Released: Backspace

mouseEvent

- the MouseListener interface in AWT is an important tool for managing mouse events in Java applications.
- The **MouseListener** interface is part of the *'java.awt.event'* package.
- It is used to retrieve and respond to mouse-related events in Java applications.
- Mouse clicks, mouse button presses and releases, mouse enter and exit events are examples of these events.
- By implementing the **MouseListener** interface, you can define specific actions that your application has to perform in response to the interactions with the mouse.
- **MouseListener** interface is used.
    - **void mouseClicked(MouseEvent e):** Triggered when the mouse button is clicked (pressed and released) on a component.
    - **void mousePressed(MouseEvent e):** Triggered when a mouse button is pressed on a component.
    - **void mouseReleased(MouseEvent e):** This method is called when a mouse button is released on a component.
    - **void mouseEntered(MouseEvent e):** Invoked after mouse entry into a component.
    - **void mouseExited(MouseEvent e):** Invoked when the mouse exits a component.

```
import java.awt.*;
import java.awt.event.*;
public class mouseEventExample extends Frame {
public mouseEventExample() {
        setTitle("MouseEvent Example");
        setSize(400, 300);
        addMouseListener(new MouseAdapter() {
       public void mouseClicked(MouseEvent e) {
          System.out.println("Mouse Clicked at: (" +
e.getX() + ", " + e.getY() + ")");
       }
       public void mouseEntered(MouseEvent e) {
          System.out.println("Mouse Entered the
window.");
       }
       public void mouseExited(MouseEvent e) {
          System.out.println("Mouse Exited the
window.");
       }
       public void mousePressed(MouseEvent e) {
          System.out.println("Mouse Pressed.");
       }
       public void mouseReleased(MouseEvent e) {
          System.out.println("Mouse Released.");
       }
    });
    setVisible(true);
    addWindowListener(new WindowAdapter() {
       public void windowClosing(WindowEvent we) {
          System.exit(0);
       }
    });
  }
  public static void main(String[] args) {
    new mouseEventExample();
  }
}
```



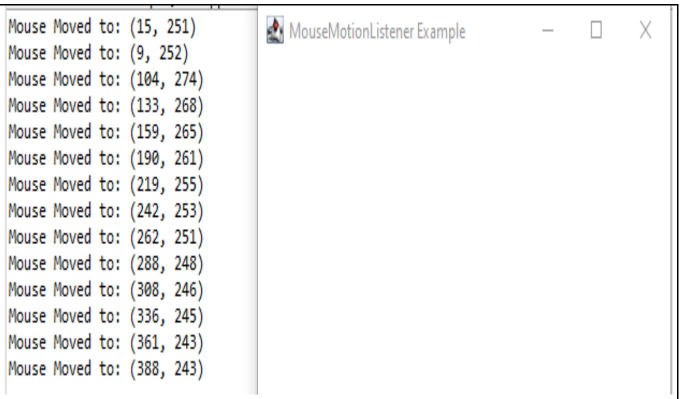
```
Mouse Entered the window.
Mouse Exited the window.
Mouse Entered the window.
Mouse Exited the window.
Mouse Entered the window.
Mouse Exited the window.
Mouse Entered the window.
Mouse Pressed.
Mouse Released.
Mouse Pressed.
Mouse Released.
Mouse Clicked at: (269, 186)
Mouse Exited the window.
Mouse Entered the window.
```

mouseMotion

- MouseMotionListener handles the events when mouse is in motion.
- To track sneak out motion events.

- This hearer allows you to respond to the mouse's front, such as sleuthing when the mouse is dragged or moved within a component.
- In this clause, we wish research the MouseMotionListener.
- MouseMotionListener interface is used.
    - public void mouseDragged(MouseEvent e): This method is called when the mouse is dragged with a button down. It provides information about the mouse event, such as the mouse's location and the component it occurred on.
    - public void mouseMoved(MouseEvent e): This method is called when the mouse is moved without any buttons being pressed. It also provides information about the mouse event.

```
import java.awt.*;
import java.awt.event.*;
public class MouseMotionEventExample extends Frame {
  public MouseMotionEventExample() {
    setTitle("MouseMotionListener Example");
    setSize(400, 300);
    addMouseMotionListener(new MouseMotionAdapter() {
      public void mouseMoved(MouseEvent e) {
        // Display the current mouse position in the window
        System.out.println("Mouse Moved to: (" + e.getX() + ", " + e.getY() + ")");
      }

      public void mouseDragged(MouseEvent e) {
        // Display the current mouse position while dragging
        System.out.println("Mouse Dragged to: (" + e.getX() + ", " + e.getY() + ")");
      }
    });

    setVisible(true);
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent we) {
        System.exit(0);
      }
    });
  }
  public static void main(String[] args) {
    new MouseMotionEventExample();
  }
}
```

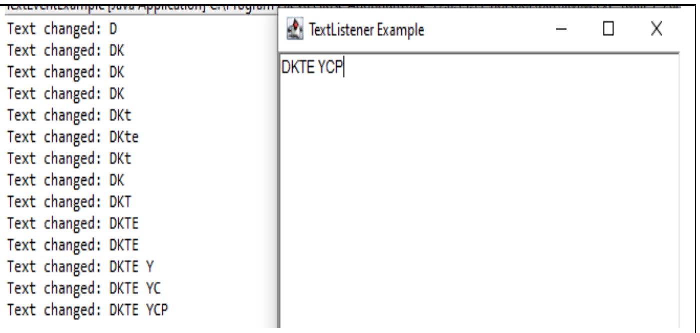


```
Mouse Moved to: (15, 251)
Mouse Moved to: (9, 252)
Mouse Moved to: (104, 274)
Mouse Moved to: (133, 268)
Mouse Moved to: (159, 265)
Mouse Moved to: (190, 261)
Mouse Moved to: (219, 255)
Mouse Moved to: (242, 253)
Mouse Moved to: (262, 251)
Mouse Moved to: (288, 248)
Mouse Moved to: (308, 246)
Mouse Moved to: (336, 245)
Mouse Moved to: (361, 243)
Mouse Moved to: (388, 243)
```

textEvent

- An event that occurs when an object's text changes.
- Ie. The TextEvent is generated when character is entered in the text fields or text area.
- The TextEvent instance does not include the characters currently in the text component that generated the event rather we are provided with other methods to retrieve that information.
- TextListener interface is used.
    - textValueChanged(TextEvent e). : This method is triggered whenever the content of a TextField changes, such as when the user types or deletes text.

```
import java.awt.*;
import java.awt.event.*;
public class TextEventExample extends Frame {
   public TextEventExample() {
      setTitle("TextListener Example");
      setSize(400, 300);
      TextField textField = new TextField();
      add(textField, BorderLayout.CENTER);

      // Add a TextListener to the TextField
      textField.addTextListener(new TextListener()
{
         public void textValueChanged(TextEvent
e) {
            System.out.println("Text changed: " +
textField.getText());
         }
      });
      setVisible(true);
  addWindowListener(new WindowAdapter() {
   public void windowClosing(WindowEvent we)
{       System.exit(0);          }       });
   }
   public static void main(String[] args) {
      new TextEventExample();
   }
}
```



```
TextEventExample [Java Application] C:\Program
Text changed: D
Text changed: DK
Text changed: DK
Text changed: DK
Text changed: DKt
Text changed: DKte
Text changed: DKt
Text changed: DK
Text changed: DKT
Text changed: DKTE
Text changed: DKTE
Text changed: DKTE Y
Text changed: DKTE YC
Text changed: DKTE YCP
```