**Unit-10 (Java Application)**

**Concept of AWT (Abstract Window Toolkit):**

Java **AWT** is an API that contains large number of **classes** and **methods** to create and manage graphical user interface ( **GUI** ) applications. The AWT was designed to provide a common set of tools for GUI design that could work on a variety of platforms. The tools provided by the AWT are implemented using each platform's native GUI toolkit, hence preserving the look and feel of each platform. This is an advantage of using AWT. But the disadvantage of such an approach is that GUI designed on one platform may look different when displayed on another platform that means AWT component are platform dependent.

AWT is the foundation upon which Swing is made i.e Swing is a improved GUI API that extends the AWT. But now a days AWT is merely used because most GUI Java programs are implemented using Swing because of its rich implementation of GUI controls and light-weighted nature.

**Swing:**

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

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

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

**What is JFC**

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

**Java AWT Vs Swing**

| AWT | Swing |
| --- | --- |
| AWT stands for “Abstract Windows Toolkit”. | Swing is derived from Java Foundation Classes (JFC). |
| AWT components are heavyweight | Swing components are written on top of AWT and as such the components are light-weight. |
| AWT components are part of the **java.awt** package. | Swing components are part of **javax.swing** package. |
| AWT is platform - dependent. | Swing components are written in Java and are platform-independent. |
| AWT doesn’t have its look and feel. It adapts the look and feel of the platform on which it runs. | Swing provides a different look and feel of its own. |
| AWT only has basic features and does not support advanced features like the table, tabbed panel, etc. | Swing provides advanced features like JTabbed panel, JTable, etc. |
| AWT is as good as a thin layer of classes sitting on top of the Operating system which makes it platform-dependent. | Swing is larger and also contains rich functionality. |
| AWT makes us write a lot of things. | Swing has most of the features built-in. |

**Hierarchy of Java Swing classes**

The hierarchy of java swing API is given below.



**About JFrame:**

Whenever you create a **graphical user interface** with Java Swing functionality, you will need a **container** for your application. In the case of Swing, this container is called a **JFrame**. All GUI applications require a **JFrame**. In fact some Applets even use a JFrame.

You can't build a house without a foundation. The same is true in Java: Without a container in which to put all other elements, you won't have a GUI application. In other words, the JFrame is required as the foundation or base container for all other graphical components.

 A frame is actually a **base window** on which other **components rely**, **namely the menu bar, panels, labels, text fields, buttons, etc**. Almost every other Swing application starts with the **JFrame** window.

## **How to create a JFrame?**

JFrame class has many constructors that are used to create a new JFrame. You can create a JFrame using these methods:

**JFrame():** This helps in creating a frame which is invisible.  
**JFrame(String Title):** Helps in creating a frame with a title.  
**JFrame(GraphicsConfiguration gc):** Creates a frame with blank title and the graphics configuration of screen.

**Java Swing Examples**

There are two ways to create a frame:

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

We can write the code of swing inside the main(), constructor or any other method.

* By creating the object of Frame class (association)

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

**Simple Java Swing Example**

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

import javax.swing.\*;

public class App {

    public static void main(String[] args) throws Exception {

        JFrame f=new JFrame();//creating instance of JFrame

        JButton b=new JButton("click");//creating instance of JButton

        b.setBounds(130,100,100, 40);//x axis, y axis, width, height

        f.add(b);//adding button in JFrame

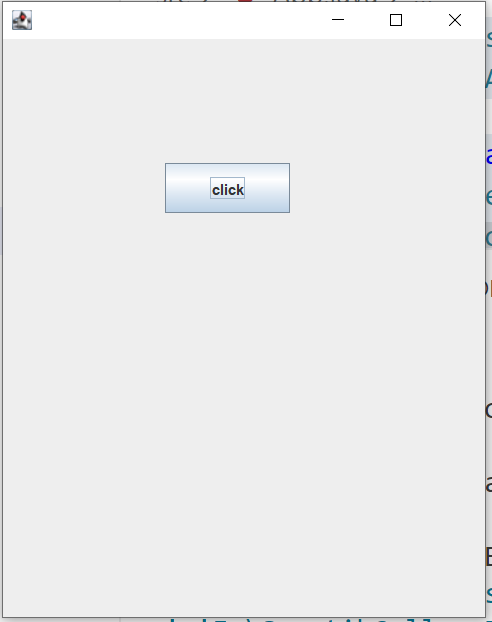
        f.setSize(400,500);//400 width and 500 height

        f.setLayout(null);//using no layout managers

        f.setVisible(true);//making the frame visible

    }

}



**Example of Swing by Association inside constructor**

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

import javax.swing.\*;

public class App {

    App()

    {

        JFrame f=new JFrame();//creating instance of JFrame

        JButton b=new JButton("click");//creating instance of JButton

        b.setBounds(130,100,100, 40);//x axis, y axis, width, height

        f.add(b);//adding button in JFrame

        f.setSize(400,500);//400 width and 500 height

        f.setLayout(null);//using no layout managers

        f.setVisible(true);//making the frame visible

    }

    public static void main(String[] args) throws Exception {

        new App();

    }

}

**2.Simple example of Swing by inheritance**

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

import javax.swing.\*;

public class App extends JFrame {

    App()

    {

        JButton b=new JButton("click");//create button

        b.setBounds(130,100,100, 40);

        add(b);//adding button on frame

        setSize(400,500);

        setLayout(null);

        setVisible(true);

    }

    public static void main(String[] args) throws Exception {

       new App();

    }

}

**GUI Elements:**

1. JLabel
2. JRadioButton
3. ButtonGroup
4. JCheckBox
5. JTextField
6. JTextArea
7. JButton
8. Border
9. JComboBox
10. JTabbedPane
11. JPasswordField
12. Look and Feel Management in Java Swing

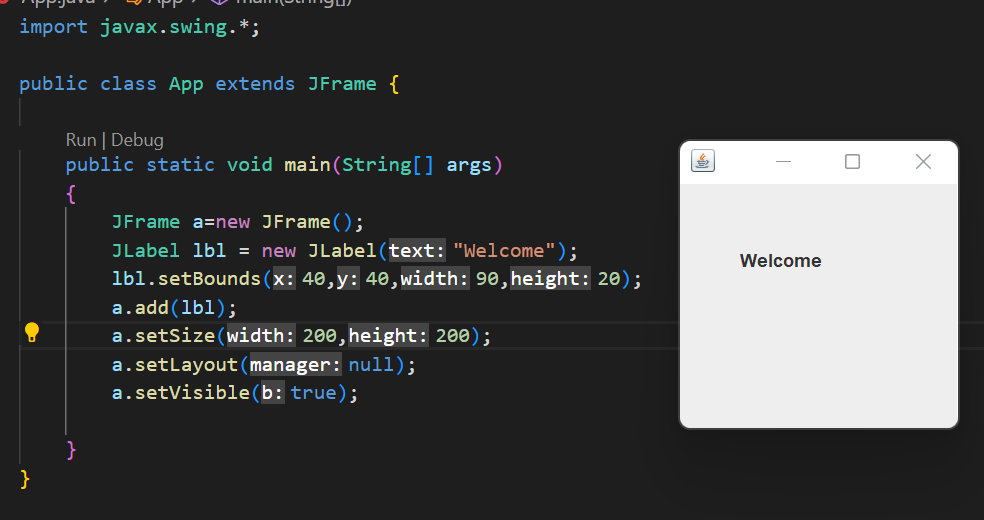
**1.JLabel**

The object of the **JLabel** class may be a component for putting text in a container. It’s used to display one line of read-only text. The text is often changed by an application but a user cannot edit it directly. It inherits the **JComponent** class.

Syntax:   
**JLabel jl = new JLabel();**

**JLabel Constructors**

* **JLabel():**It is used to create a JLabel instance with no image and with an empty string for the title.
* **JLabel(String s):**It is used to create a JLabel instance with the specified text.
* **JLabel(Icon i):**It is used to create a JLabel instance with the specified image.
* **JLabel(String s, Icon I, int horizontalAlignment):**It is used to create a JLabel instance with the specified text, image, and horizontal alignment.

Example:

##### **2.JRadioButton**

This component allows the user to select only one item from a group item. By using the JRadioButton component you can choose one option from multiple options.

Syntax**: JRadioButton jrb = new JRadioButton();**

##### **JRadioButton Constructors**

* **JRadioButton():**It is used to create an unselected radio button with no text.
* **JRadioButton(Label):**It is used to create an unselected radio button with specified text.
* **JRadioButton(Label, boolean):**It is used to create a radio button with the specified text and selected status.

Code:

import javax.swing.\*;

public class App {

    public static void main(String[] args)

    {

        JFrame f=new JFrame();

        JRadioButton r1=new JRadioButton("A) Male",true);

        JRadioButton r2=new JRadioButton("B) Female");

        r1.setBounds(75,50,100,30);

        r2.setBounds(75,100,100,30);

        ButtonGroup bg=new ButtonGroup();

        bg.add(r1);

        bg.add(r2);

        f.add(r1);

        f.add(r2);

        f.setSize(300,300);

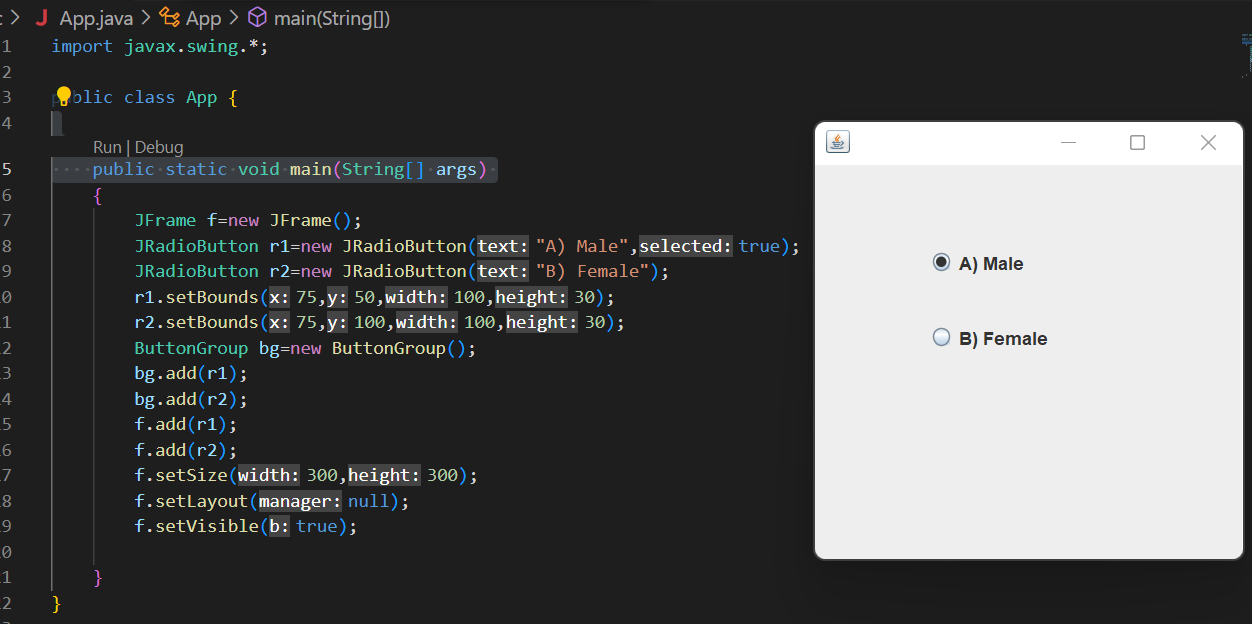
        f.setLayout(null);

        f.setVisible(true);

    }

}

Output:



**3.JPasswordField**

The object of a JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text. It inherits JTextField class.

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JPasswordField() | Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width. |
| JPasswordField(int columns) | Constructs a new empty JPasswordField with the specified number of columns. |
| JPasswordField(String text) | Constructs a new JPasswordField initialized with the specified text. |
| JPasswordField(String text, int columns) | Construct a new JPasswordField initialized with the specified text and columns. |

Code:

import javax.swing.\*;

public class App {

    public static void main(String[] args)

    {

        JFrame f=new JFrame();

        JPasswordField value = new JPasswordField();

        JLabel l1=new JLabel("Password:");

        l1.setBounds(20,100, 80,30);

        value.setBounds(100,100,100,30);

        f.add(value);  f.add(l1);

        f.setSize(300,300);

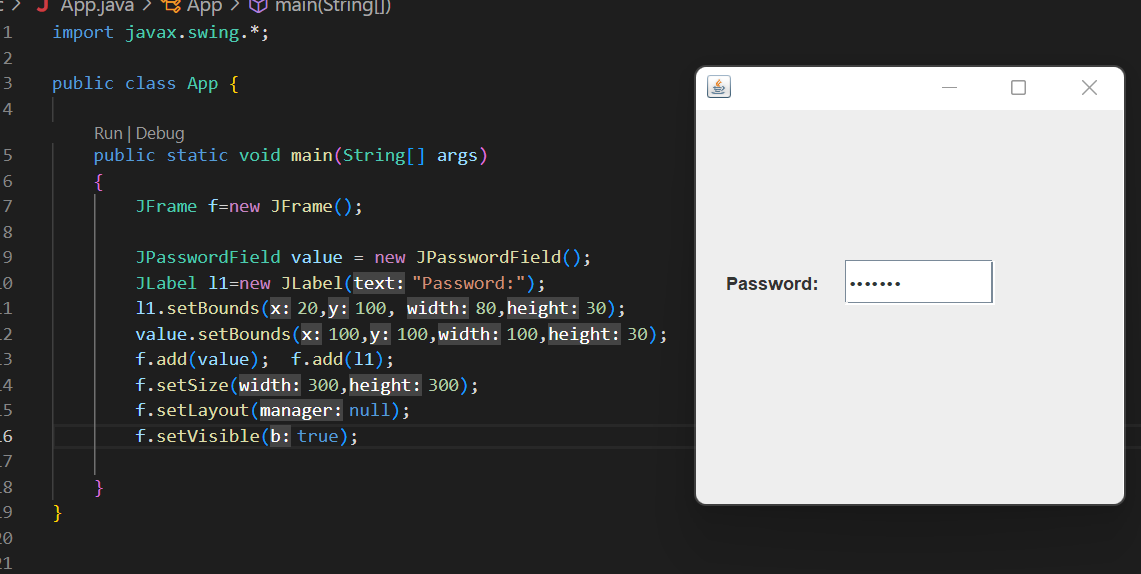
        f.setLayout(null);

        f.setVisible(true);

    }

}

Output:



**4.JCheckBox**

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits [**JToggleButton**](https://www.javatpoint.com/java-jtogglebutton) class.

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JJCheckBox() | Creates an initially unselected check box button with no text, no icon. |
| JChechBox(String s) | Creates an initially unselected check box with text. |
| JCheckBox(String text, boolean selected) | Creates a check box with text and specifies whether or not it is initially selected. |
| JCheckBox(Action a) | Creates a check box where properties are taken from the Action supplied. |

Code:

import javax.swing.\*;

public class App {

    public static void main(String[] args)

    {

        JFrame f=new JFrame();

        JCheckBox checkBox1 = new JCheckBox("C++");

        checkBox1.setBounds(100,100, 50,50);

        JCheckBox checkBox2 = new JCheckBox("Java", true);

        checkBox2.setBounds(100,150, 100,50);

        f.add(checkBox1);

        f.add(checkBox2);

        f.setSize(200,200);

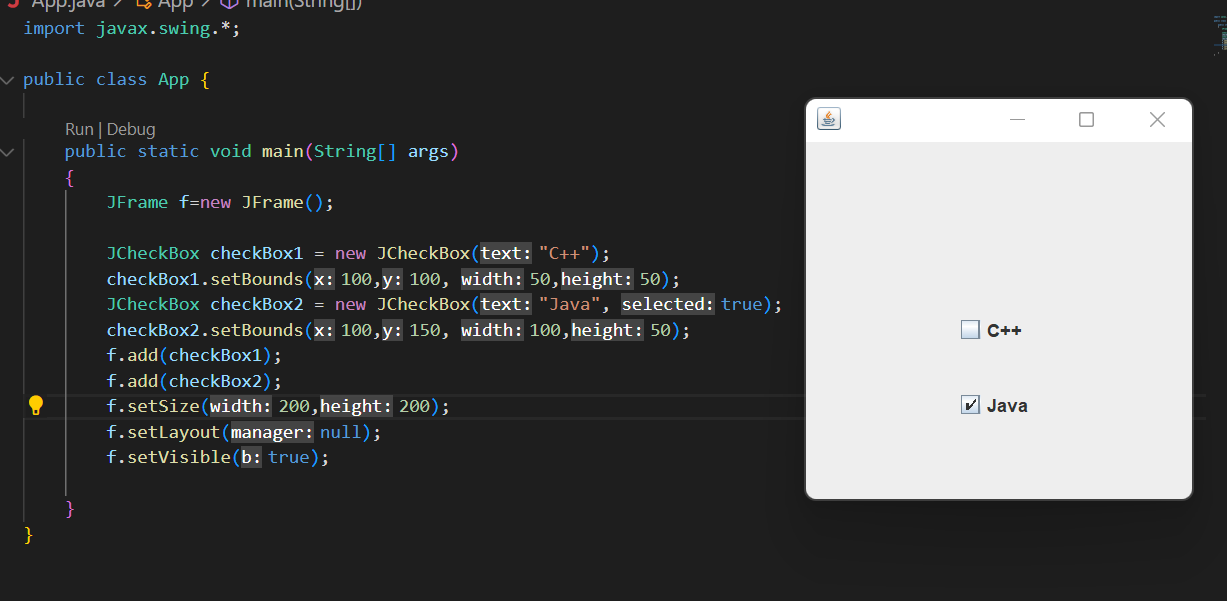
        f.setLayout(null);

        f.setVisible(true);

    }

}

Output:



**5. JComboBox**

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

import javax.swing.\*;

public class App {

    public static void main(String[] args)

    {

        JFrame f=new JFrame();

        String courses[] = { "core java","advance java", "java servlet"};

        JComboBox c = new JComboBox(courses);

        c.setBounds(40,40,90,20);

        f.add(c);

        f.setSize(400,400);

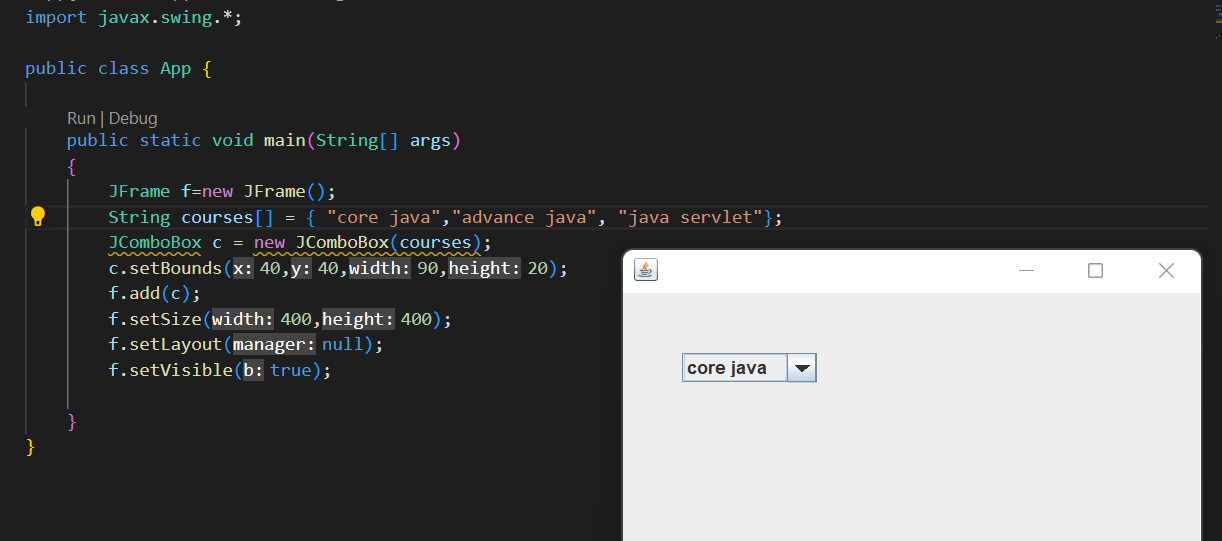
        f.setLayout(null);

        f.setVisible(true);

    }

}

Output:



**6. JTextArea**

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

**Commonly used Constructors:**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JTextArea() | Creates a text area that displays no text initially. |
| JTextArea(String s) | Creates a text area that displays specified text initially. |
| JTextArea(int row, int column) | Creates a text area with the specified number of rows and columns that displays no text initially. |
| JTextArea(String s, int row, int column) | Creates a text area with the specified number of rows and columns that displays specified text. |

Code:

import javax.swing.\*;

public class App {

    public static void main(String[] args)

    {

        JFrame f=new JFrame();

        JTextArea area=new JTextArea("Welcome to javatpoint");

        area.setBounds(10,30, 200,200);

        f.add(area);

        f.setSize(300,300);

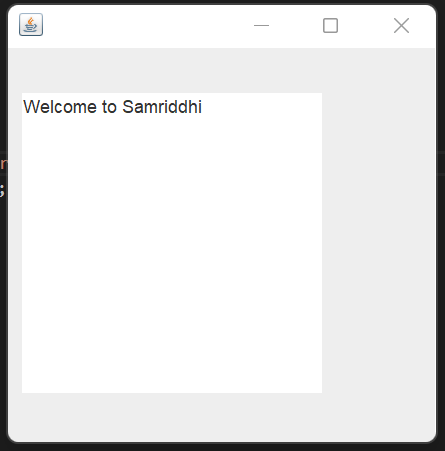
        f.setLayout(null);

        f.setVisible(true);

    }

}

Output



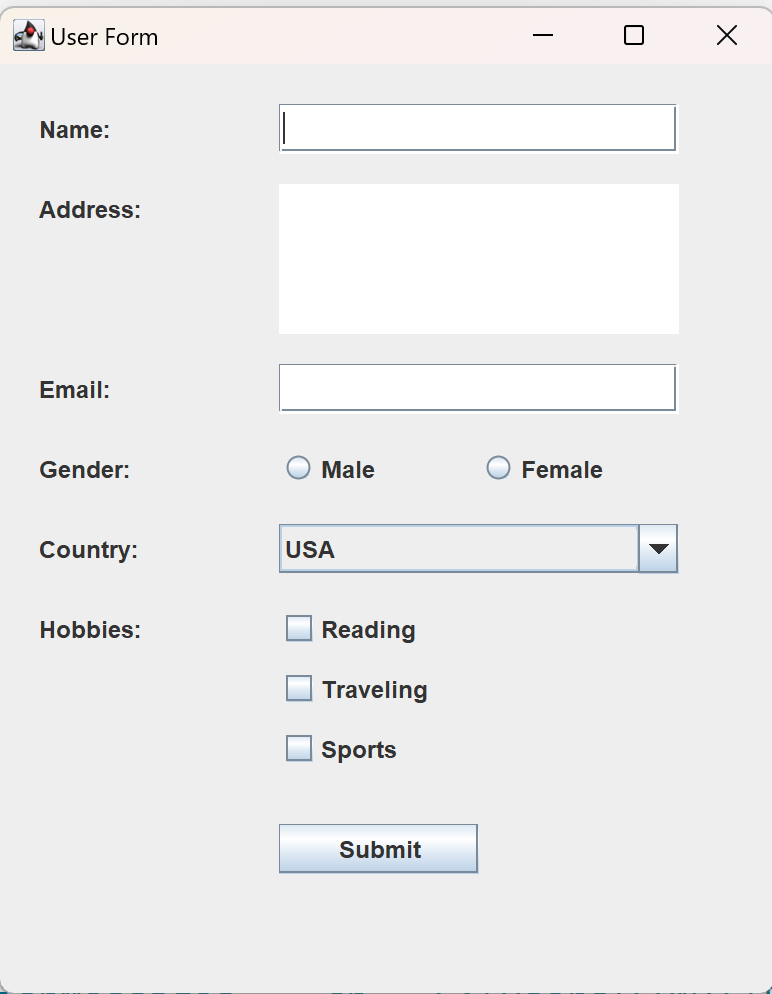
Write a java program to create a GUI form as shown below

**Code:**

import javax.swing.\*;

public class App {

    public static void main(String[] args) throws Exception {

    JFrame f=new JFrame();

    JTextField nameField;

    JTextArea addressArea;

    JTextField emailField;

    JRadioButton maleButton;

    JRadioButton femaleButton;

    JComboBox<String> countryComboBox;

    JCheckBox hobbyReading;

    JCheckBox hobbyTraveling;

    JCheckBox hobbySports;

    JButton submitButton;

    f.setTitle("User Form");

        // Create the components

        JLabel nameLabel = new JLabel("Name:");

        nameLabel.setBounds(20, 20, 100, 25);

        nameField = new JTextField();

        nameField.setBounds(140, 20, 200, 25);

        f.add(nameLabel);

        f.add(nameField);

JLabel addressLabel = new JLabel("Address:");

        addressLabel.setBounds(20, 60, 100, 25);

        addressArea = new JTextArea();

        addressArea.setBounds(140, 60, 200, 75);

        f.add(addressLabel);

        f.add(addressArea);

        JLabel emailLabel = new JLabel("Email:");

        emailLabel.setBounds(20, 150, 100, 25);

        emailField = new JTextField();

        emailField.setBounds(140, 150, 200, 25);

        f.add(emailLabel);

        f.add(emailField);

        JLabel genderLabel = new JLabel("Gender:");

        genderLabel.setBounds(20, 190, 100, 25);

        maleButton = new JRadioButton("Male");

        maleButton.setBounds(140, 190, 100, 25);

        femaleButton = new JRadioButton("Female");

        femaleButton.setBounds(240, 190, 100, 25);

        ButtonGroup genderGroup = new ButtonGroup();

        genderGroup.add(maleButton);

        genderGroup.add(femaleButton);

        f.add(genderLabel);

        f.add(maleButton);

        f.add(femaleButton);

        JLabel countryLabel = new JLabel("Country:");

        countryLabel.setBounds(20, 230, 100, 25);

        String[] countries = {"USA", "Canada", "UK", "Australia", "Other"};

        countryComboBox = new JComboBox<>(countries);

        countryComboBox.setBounds(140, 230, 200, 25);

        f.add(countryLabel);

        f.add(countryComboBox);

        JLabel hobbiesLabel = new JLabel("Hobbies:");

        hobbiesLabel.setBounds(20, 270, 100, 25);

        hobbyReading = new JCheckBox("Reading");

        hobbyReading.setBounds(140, 270, 100, 25);

        hobbyTraveling = new JCheckBox("Traveling");

        hobbyTraveling.setBounds(140, 300, 100, 25);

        hobbySports = new JCheckBox("Sports");

        hobbySports.setBounds(140, 330, 100, 25);

        f.add(hobbiesLabel);

        f.add(hobbyReading);

        f.add(hobbyTraveling);

        f.add(hobbySports);

        submitButton = new JButton("Submit");

        submitButton.setBounds(140, 380, 100, 25);

        f.add(submitButton);

        f.setSize(400, 500);

        f.setLayout(null);

        f.setVisible(true);

    }

}

**Border Swing Control in Java**

The border is an interface using which we can apply a border to every component. To create the borders we have to use the methods available in **BorderFactory** class. We can apply the created border to any component by using the **SetBorder()** method.

Syntax: **Component.setBorder(Border);**

**Methods of Border**

* **Border createLineBorder(Color, int):**It is used to create a line border. Here, the Color object specifies the color of the line and int specifies the width in pixels of the line.
* **Border createEtchedBorder(int, Color, Color):**It is used to create an etched border. Here, Color arguments specify the highlight and shadow colors to be used. Here, int arguments allow the border methods to be specified as either EtchedBorder.RAISED or EtchedBorder.LOWERED. The methods without the int arguments create a lowered etched border.
* **Border createBevelBorder(int, Color, Color):**It is used to create a raised or lowered beveled border, specifying the colors to use. Here, the integer argument can be either BevelBorder.RAISED or BevelBorder.LOWERED. Here, Color specifies the highlight and shadow colors.
* **MatteBorder createMatteBorder(int, int, int, int, Icon):**It is used to create a matte border. Here, the integer arguments specify the number of pixels that the border occupies at the top, left, bottom, and right (in that order) of whatever component uses it. Here, the color argument specifies the color which with the border should fill its area. Here, the icon argument specifies the icon which with the border should tile its area.
* **TitledBorder createTitledBorder(Border, String, int, int, Font, Color):**Create a titled border. Here, the string argument specifies the title to be displayed. Here, the optional font and color arguments specify the font and color to be used for the title’s text. Here, the border argument specifies the border that should be displayed along with the title. Here, the integer arguments specify the number of pixels that the border occupies at the top, left, bottom, and right (in that order) of whatever component uses it.
* **CompoundBorder createCompoundBorder(Border, Border):**Combine two borders into one. Here, the first argument specifies the outer border; the second, the inner border.

**Code:**

import javax.swing.\*;

public class App {

    public static void main(String[] args)

    {

        JFrame f=new JFrame();

        JPanel j=new JPanel();

        JTextArea area=new JTextArea("Welcome to Samriddhi");

        area.setBorder(BorderFactory.createTitledBorder("Group Box"));

        area.setBounds(10,30, 200,200);

        f.add(j);

        f.add(area);

        f.setSize(300,300);

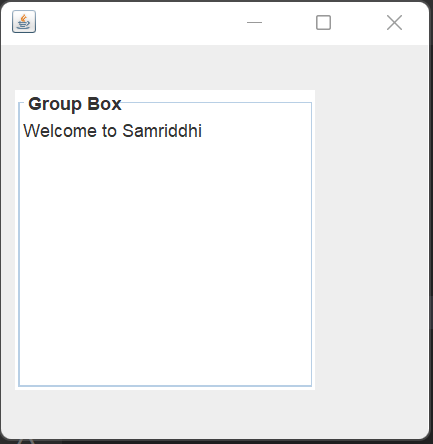
        f.setLayout(null);

        f.setVisible(true);

    }

}

Output:



**Layout Management:**

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

The Abstract Windowing Toolkit (AWT) has the following 3 layout managers:

* java.awt.BorderLayout
* java.awt.FlowLayout
* java.awt.GridLayout

**No Layout:**

import javax.swing.\*;

public class App extends JFrame{

    JFrame f;

    App()

    {

        JButton b=new JButton("click");//create button

        b.setBounds(130,100,100, 40);

        add(b);//adding button on frame

        setSize(400,500);

        setLayout(null);

        setVisible(true);

    }

    public static void main(String[] args)

    {

        App app=new App();

    }

}

**Flow Layout:**

The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the panel.

**Fields of FlowLayout class**

* public static final int LEFT(3)
* public static final int RIGHT(2)
* public static final int CENTER(1)
* public static final int LEADING
* public static final int TRAILING

**Example:**

import javax.swing.\*;

import java.awt.\*;

public class App{

    JFrame f;

    App()

    {

   f=new JFrame();

   JButton b1 = new JButton("1");

   JButton b2 = new JButton("2");

   JButton b3 = new JButton("3");

   JButton b4 = new JButton("4");

   JButton b5 = new JButton("5");

   JButton b6 = new JButton("6");

   JButton b7 = new JButton("7");

   JButton b8 = new JButton("8");

   JButton b9 = new JButton("9");

   JButton b10 = new JButton("10");

   f.add(b1); f.add(b2); f.add(b3); f.add(b4); f.add(b5); f.add(b6);  f.add(b7);  f.add(b8);

    f.add(b9);  f.add(b10);

    // parameter less constructor is used

    // therefore, alignment is center

    // horizontal as well as the vertical gap is 5 units.

    f.setLayout(new FlowLayout());

    f.setSize(300, 300);

    f.setVisible(true);

    }

    public static void main(String[] args)

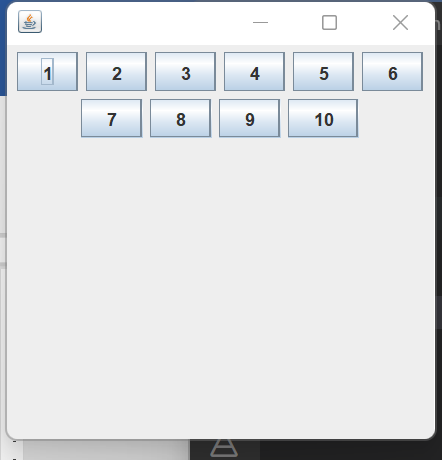
    {

        App app=new App();

    }

}

**Output:**

****

**Java BorderLayout**

The BorderLayout is used to arrange the components in five regions: north, south, east, west, and center. Each region (area) may contain one component only. It is the default layout of a frame or window. The BorderLayout provides five constants for each region:

1. public static final int NORTH
2. public static final int SOUTH
3. public static final int EAST
4. public static final int WEST
5. public static final int CENTER

**Constructors of BorderLayout class:**

* **BorderLayout():** creates a border layout but with no gaps between the components.
* **BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example:

import javax.swing.\*;

import java.awt.\*;

public class App{

    JFrame f;

    App()

    {

     f=new JFrame();

     // creating buttons

     JButton b1 = new JButton("NORTH");; // the button will be labeled as NORTH

     JButton b2 = new JButton("SOUTH");; // the button will be labeled as SOUTH

     JButton b3 = new JButton("EAST");; // the button will be labeled as EAST

     JButton b4 = new JButton("WEST");; // the button will be labeled as WEST

     JButton b5 = new JButton("CENTER");; // the button will be labeled as CENTER

     f.add(b1, BorderLayout.NORTH); // b1 will be placed in the North Direction

     f.add(b2, BorderLayout.SOUTH);  // b2 will be placed in the South Direction

     f.add(b3, BorderLayout.EAST);  // b2 will be placed in the East Direction

     f.add(b4, BorderLayout.WEST);  // b2 will be placed in the West Direction

     f.add(b5, BorderLayout.CENTER);  // b2 will be placed in the Center

     f.setSize(300, 300);

     f.setVisible(true);

    }

    public static void main(String[] args)

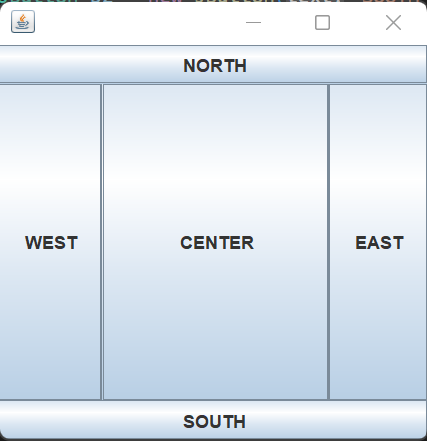
    {

        App app=new App();

    }

}

Output:



**Grid Layout:**

The Java GridLayout class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.

**Constructors of GridLayout class**

* **GridLayout():** creates a grid layout with one column per component in a row.
* **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
* **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

Example:

import javax.swing.\*;

import java.awt.\*;

public class App{

    JFrame f;

    App()

    {

     f=new JFrame();

     // creating 9 buttons

     JButton btn1 = new JButton("1");

     JButton btn2 = new JButton("2");

     JButton btn3 = new JButton("3");

     JButton btn4 = new JButton("4");

     JButton btn5 = new JButton("5");

     JButton btn6 = new JButton("6");

     JButton btn7 = new JButton("7");

     JButton btn8 = new JButton("8");

     JButton btn9 = new JButton("9");

     f.add(btn1); f.add(btn2); f.add(btn3);

f.add(btn4); f.add(btn5); f.add(btn6);

f.add(btn7); f.add(btn8); f.add(btn9);

// setting the grid layout using the parameterless constructor

f.setLayout(new GridLayout());

f.setSize(300, 300);

f.setVisible(true);

    }

    public static void main(String[] args)

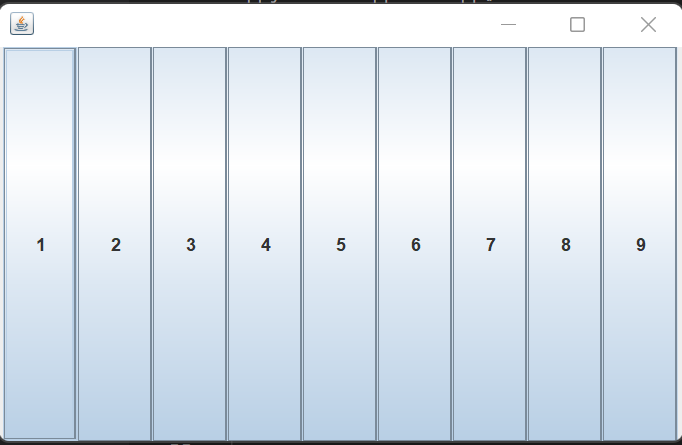
    {

        App app=new App();

    }

}

Output:



**These are some of the most used Event classes:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Event Class** | **Listener Interface** | **Methods** | **Descriptions** |
| ActionEvent | ActionListener | actionPerformed() | ActionEvent indicates that a component-defined action occurred. |
| FocusEvent | FocusListener | focusLost() and focusGained() | Focus events include focus, focusout, focusin, and blur. |
| ItemEvent | ItemListener | itemStateChanged() | Item event occurs when an item is selected. |
| KeyEvent | KeyListener | keyPressed(), keyReleased(), and keyTyped(). | A key event occurs when the user presses a key on the keyboard. |
| MouseEvent | MouseListener and MouseMotionListener | mouseClicked(), mousePressed(), mouseEntered(), mouseExited() and mouseReleased() are the mouseListener methods. mouseDregged() and mouseMoved() are the MouseMotionListener() methods. | A mouse event occurs when the user interacts with the mouse. |
| MouseWheelEvent | MouseWheelListener | mouseWheelMoved(). | MouseWheelEvent occurs when the mouse wheel rotates in a component. |
| TextEvent | TextListener | textChanged() | TextEvent occurs when an object's text change. |

**ActionListener:**

The Java ActionListener is notified whenever you click on the button or menu item. It is notified against **ActionEvent**. The **ActionListener** interface is found in **java.awt.event** package. It has only one method: **actionPerformed().**

**ActionListener Example:**

import javax.swing.\*;

import java.awt.event.\*;

//1st Step Implement ActionListner

public class App implements ActionListener{

   JTextField tf;

   JButton b;

   App()

   {

    JFrame f=new JFrame();

    tf=new JTextField();

    tf.setBounds(50,50, 150,20);

    b=new JButton("Click Here");

    b.setBounds(50,100,60,30);

    //2nd step

    b.addActionListener(this);

    f.add(b);

    f.add(tf);

    f.setSize(400,400);

    f.setLayout(null);

    f.setVisible(true);

   }

   //3rd Step

    public void actionPerformed(ActionEvent e)

    {

        tf.setText("Hello World");

    }

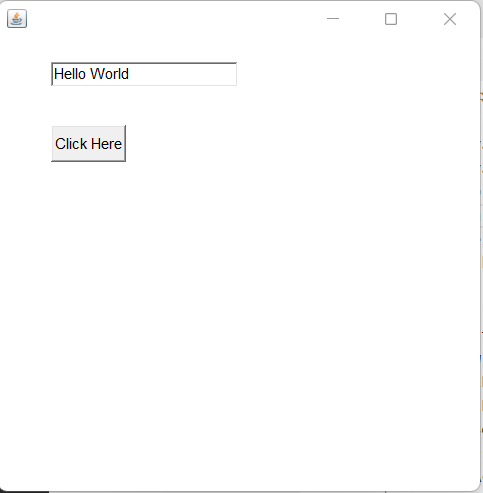
    public static void main(String[] args) throws Exception {

      new App();

    }

}

Output:



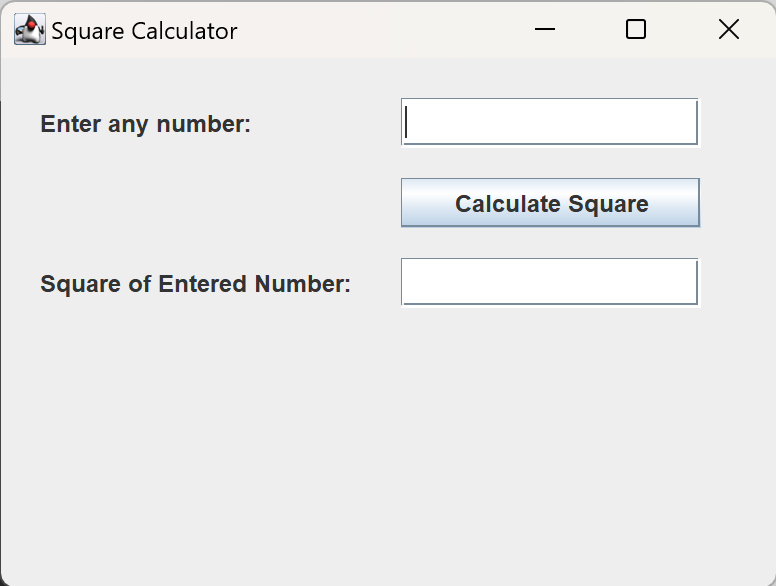
Lab: Write a java GUI Program to calculate square of entered number

**Code:**

import javax.swing.\*;

import java.awt.event.\*;

public class App implements ActionListener{

    JTextField inputField;

    JButton calculateButton;

    JTextField  outputField;

    JFrame f=new JFrame();

    public App()

    {

        f.setTitle("Square Calculator");

        // Create the components

       JLabel inputLabel = new JLabel("Enter any number:");

       inputLabel.setBounds(20, 20, 200, 25);

       inputField = new JTextField();

       inputField.setBounds(200, 20, 150, 25);

       f.add(inputLabel);

       f.add(inputField);

       calculateButton = new JButton("Calculate Square");

       calculateButton.setBounds(200, 60, 150, 25);

       f.add(calculateButton);

       JLabel outputLabel = new JLabel("Square of Entered Number:");

       outputLabel.setBounds(20, 100, 200, 25);

       outputField = new JTextField();

       outputField.setBounds(200, 100, 150, 25);

       calculateButton.addActionListener(this);

       f.add(outputField);

       f.add(outputLabel);

       f.setSize(400, 300);

       f.setLayout(null);

       f.setVisible(true);

    }

@Override

    public void actionPerformed(ActionEvent e) {

        int number = Integer.parseInt(inputField.getText());

        int square = number \* number;

        outputField.setText(String.valueOf(square));

    }

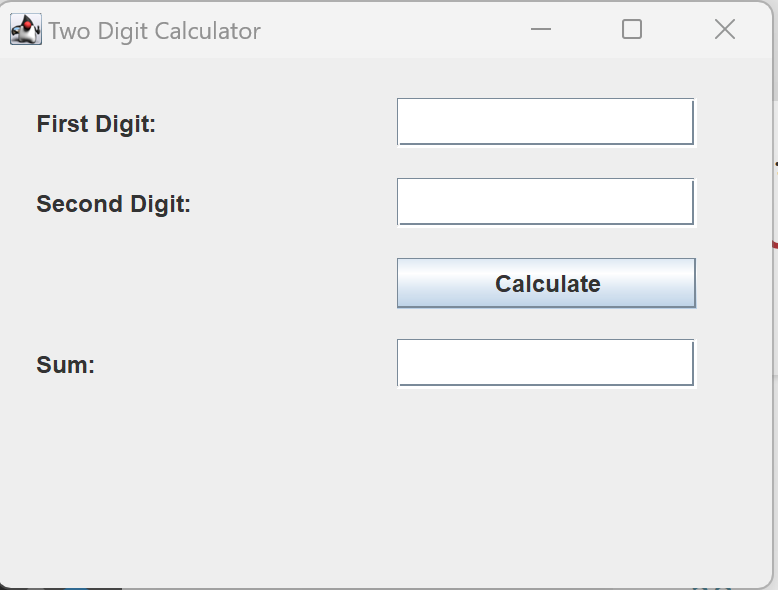
    public static void main(String[] args) throws Exception {

    new App();

    }

}

Lab: Write a java GUI Program to add two numbers from input fields and display sum in text another textfield after clicking button calculate



import javax.swing.\*;

import java.awt.event.\*;

public class App implements ActionListener{

    JTextField firstDigit;

    JTextField secondDigit;

    JButton calculateButton;

    JTextField  outputField;

    JFrame f=new JFrame();

    public App()

    {

        f.setTitle("Two Digit Calculator");

        // Create the components

       JLabel firstDigitLable = new JLabel("First Digit:");

       firstDigitLable.setBounds(20, 20, 200, 25);

       firstDigit = new JTextField();

       firstDigit.setBounds(200, 20, 150, 25);

       f.add(firstDigitLable);

       f.add(firstDigit);

       JLabel secondDigitLable = new JLabel("Second Digit:");

       secondDigitLable.setBounds(20, 60, 200, 25);

       secondDigit = new JTextField();

       secondDigit.setBounds(200, 60, 150, 25);

       f.add(secondDigitLable);

       f.add(secondDigit);

       calculateButton = new JButton("Calculate");

       calculateButton.setBounds(200, 100, 150, 25);

       f.add(calculateButton);

       JLabel outputLabel = new JLabel("Sum:");

       outputLabel.setBounds(20, 140, 200, 25);

       outputField = new JTextField();

       outputField.setBounds(200, 140, 150, 25);

       calculateButton.addActionListener(this);

       f.add(outputField);

       f.add(outputLabel);

       f.setSize(400, 300);

       f.setLayout(null);

       f.setVisible(true);

    }

    public static void main(String[] args) throws Exception {

    new App();

    }

    @Override

    public void actionPerformed(ActionEvent e) {

        int number1 = Integer.parseInt(firstDigit.getText());

        int number2 = Integer.parseInt(secondDigit.getText());

        int sum = number1 + number2;

        outputField.setText(String.valueOf(sum));

    }

}

**ItemListener:**

The Java **ItemListener** is notified whenever you click on the checkbox. It is notified against **ItemEvent**. The ItemListener interface is found in java.awt.event package. It has only one method: **itemStateChanged().**

Eample:

import java.awt.\*;

import java.awt.event.\*;

public class App implements ItemListener{

    Checkbox checkBox1,checkBox2;

    Label label;

   App()

   {

     Frame f=new Frame();

     label = new Label();

     label.setAlignment(Label.CENTER);

     label.setSize(400,100);

     checkBox1 = new Checkbox("C++");

     checkBox1.setBounds(100,100, 50,50);

     checkBox2 = new Checkbox("Java");

     checkBox2.setBounds(100,150, 50,50);

     f.add(checkBox1); f.add(checkBox2); f.add(label);

    checkBox1.addItemListener(this);

     checkBox2.addItemListener(this);

    f.setSize(300,300);

    f.setLayout(null);

    f.setVisible(true);

   }

   public void itemStateChanged(ItemEvent e) {

        if(e.getSource()==checkBox1)

        label.setText("C++ Checkbox: "

        + (e.getStateChange()==1?"checked":"unchecked"));

        if(e.getSource()==checkBox2)

        label.setText("Java Checkbox: "

        + (e.getStateChange()==1?"checked":"unchecked"));

    }

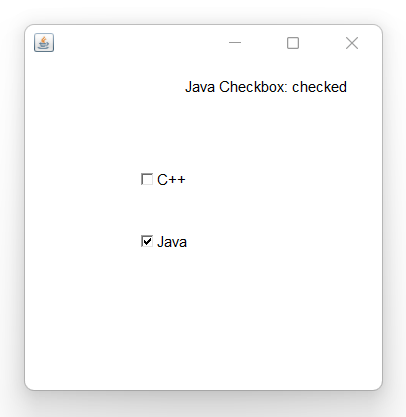
    public static void main(String[] args) throws Exception {

      new App();

    }

}

Output:



**KeyListener:**

The Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package. It has three methods.

1. keyPressed(KeyEvent e);
2. keyReleased(KeyEvent e);
3. keyTyped(KeyEvent e);

**Example:**

import java.awt.\*;

import java.awt.event.\*;

public class App implements KeyListener{

    Label l;

    TextArea area;

   App()

   {

     Frame f=new Frame();

     l = new Label();

     l.setBounds(20,50,100,20);

     area=new TextArea();

    area.setBounds(20,80,300, 300);

    area.addKeyListener(this);

    f.add(l);

    f.add(area);

    f.setSize(300,300);

    f.setLayout(null);

    f.setVisible(true);

   }

   public void keyPressed(KeyEvent e) {

    l.setText("Key Pressed");

    }

    public void keyReleased(KeyEvent e) {

    l.setText("Key Released");

    }

    public void keyTyped(KeyEvent e) {

    l.setText("Key Typed");

    }

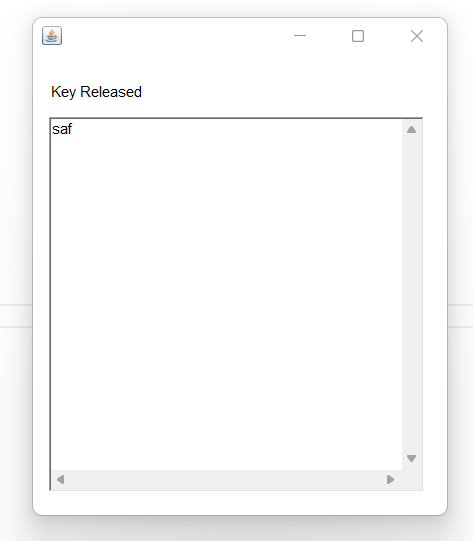
    public static void main(String[] args) throws Exception {

      new App();

    }

}

**Output:**

****

**MouseListener:**

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

**mouseClicked(), mousePressed(), mouseEntered(), mouseExited() and mouseReleased()**

**Mouse Listener Example:**

import java.awt.\*;

import java.awt.event.\*;

//1st Step Implement ActionListner

public class App implements MouseListener{

    Label l;

   App()

   {

     Frame f=new Frame();

     l=new Label();

     l.setBounds(20,50,100,20);

    f.add(l);

    f.addMouseListener(this);

    f.setSize(300,300);

    f.setLayout(null);

    f.setVisible(true);

   }

   public void mouseClicked(MouseEvent e) {

    l.setText("Mouse Clicked");

    }

    public void mouseEntered(MouseEvent e) {

    l.setText("Mouse Entered");

    }

    public void mouseExited(MouseEvent e) {

    l.setText("Mouse Exited");

    }

    public void mousePressed(MouseEvent e) {

    l.setText("Mouse Pressed");

    }

    public void mouseReleased(MouseEvent e) {

    l.setText("Mouse Released");

    }

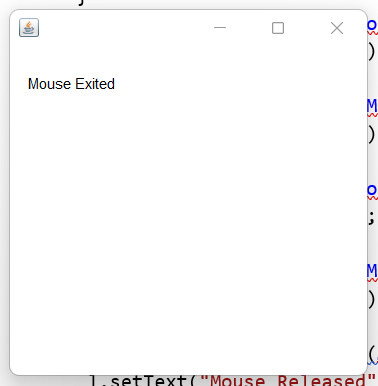
    public static void main(String[] args) throws Exception {

      new App();

    }

}

**Output:**

****

**LAB:** Write a GUI program using components to find sum and difference of two numbers. Use two text fields for giving input and a label for output. The program should display sum if user **presses mouse** and difference if user **release mouse** using adapter class.

**LAB:** Write a GUI program using components to find sum and difference of two numbers.Use two text fields for giving input and a label for output. The program should display sum if user **presses key** and difference if user **release key** using adapter class.

import javax.swing.\*;

import java.awt.event.\*;

public class App {

        JLabel l1, l2, l3;

        JTextField t1, t2, t3;

        JFrame f = new JFrame("New Key Adapter");

            App()

            {

            l1 = new JLabel("First Number:");

            l1.setBounds(10,10,200,20);

            t1 = new JTextField();

            t1.setBounds(150,10,200,20);

            l2 = new JLabel("Second Number:");

            l2.setBounds(10,40,200,20);

            t2 = new JTextField();

            t2.setBounds(150,40,200,20);

            l3 = new JLabel("Press any key");

            l3.setBounds(10,70,200,20);

            t3 = new JTextField();

            t3.setBounds(150,70,200,20);

            t3.addKeyListener(new checkkey());

f.add(l1); f.add(t1); f.add(l2); f.add(t2);f.add(l3);f.add(t3);

            f.setSize(600,600);

            f.setLayout(null);

            f.setVisible(true);

            }

            class checkkey extends KeyAdapter{

                public void keyPressed(KeyEvent e)

                {

                     int num1 = Integer.parseInt(t1.getText());

                     int num2 = Integer.parseInt(t2.getText());

                     int sum = num1 + num2;

                     t3.setText(String.valueOf(sum));

                }

            }

    public static void main(String[] args) throws Exception {

             new App();

    }

}

**MouseMotionListener:**

The Java **MouseMotionListener** is notified whenever you move or drag mouse. It is notified against **MouseEvent**. The **MouseMotionListener** interface is found in **java.awt.event** package. It has two methods.

The signature of 2 methods found in MouseMotionListener interface are given below:

1. mouseDragged(MouseEvent e);
2. mouseMoved(MouseEvent e);

**MouseMotionListener Example:**

import java.awt.\*;

import java.awt.event.\*;

//1st Step Implement ActionListner

public class App implements MouseMotionListener{

    Frame f;

   App()

   {

     f=new Frame();

    f.addMouseMotionListener(this);

    f.setSize(300,300);

    f.setLayout(null);

    f.setVisible(true);

   }

   public void mouseDragged(MouseEvent e) {

    Graphics g=f.getGraphics();

    g.setColor(Color.BLUE);

    g.fillOval(e.getX(),e.getY(),20,20);

    }

    public void mouseMoved(MouseEvent e) {}

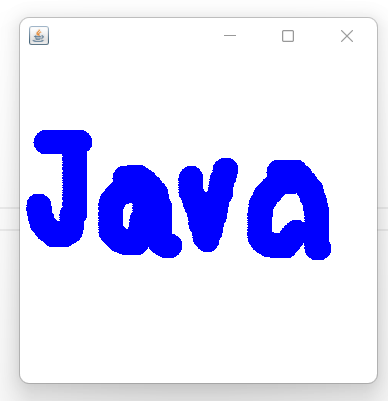
    public static void main(String[] args) throws Exception {

      new App();

    }

}

**Output:**

****

**JList and Its Event**

import javax.swing.\*;

import java.awt.event.\*;

public class App implements ActionListener{

  JFrame f;

  JList<String> list1;

  JLabel label;

  JButton b;

    App()

    {

        f= new JFrame();

        label = new JLabel();

        label.setSize(500,100);

         b=new JButton("Show");

        b.setBounds(200,150,80,30);

        String[] l1=new String[]{"Java","C#","C++"};

          list1 = new JList<>(l1);

          list1.setBounds(100,100, 75,75);

          f.add(list1); f.add(b); f.add(label);

          f.setSize(450,450);

          f.setLayout(null);

          f.setVisible(true);

          b.addActionListener(this);

    }

     public void actionPerformed(ActionEvent e) {

          String data = "";

          if (list1.getSelectedIndex() != -1) {

lable.setText(list1.getSelectedValue());

or

            data = "Programming language Selected: " + list1.getSelectedValue();

            label.setText(data);

            }

        label.setText(data);

    }

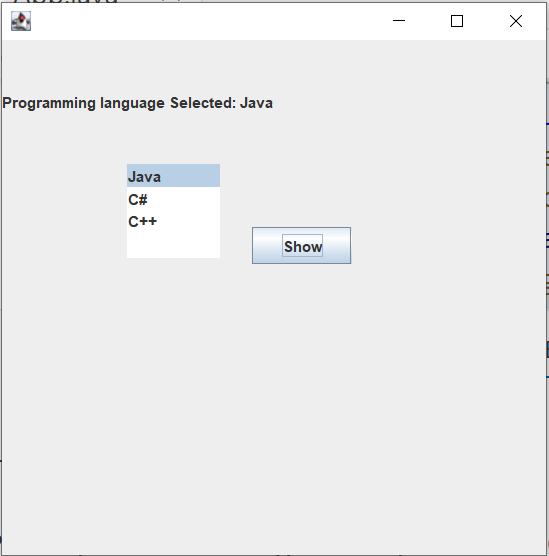
    public static void main(String[] args)

    {

       new App();

    }

}



**Menu, Menu Item:**

**Code:**

import javax.swing.\*;

public class App {

    public static void main(String[] args)

    {

        JMenu menu, submenu;

        JMenuItem i1, i2, i3, i4, i5;

        JFrame f= new JFrame("Menu and MenuItem Example");

        JMenuBar mb=new JMenuBar();

        menu=new JMenu("Menu");

        submenu=new JMenu("Sub Menu");

        i1=new JMenuItem("Item 1");

        i2=new JMenuItem("Item 2");

        i3=new JMenuItem("Item 3");

        i4=new JMenuItem("Item 4");

        i5=new JMenuItem("Item 5");

        menu.add(i1);

        menu.add(i2);

        menu.add(i3);

        submenu.add(i4);

        submenu.add(i5);

        menu.add(submenu);

        mb.add(menu);

        f.setJMenuBar(mb);

        f.setSize(400,400);

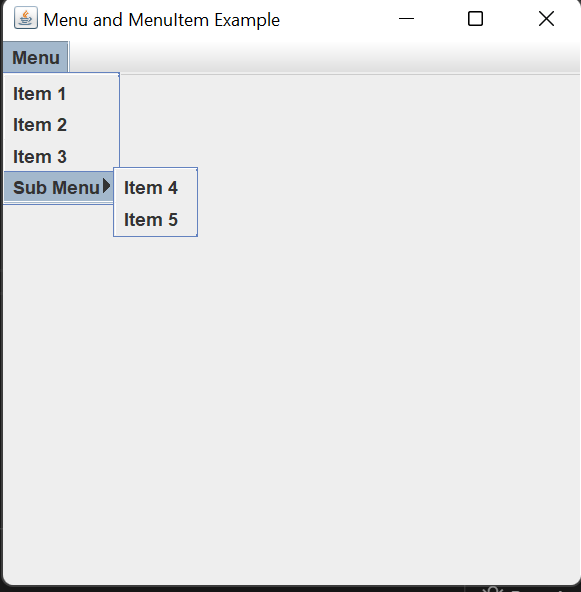
        f.setLayout(null);

        f.setVisible(true);

    }

}

**Output:**

****

**Jpopup Menu**

**Code:**

import javax.swing.\*;

import java.awt.event.\*;

public class App {

    public static void main(String[] args)

    {

        JFrame f= new JFrame("PopupMenu Example");

        JPopupMenu popupmenu = new JPopupMenu("Edit");

         JMenuItem cut = new JMenuItem("Cut");

         JMenuItem copy = new JMenuItem("Copy");

         JMenuItem paste = new JMenuItem("Paste");

         popupmenu.add(cut);

         popupmenu.add(copy);

         popupmenu.add(paste);

         f.addMouseListener(new MouseAdapter() {

            public void mouseClicked(MouseEvent e) {

                popupmenu.show(f , e.getX(), e.getY());

            }

         });

         f.add(popupmenu);

         f.setSize(300,300);

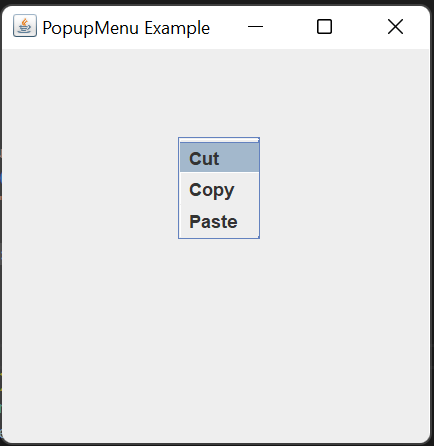
         f.setLayout(null);

         f.setVisible(true);

    }

}

**Output:**

****

**Option Dialog:**

**JOptionPane**class is used to display four types of dialog boxes

1. **MessageDialog**-  dialog box that displays a message making it possible to add icons to alert the user.
2. **ConfirmDialog**-  dialog box that besides sending a message, enables the user to answer a question.
3. **InputDialog**-  dialog box that besides sending a message, allows entry of a text.
4. **OptionDialog**-  dialog box that covers the three previous types.

Example:

import javax.swing.\*;

public class App {

    public static void main(String[] args) {

        JFrame frame = new JFrame("Main Window");

        JOptionPane.showMessageDialog(frame, "Hello Java");

        JOptionPane.showMessageDialog(frame,"Successfully Updated.","Alert",JOptionPane.WARNING\_MESSAGE);

        String name=JOptionPane.showInputDialog(frame,"Enter Name");

        int result = JOptionPane.showConfirmDialog(null, "Do you want to remove item now?");

        frame.setSize(350,350);

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        frame.setVisible(true);

    }

}

**Program to create a simple JDialog**

import javax.swing.\*;

import java.awt.event.\*;

public class App extends JFrame implements ActionListener {

    static JFrame f;

    public static void main(String[] args) {

      // create a new frame

      f = new JFrame("frame");

      // create a object

      App s = new App();

      JButton b = new JButton("click");

      b.setBounds(100,120,100,40);

      // add actionlistener to button

      b.addActionListener(s);

      f.add(b);

      // set the size of frame

      f.setSize(400, 400);

      f.setLayout(null);

      f.setVisible(true);

    }

    public void actionPerformed(ActionEvent e)

    {

        String s = e.getActionCommand();

        if (s.equals("click")) {

            // create a dialog Box

            JDialog d = new JDialog(f, "dialog Box");

            // create a label

            JLabel l = new JLabel("this is a dialog box");

            d.add(l);

            // setsize of dialog

            d.setSize(100, 100);

            // set visibility of dialog

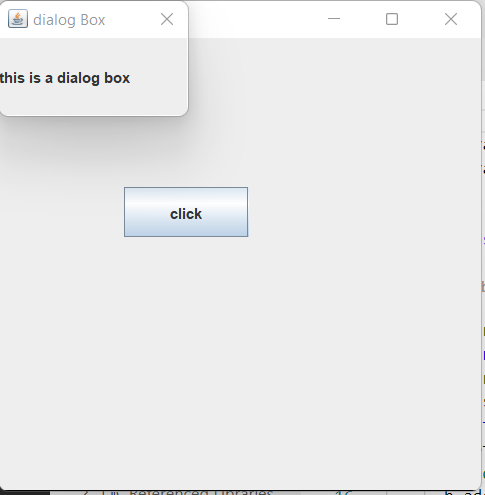
            d.setVisible(true);

        }

    }

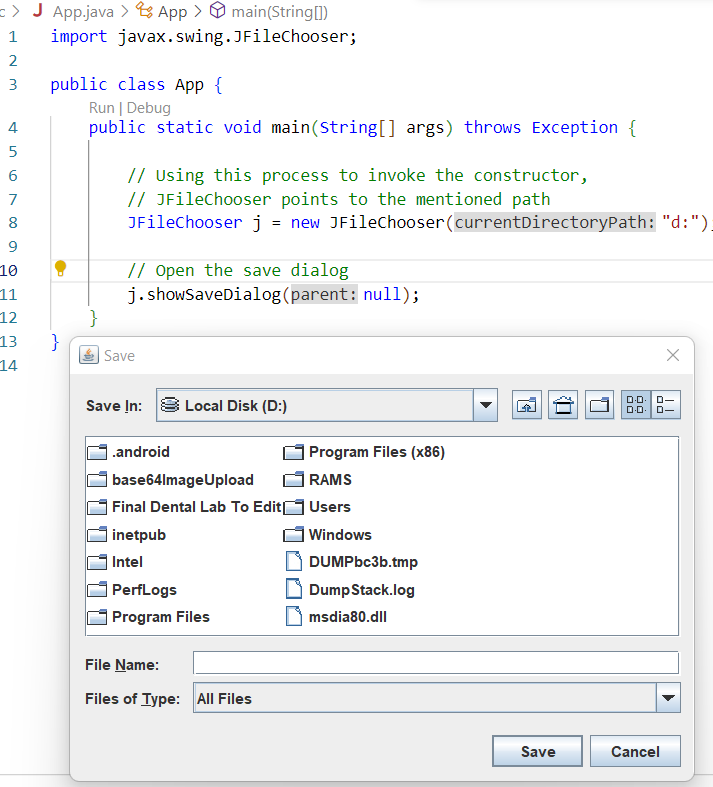
}

Output:



|  |
| --- |
|  |
| File Chooser:  import javax.swing.JFileChooser;  public class App {      public static void main(String[] args) throws Exception {            // Using this process to invoke the constructor,          // JFileChooser points to user's default directory          JFileChooser j = new JFileChooser();            // Open the save dialog          j.showSaveDialog(null);      }  }  Output: |

**Example:**

****

Color Chooser:

import java.awt.event.\*;

import java.awt.\*;

import javax.swing.\*;

public class App extends JFrame implements ActionListener{

    // create a button

    JButton b = new JButton("color");

    Container c = getContentPane();

    // Constructor

    App()

    {

        // set Layout

        c.setLayout(new FlowLayout());

        // add Listener

        b.addActionListener(this);

        // add button to the Container

        c.add(b);

        setSize(400, 400);

        setVisible(true);

        setDefaultCloseOperation(EXIT\_ON\_CLOSE);

    }

    public void actionPerformed(ActionEvent e)

    {

        Color initialcolor = Color.RED;

        // color chooser Dialog Box

        Color color = JColorChooser.showDialog(this,

                    "Select a color", initialcolor);

        // set Background color of the Container

        c.setBackground(color);

    }

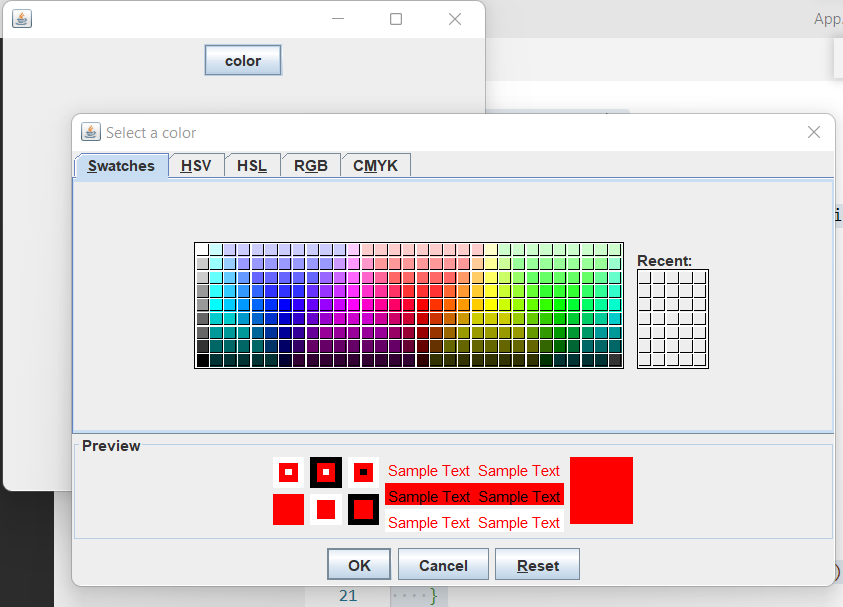
    public static void main(String[] args) throws Exception {

        App ch = new App();

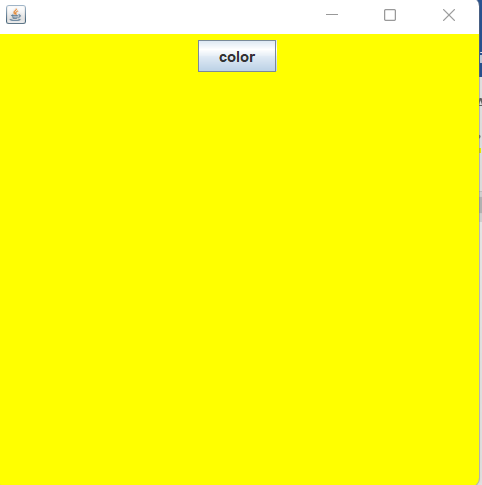
    }

}

Output:



After Choosing Color Yellow Pressing Ok:



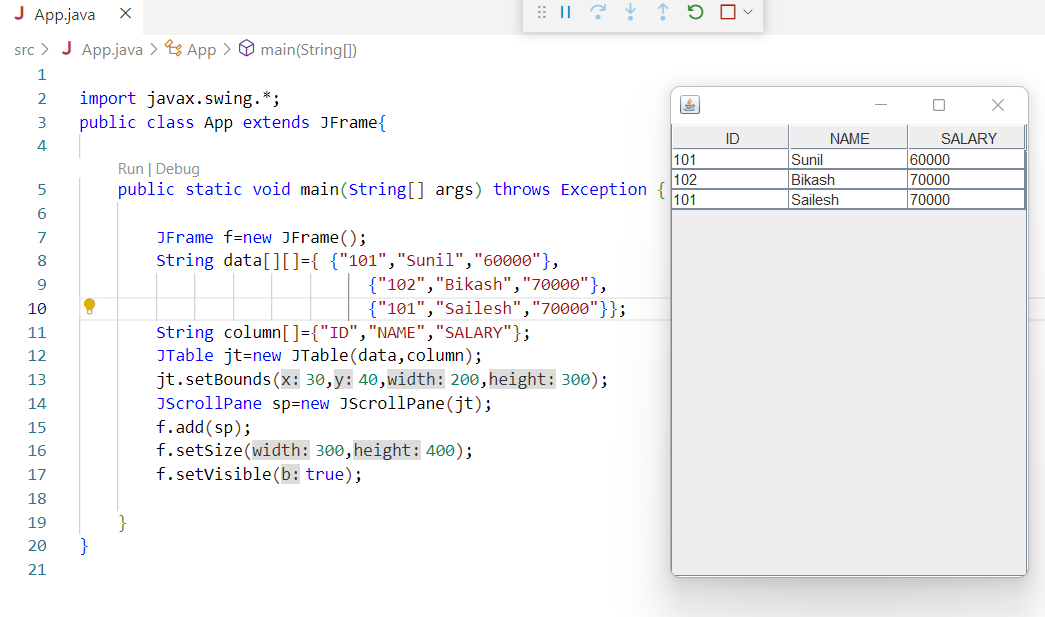
**Tables:**

In Java, tables are used to arrange data into *columns* and *rows*. A column is space that runs horizontally on a table, while a row is a space that runs horizontally in your table. The intersection between a column and a row is called a *cell* and is used to hold singular piece of data.

In Java, developers can use the **JTable** method to create a table in their applications. **JTable** is a Swing component that inherits from the **JComponent** class.

*JTable table = new JTable (row, column);*

***Lab:Write a java program to display data in tabular format using JTable.***

**

**Java Adapter Classes:**

Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code.

The adapter classes are found in **java.awt.event**, **java.awt.dnd** *(drag and drop)* and **javax.swing.event** packages. The Adapter classes with their corresponding listener interfaces are given below.

**java.awt.event Adapter classes**

|  |  |
| --- | --- |
| **Adapter class** | **Listener interface** |
| WindowAdapter | [WindowListener](https://www.javatpoint.com/java-windowlistener) |
| KeyAdapter | [KeyListener](https://www.javatpoint.com/java-keylistener) |
| MouseAdapter | [MouseListener](https://www.javatpoint.com/java-mouselistener) |
| MouseMotionAdapter | [MouseMotionListener](https://www.javatpoint.com/java-mousemotionlistener) |
| FocusAdapter | FocusListener |
| ComponentAdapter | ComponentListener |
| ContainerAdapter | ContainerListener |
| HierarchyBoundsAdapter | HierarchyBoundsListener |

**java.awt.dnd Adapter classes**

|  |  |
| --- | --- |
| **Adapter class** | **Listener interface** |
| DragSourceAdapter | DragSourceListener |
| DragTargetAdapter | DragTargetListener |

**javax.swing.event Adapter classes**

|  |  |
| --- | --- |
| **Adapter class** | **Listener interface** |
| MouseInputAdapter | MouseInputListener |
| InternalFrameAdapter | InternalFrameListener |

**Example:**

import java.awt.\*;

import java.awt.event.\*;

public class App{

    Frame f;

    App(){

    f=new Frame("Window Adapter");

    f.addWindowListener(new WindowAdapter()

{

    public void windowClosing(WindowEvent e) {

    f.dispose();

    }

    });

    f.setSize(400,400);

    f.setLayout(null);

    f.setVisible(true);

    }

    public static void main(String[] args) throws Exception {

      new App();

    }

}