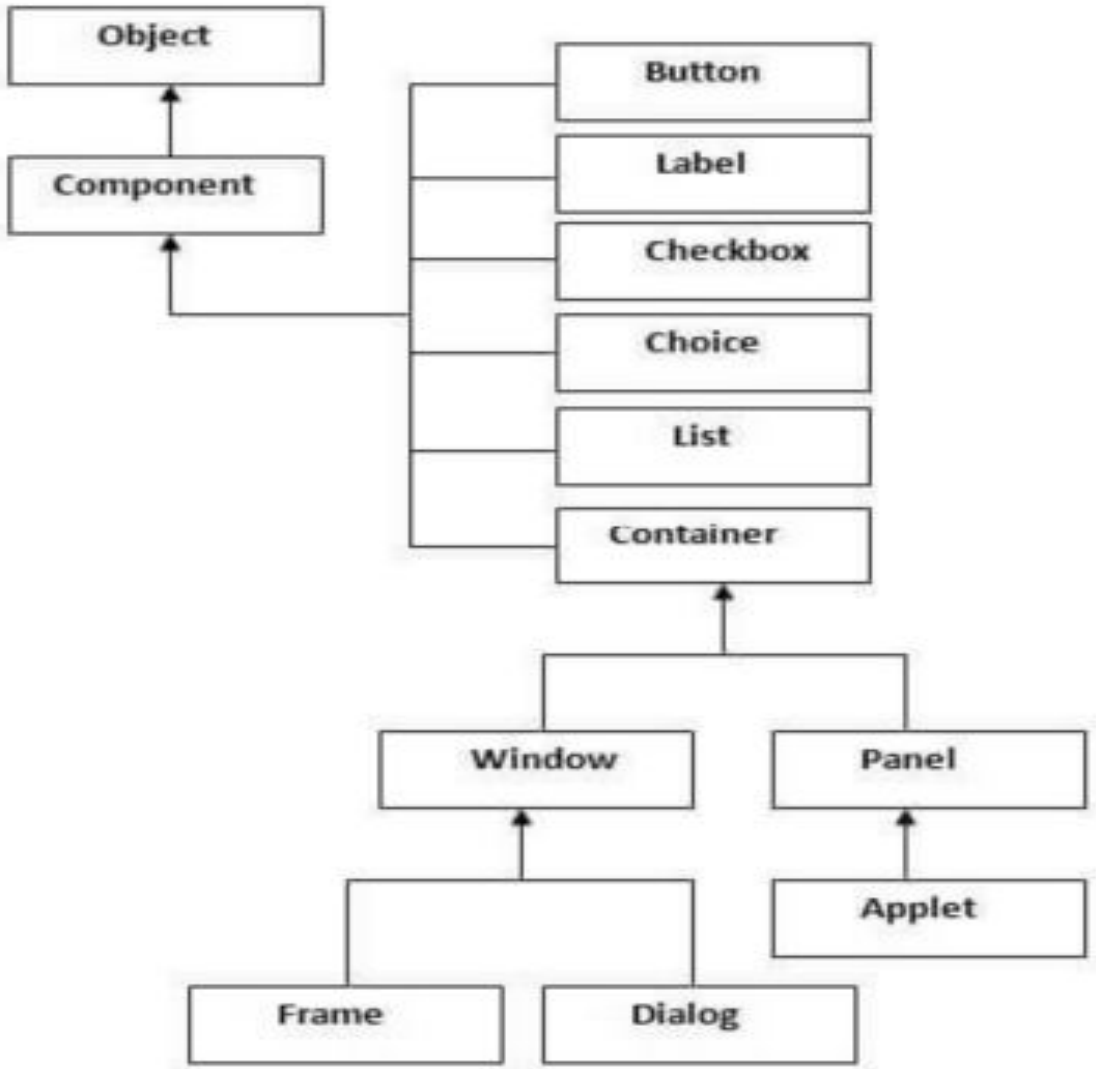
Unit 1
GUI Programming

Java AWT

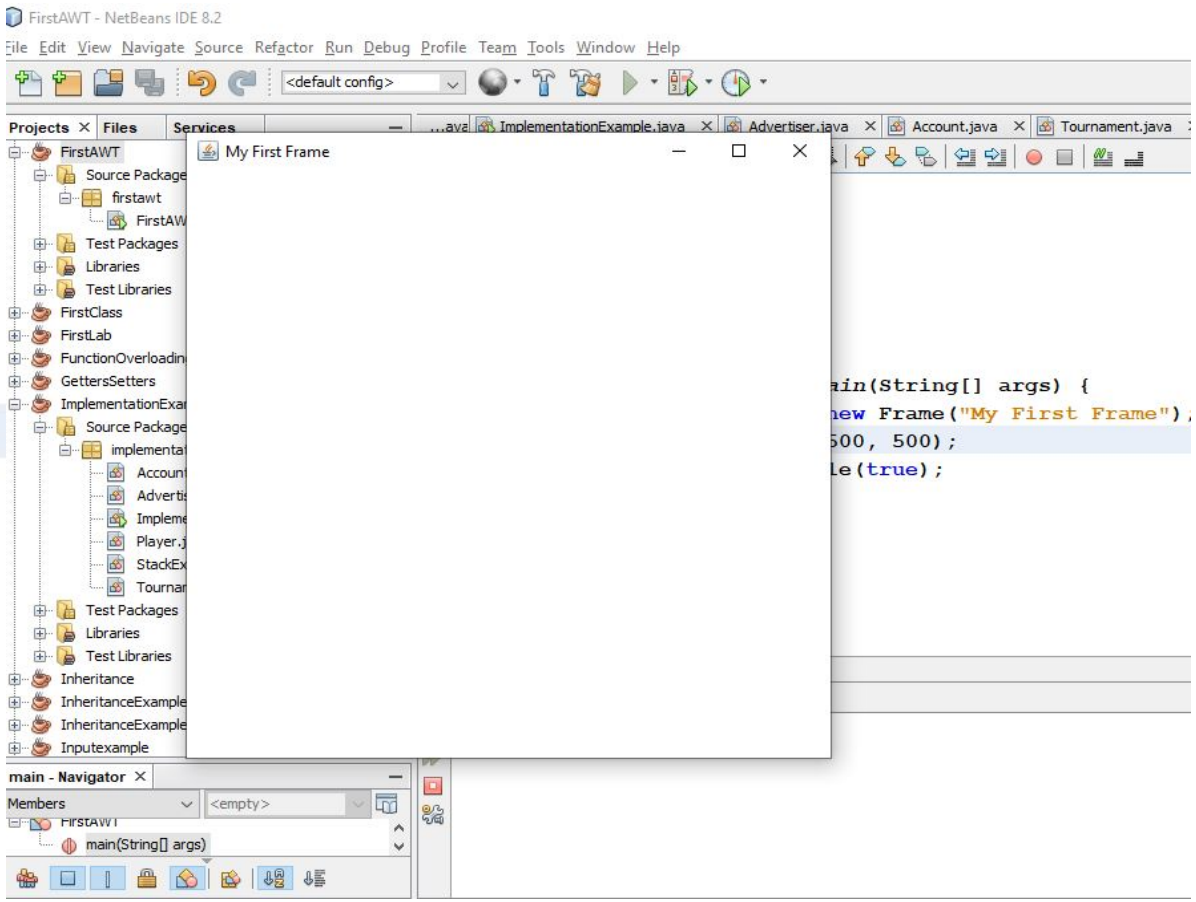
- Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.
- Java AWT components are **platform-dependent i.e**. components are displayed according to the view of operating system.
- AWT is heavyweight, consuming more resource

Java AWT Class Hierarchy



First Java AWT Program

```java
import java.awt.*;

public class FirstAWT {

    public static void main(String[] args) {
        Frame myFrame = new Frame("My First Frame");
        myFrame.setSize(500, 500);
        myFrame.setVisible(true);
    }

}
```



Note: The close button of the frame will not work as we have not handled this event (which can be done).To close the frame we can do ⍰

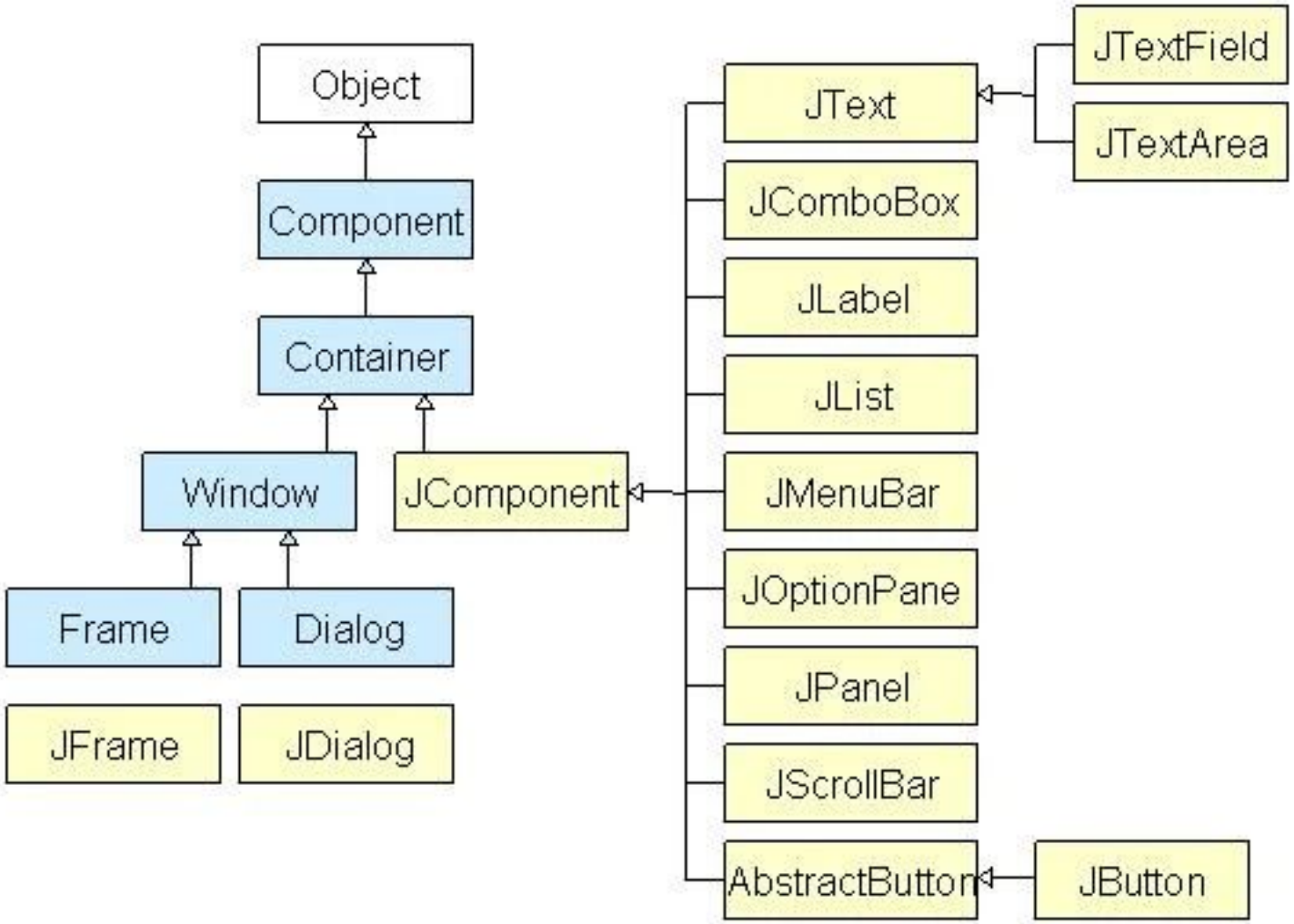
Java Swing

- Java Swing tutorial is a part of Java Foundation Classes (JFC) that is used to create window-based applications.
- The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.
- It is built on the top of AWT API and entirely written in java.
- **Unlike AWT, Java Swing provides platform-independent and lightweight components**
- We need to import javax.swing.*;

AWT Vs Swing

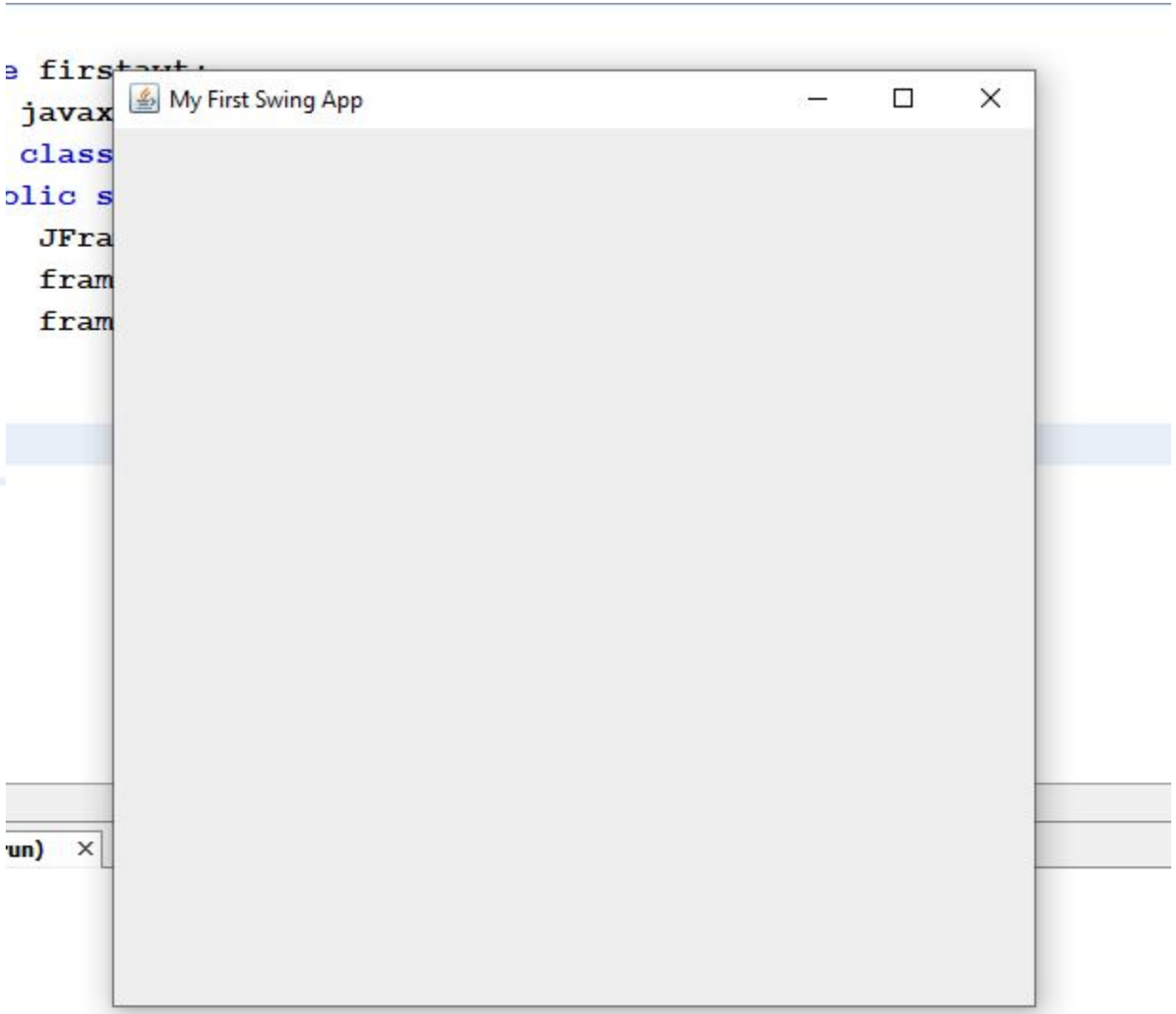
No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Java Swing Hierarchy



Java Swing

```java
import javax.swing.*;
public class SwingExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My First Swing App");
        frame.setSize(500, 500);
        frame.setVisible(true);
    }
}
```



JFrame

- The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class.
- JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI.
- JFrame Class is the top-level container that contains **content pane.** All visible components are contained in the content pane.
- Unlike Frame, JFrame has the option to hide or close the window with the help of *setDefaultCloseOperation(int)* method.
- Closing the frame will not close the java program, just hide the frame. To close the application,

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

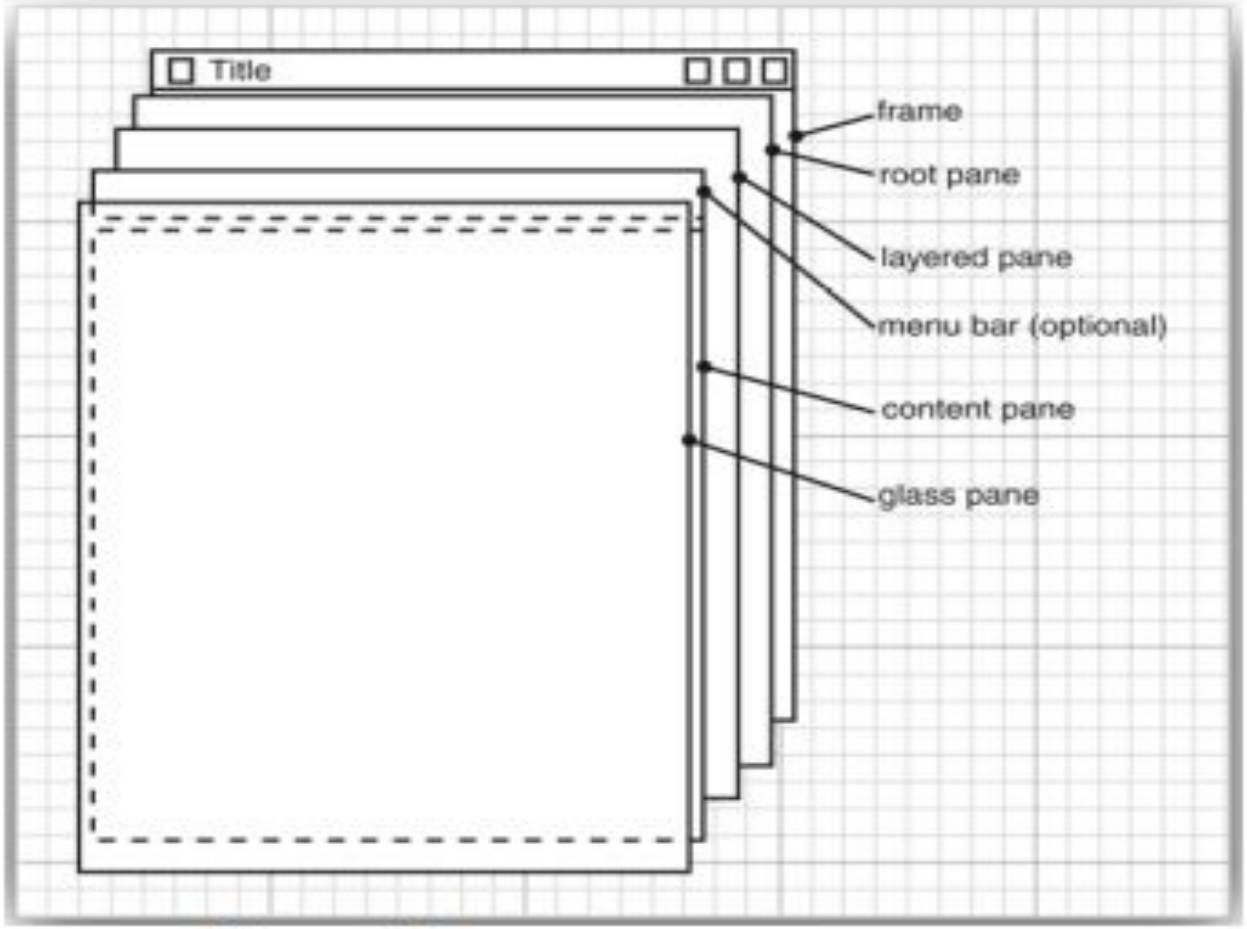
JFrame

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

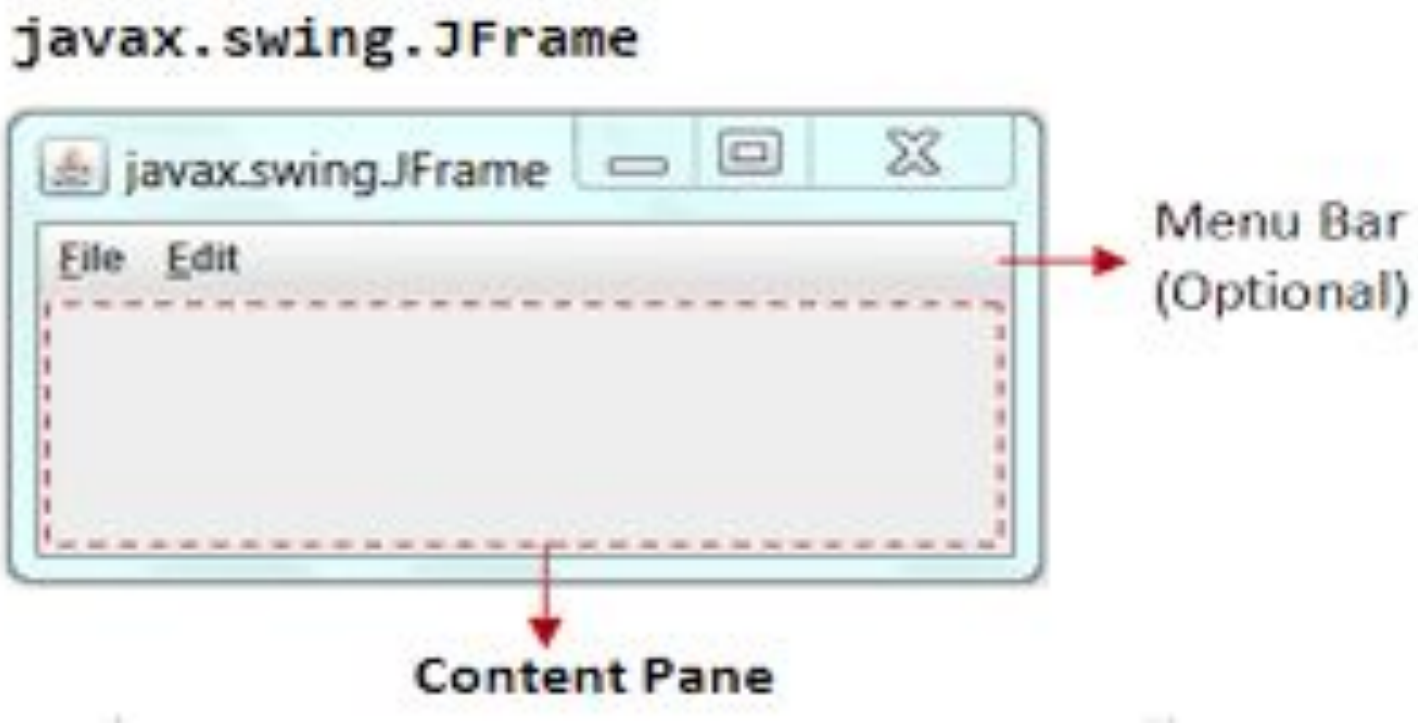
- By default, it uses *HIDE_ON_CLOSE,* we can also write *DO_NOTHING_ON_CLOSE* which does not allow user to close the window.
- We can also prevent user from resizing the frame

```
frame.setResizable(false);
```

Jframe-Internal Structure

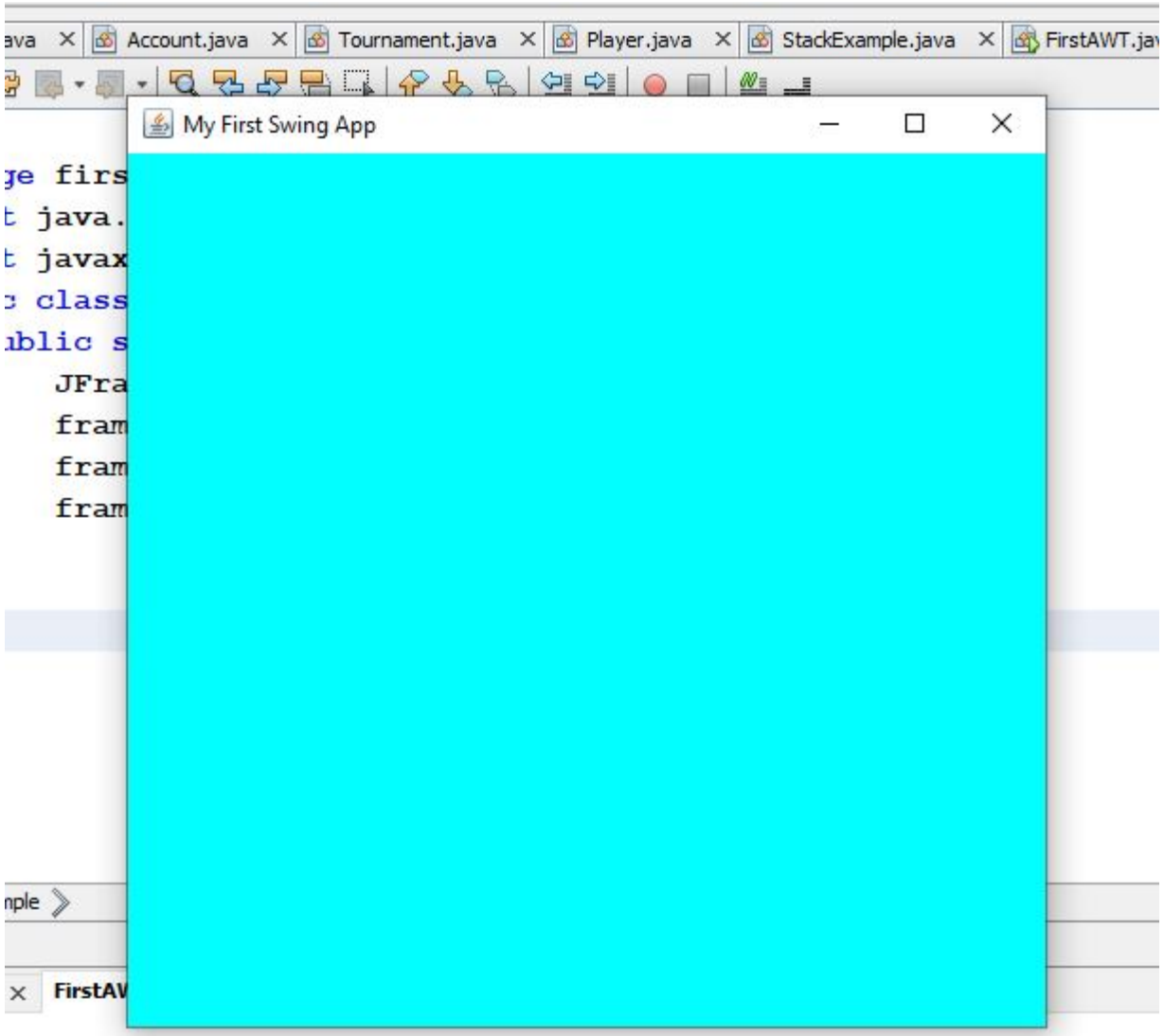


Jframe-Content Pane



Jframe-Setting Background color

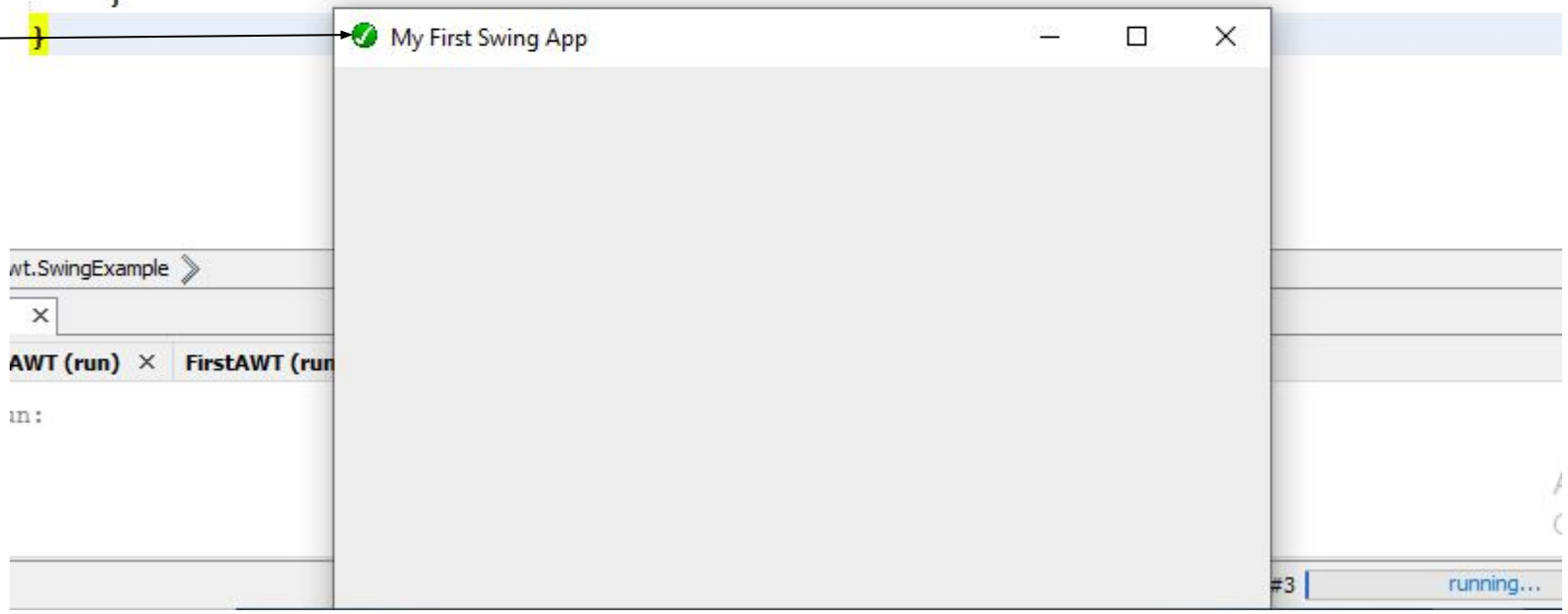
```java
import java.awt.Color;
import javax.swing.*;
public class SwingExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My First Swing App");
        frame.setSize(500, 500);
        frame.setVisible(true);
        frame.getContentPane().setBackground(Color.cyan);

    }
}
```



JFrame-Changing Icon on the top

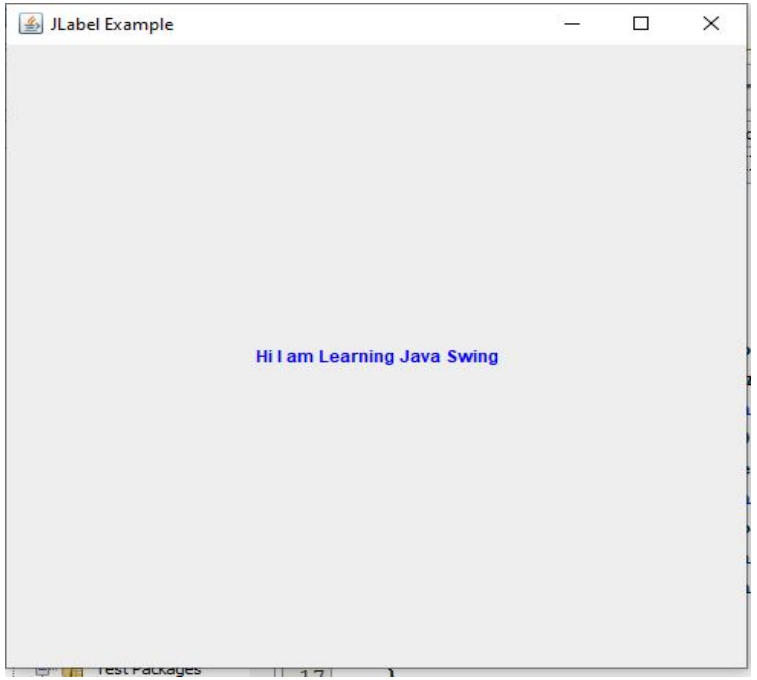
```
public class SwingExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My First Swing App");
        frame.setSize(500, 500);
        frame.setVisible(true);
       ImageIcon img = new ImageIcon("C:\\Users\\Sudar\\Desktop\\icon.jpg");
       frame.setIconImage(img.getImage());

    }
}
```

Icon Changed



Swing Components-JLabel

- It is used to display a single line of read only text.
- We define a JLabel and add it to our frame.



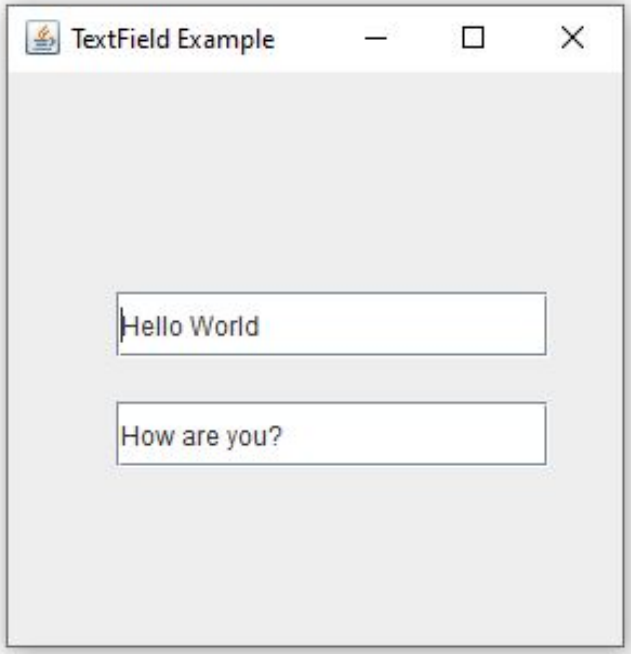

```java
import java.awt.Color;
import javax.swing.*;

public class JLabelExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JLabel Example");
        frame.setSize(500, 500);
        frame.setVisible(true);
        JLabel label = new JLabel("Hi I am Learning Java Swing");
        label.setForeground(Color.BLUE); //sets the text color to blue
        label.setVerticalAlignment(JLabel.CENTER);  //aligns the label to center of frame vertically
        label.setHorizontalAlignment(JLabel.CENTER); //aligns the label to center of frame horizontally
        frame.add(label);
    }
}
```

Swing Components-JTextField

- The object of a JTextField class is a text component that allows the editing of a single line text.
- It inherits JTextComponent class.
- The constructor of the class are :
  a. **JTextField()** : constructor that creates a new TextField
  b. **JTextField(int columns)** : constructor that creates a new empty TextField with specified number of columns.
  c. **JTextField(String text)** : constructor that creates a new empty text field initialized with the given string.
  d. **JTextField(String text, int columns)** : constructor that creates a new empty textField with the given string and a specified number of columns .
  e. **JTextField(Document doc, String text, int columns)** : constructor that creates a textfield that uses the given text storage model and the given number of columns.

Swing Components-JTextField Example

```java
import javax.swing.*;


public class JavaSwing {
    public static void main(String[] args) {
        JFrame f= new JFrame("TextField Example");
        JTextField t1,t2;
        t1=new JTextField("Hello World");
        t1.setBounds(50,100, 200,30);
        t2=new JTextField("How are you?");
        t2.setBounds(50,150, 200,30);
        f.add(t1); f.add(t2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



Swing Components-JTextField Example

❖ The same thing can also be done using constructor as below:

JavaSwing.java | FirstSwing.java | DatabaseConnectivity.java | JDBCConnectivityExamp...

```java
import javax.swing.*;

public class JavaSwing {
    public static void main(String[] args) {
        new FrameExample();
    }
}
```

FrameExample.java

```java
import javax.swing.*;

public class FrameExample extends JFrame{
    public FrameExample() {
    this.setTitle("JTextField Example");
    JTextField t1,t2;
    t1=new JTextField("Hello World");
    t1.setBounds(50,100, 200,30);
    t2=new JTextField("How are you?");
    t2.setBounds(50,150, 200,30);
    this.add(t1); this.add(t2);
    this.setSize(300,300);
    this.setLayout(null);
    this.setVisible(true);
    }
}
```

Swing Components-JTextArea

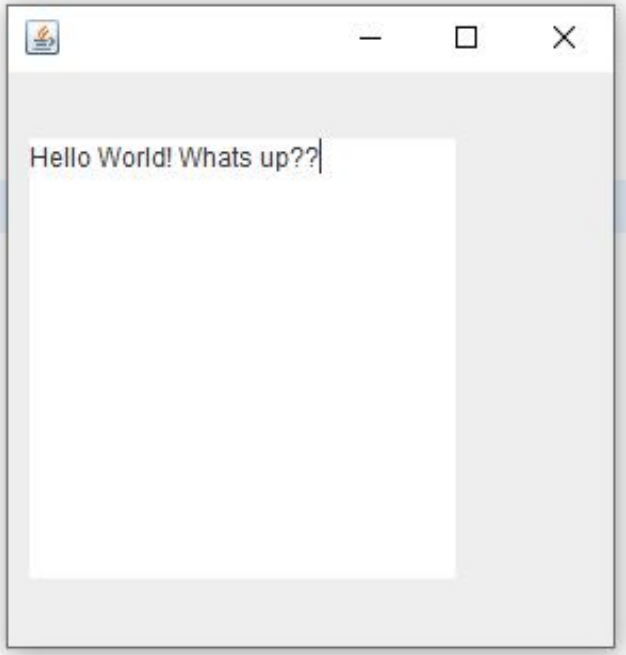
- It represents a multi line area that displays text. It is used to edit the text .
- JTextArea inherits JComponent class.
- Constructors of JTextArea are:
  - JTextArea() : constructs a new blank text area .
  - JTextArea(String s) : constructs a new text area with a given initial text.
  - JTextArea(int row, int column) : constructs a new text area with a given number of rows and columns.
  - JTextArea(String s, int row, int column) : constructs a new text area with a given number of rows and columns and a given initial text.

Swing Components-JTextArea Methods

Methods	Description
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

Swing Components-JTextArea Example

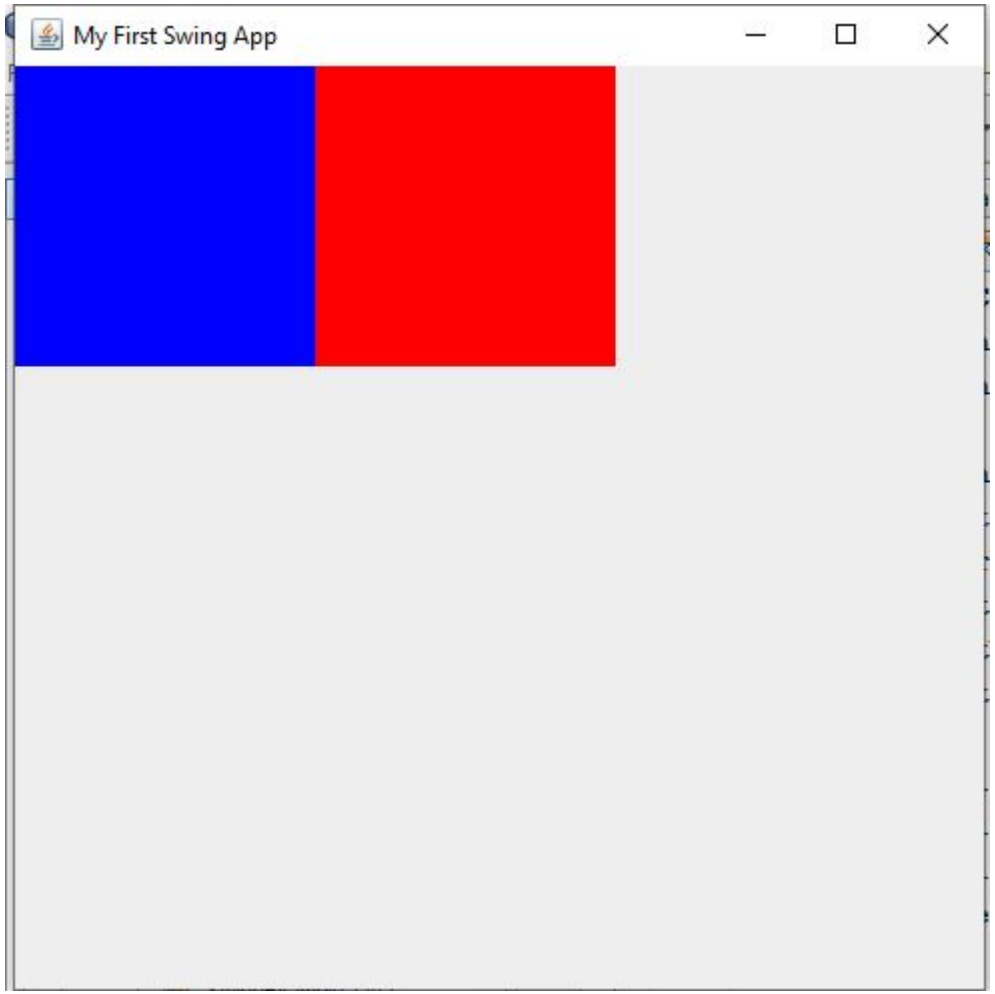
```java
import javax.swing.*;

public class FrameExample extends JFrame{
    public FrameExample() {
        JTextArea area=new JTextArea("Hello World! Whats up??");
        area.setBounds(10,30, 200,200);
        this.add(area);
        this.setSize(300,300);
        this.setLayout(null);
        this.setVisible(true);
    }
}
```



Swing Components-JPanel Example

- The JPanel is a simplest container class.
- It provides space in which an application can attach any other component.

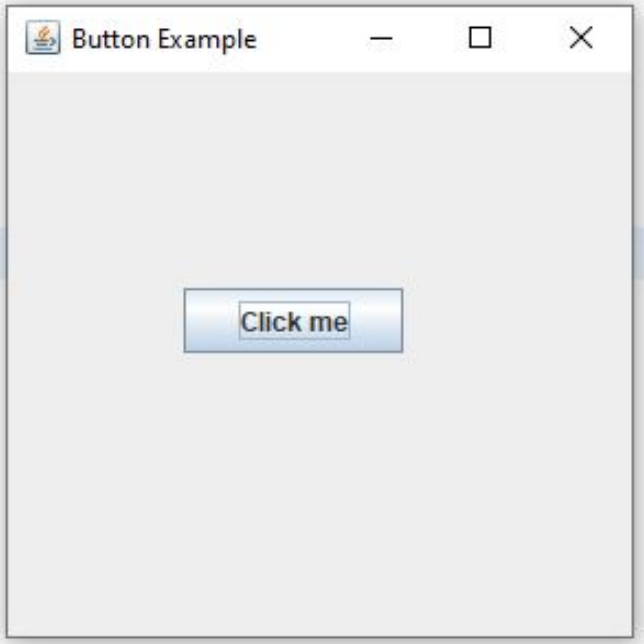
```java
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class JPanelExample {
        public static void main(String[] args) {
        JFrame frame = new JFrame("My First Swing App");
        frame.setSize(500, 500);
        frame.setVisible(true);
        frame.setLayout(null); //to set the layout

        JPanel bluePanel = new JPanel();
        bluePanel.setBackground(Color.BLUE);
        bluePanel.setBounds(0, 0, 150, 150);
        JPanel redPanel = new JPanel();
        redPanel.setBackground(Color.RED);
        redPanel.setBounds(150, 0, 150, 150);
        frame.add(bluePanel);
        frame.add(redPanel);
    }
}
```



Swing Components-JButton

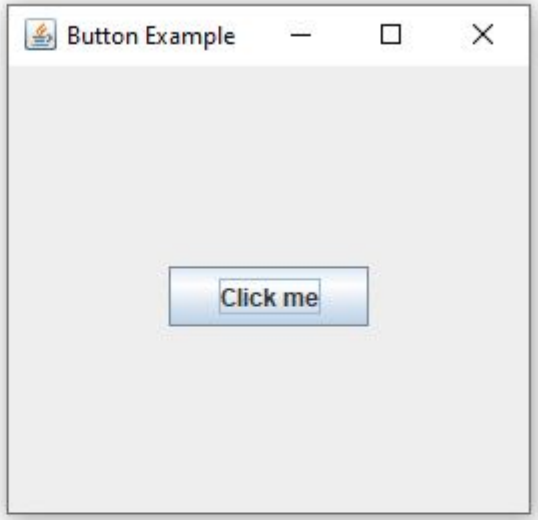
❖ The JButton class is used to create a labeled button that has platform independent implementation.

```java
import javax.swing.*;

public class FrameExample{
    public static void main(String[] args) {
        JFrame frame = new JFrame("Button Example");
        JButton btn = new JButton("Click me");
        btn.setBounds(80,100,100,30);
        frame.add(btn);
        frame.setSize(300,300);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```



Swing Components-Jbutton Example with Event Handling

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

public class FrameExample{
    public static void main(String[] args) {
        JFrame frame = new JFrame("Button Example");
        JButton btn = new JButton("Click me");
        btn.setBounds(80,100,100,30);
        frame.add(btn);
        frame.setSize(300,300);
        frame.setLayout(null);
        frame.setVisible(true);

        btn.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                System.out.println("Button is clicked");

            }
        });
    }
}
```



Markers Properties Servers Data Source Explorer Snippets Console

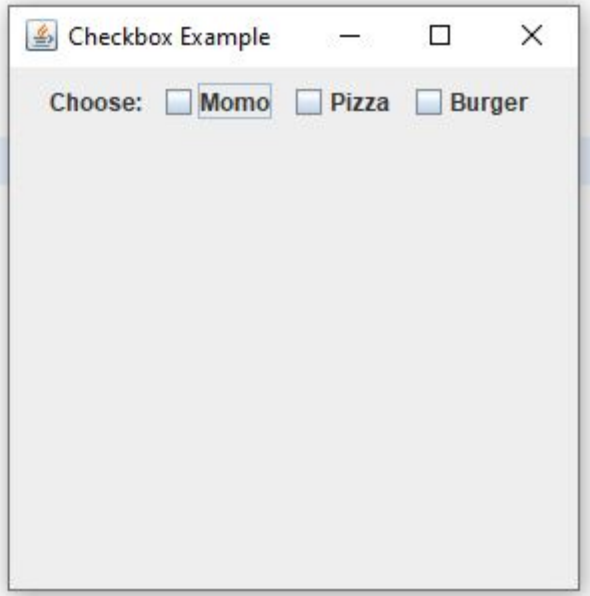
FrameExample [Java Application] C:\Program Files\Java\jre1.8.0_74\bin\javaw.exe (Mar 28, 2023, 8:29:50 AM)

```
Button is clicked
```

Swing Components-JCheckBox

- The JCheckBox class is used to create a checkbox.
- It is used to turn an option on (true) or off (false).

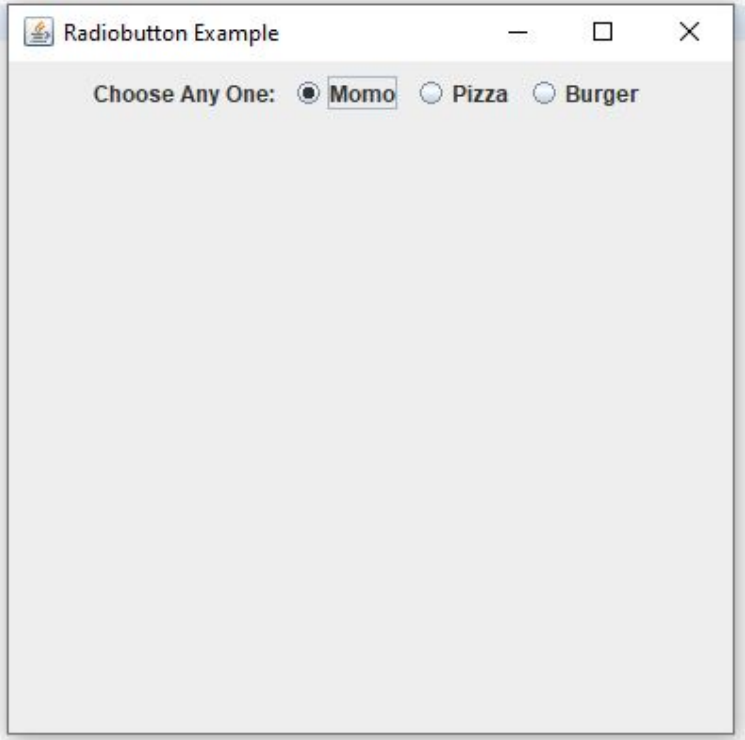
```
public class CheckboxExample{
    public static void main(String[] args) {
        JFrame frame = new JFrame("Checkbox Example");
        JPanel pnl = new JPanel();
        JLabel lbl = new JLabel("Choose: ");
        JCheckBox check1 = new JCheckBox("Momo");
        check1.setBounds(20, 100, 80, 50);
        JCheckBox check2 = new JCheckBox("Pizza");
        check2.setBounds(100, 100, 80, 50);
        JCheckBox check3 = new JCheckBox("Burger");
        check2.setBounds(180, 100, 80, 50);
        pnl.add(lbl);
        pnl.add(check1);
        pnl.add(check2);
        pnl.add(check3);
        frame.add(pnl);
        frame.setSize(300, 300);
        frame.setVisible(true);
        frame.setLayout(null);
    }
}
```



Swing Components- JRadioButton

- ❖ The JRadioButton class is used to create a radio button.
- ❖ It is used to choose one option from multiple options.
- ❖ **It should be added in <u>ButtonGroup to select one radio button only</u>**

```java
import javax.swing.*;

public class RadioButtonExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Radiobutton Example");
        JPanel pnl = new JPanel();
        JLabel lbl = new JLabel("Choose Any One: ");
        JRadioButton radio1 = new JRadioButton("Momo");
        radio1.setBounds(20, 100, 80, 50);
        JRadioButton radio2 = new JRadioButton("Pizza");
        radio2.setBounds(100, 100, 80, 50);
        JRadioButton radio3 = new JRadioButton("Burger");
        radio3.setBounds(180, 100, 80, 50);
        ButtonGroup bg = new ButtonGroup();
        bg.add(radio1);
        bg.add(radio2);
        bg.add(radio3);
        pnl.add(lbl);
        pnl.add(radio1);
        pnl.add(radio2);
        pnl.add(radio3);
        frame.add(pnl);
        frame.setSize(400, 400);
        frame.setVisible(true);
        frame.setLayout(null);
    }
}
```

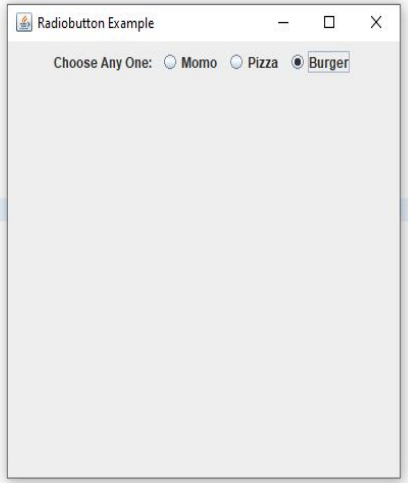


Swing Components- JRadioButton Event Handling Example

```java
import javax.swing.*;
public class RadioButtonExample{
    public static void main(String[] args) {
        JRadioButton radio1;JRadioButton radio2;
        JRadioButton radio3;
        JFrame frame = new JFrame("Radiobutton Example");
        JPanel pnl = new JPanel();
        JLabel lbl = new JLabel("Choose Any One: ");
        radio1 = new JRadioButton("Momo");
        radio1.setBounds(20, 100, 80, 50);
        radio2 = new JRadioButton("Pizza");
        radio2.setBounds(100, 100, 80, 50);
        radio3 = new JRadioButton("Burger");
        radio3.setBounds(180, 100, 80, 50);
        ButtonGroup bg = new ButtonGroup();
        bg.add(radio1);
        bg.add(radio2);
        bg.add(radio3);
        pnl.add(lbl);
        pnl.add(radio1);
        pnl.add(radio2);
        pnl.add(radio3);
        frame.add(pnl);
        frame.setSize(400, 400);
        frame.setVisible(true);
        frame.setLayout(null);
        radio1.addActionListener(new CustomActionListener());
        radio2.addActionListener(new CustomActionListener());
        radio3.addActionListener(new CustomActionListener());
    }
}
```

JavaSwing.java | FirstSwing.java | DatabaseConn... | FrameExampl... | JDBCConnecti... | FirstApplic... | RadioButton... | CustomAction...

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

public class CustomActionListener implements ActionListener{

    @Override
    public void actionPerformed(ActionEvent e) {
        JRadioButton button = (JRadioButton) e.getSource();
        System.out.println("You have selected "+ button.getText());
    }

}
```



Problems | Javadoc | Declaration | Console

RadioButtonExample [Java Application] C:\Program Files\Java\jre1.8.0_74\bin\javaw.exe (Mar 29, 2023, 1:05:43 PM)

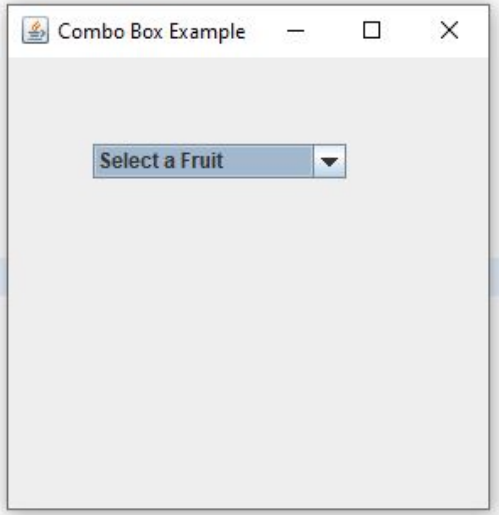
```
You have selected Momo
You have selected Pizza
You have selected Burger
```

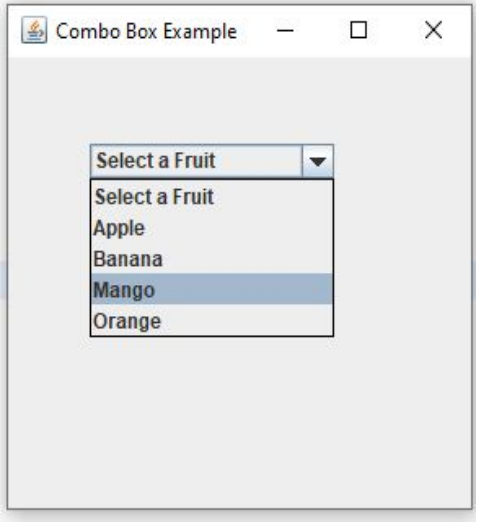
Swing Components- JComboBox

- JComboBox shows a popup menu that shows a list and the user can select a option from that specified list .
- JComboBox can be editable or read- only depending on the choice of the programmer .
- Constructor of the JComboBox are:
  - JComboBox() : creates a new empty JComboBox .
  - JComboBox(ComboBoxModel M) : creates a new JComboBox with items from specified ComboBoxModel
  - JComboBox(E [ ] i) : creates a new JComboBox with items from specified array.
  - **JComboBox(Vector items) : creates a new JComboBox with items from the specified vector**

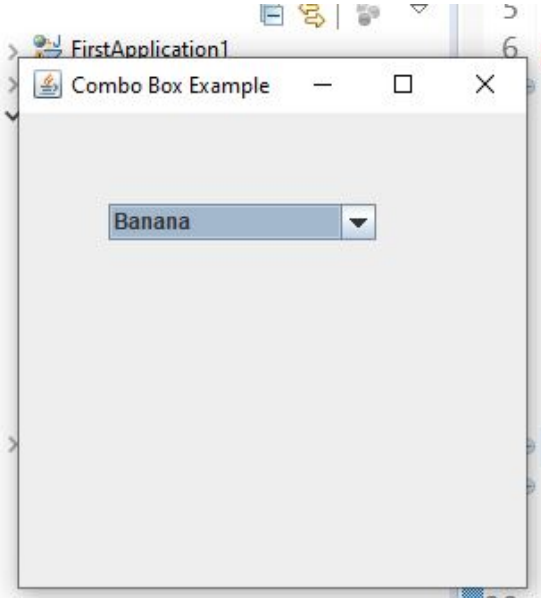
Swing Components- JComboBox

```java
import javax.swing.*;

public class JComboBoxExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Combo Box Example");
        String[] fruintList = {"Select a Fruit","Apple", "Banana","Mango","Orange"};
        JComboBox combo = new JComboBox(fruintList);
        frame.add(combo);
        combo.setBounds(50, 50, 150, 20);
        frame.setLayout(null);
        frame.setSize(300,300);
        frame.setVisible(true);
    }
}
```





Swing Components- JComboBox Example




```java
public class JComboBoxExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Combo Box Example");
        String[] fruintList = {"Select a Fruit","Apple", "Banana","Mango","Orange"};
        JComboBox combo = new JComboBox(fruintList);
        frame.add(combo);
        combo.setBounds(50, 50, 150, 20);
        frame.setLayout(null);
        frame.setSize(300,300);
        frame.setVisible(true);
        combo.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String selectedItem=combo.getSelectedItem().toString();
                System.out.println("You have selected "+ selectedItem);
            }
        });
    }
}
```

Problems @ Javadoc Declaration Console

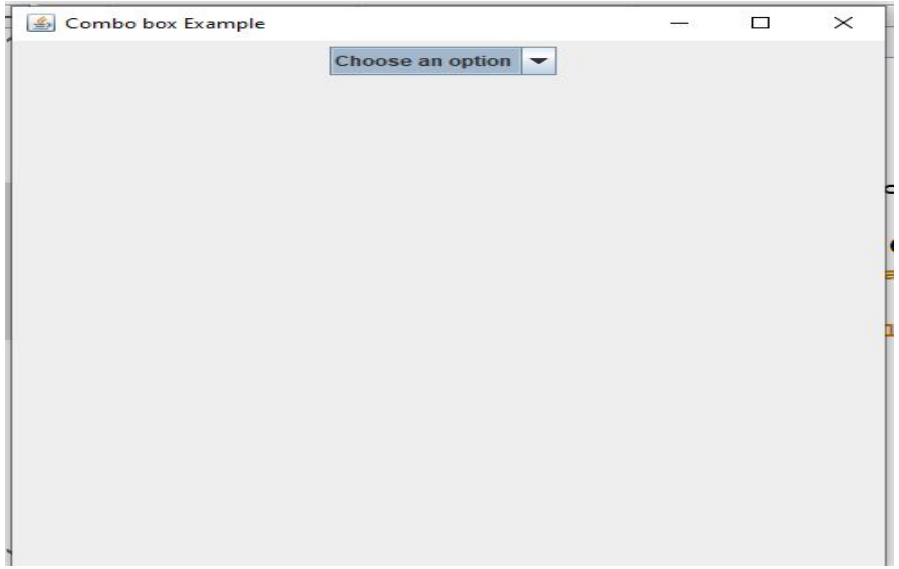
JCOmboBoxExample [Java Application] C:\Program Files\Java\jre1.8.0_74\bin\javaw.exe (Mar 29, 2023, 1:44:27 PM)

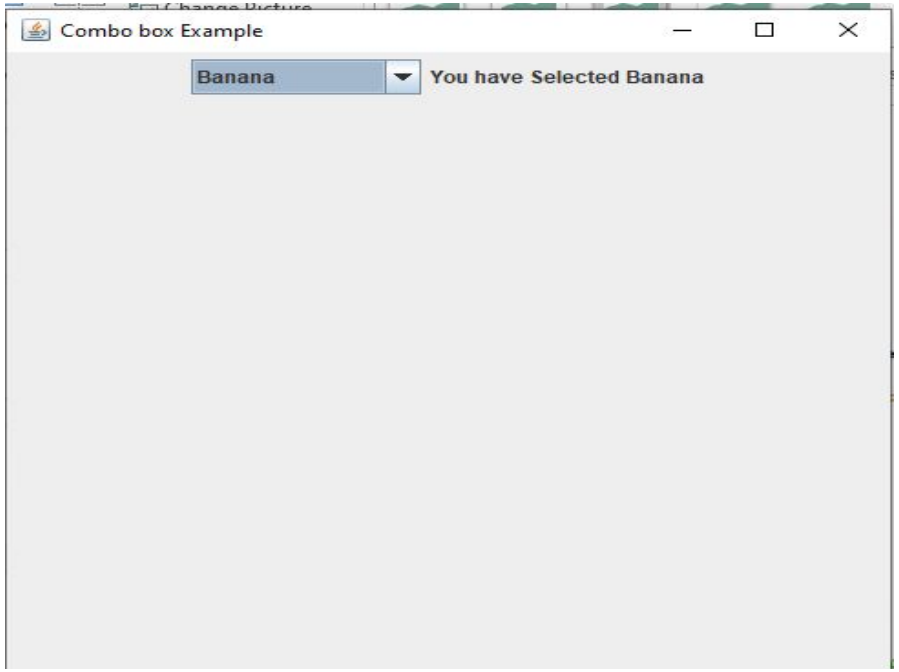
```
You have selected Banana
```

Activate Windows
Go to Settings to activate Win

Swing Components-JComboBox Example

```
public class ComboBoxExample extends JFrame implements ActionListener{
    JComboBox combo;
    JLabel lbl = new JLabel();
    public ComboBoxExample(){
        this.setTitle("Combo box Example");
        this.setSize(500, 500);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLayout(new FlowLayout());
        String[] fruitList = {"Choose an option","Apple","Banana","Mango"};
        combo = new JComboBox(fruitList);
        combo.addActionListener(this);
        this.add(combo);
        this.add(lbl);
        this.setVisible(true);
    }
   @Override
   public void actionPerformed(ActionEvent e) {
       if(e.getSource()==combo){
           if(combo.getSelectedItem().toString().equalsIgnoreCase("Choose an option")){
               lbl.setText("Choose a fruit");
           }else{
           lbl.setText("You have Selected "+ combo.getSelectedItem());
          }
      }
   }
}
```



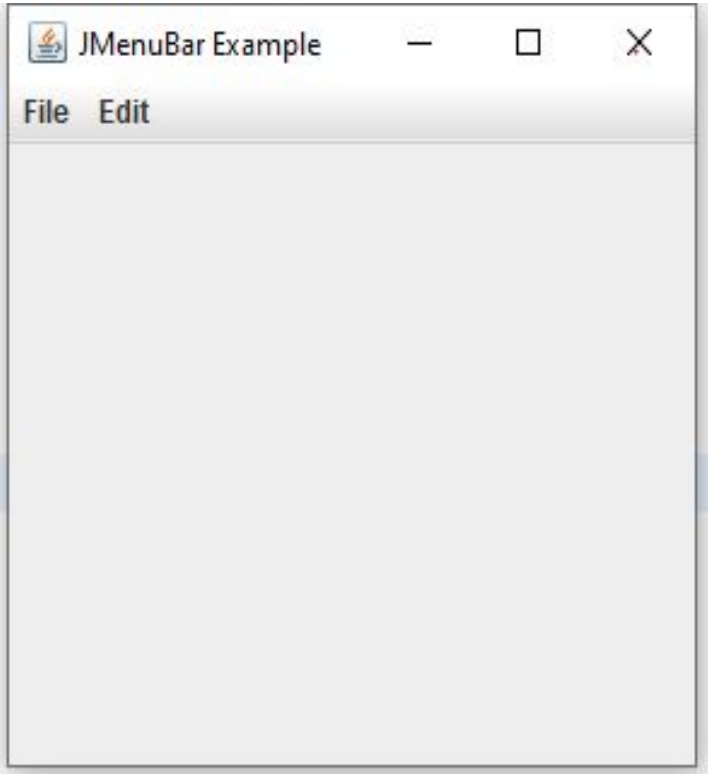


Swing Components-JMenuBar, JMenu and JMenuItem

- The JMenuBar class is used to display menubar on the window or frame. It may have several menus.
- The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.
- The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.
- In order to set the JMenuBar on frame we can use **setJMenuBar()** method.

Swing Components-JMenuBar and JMenu Example

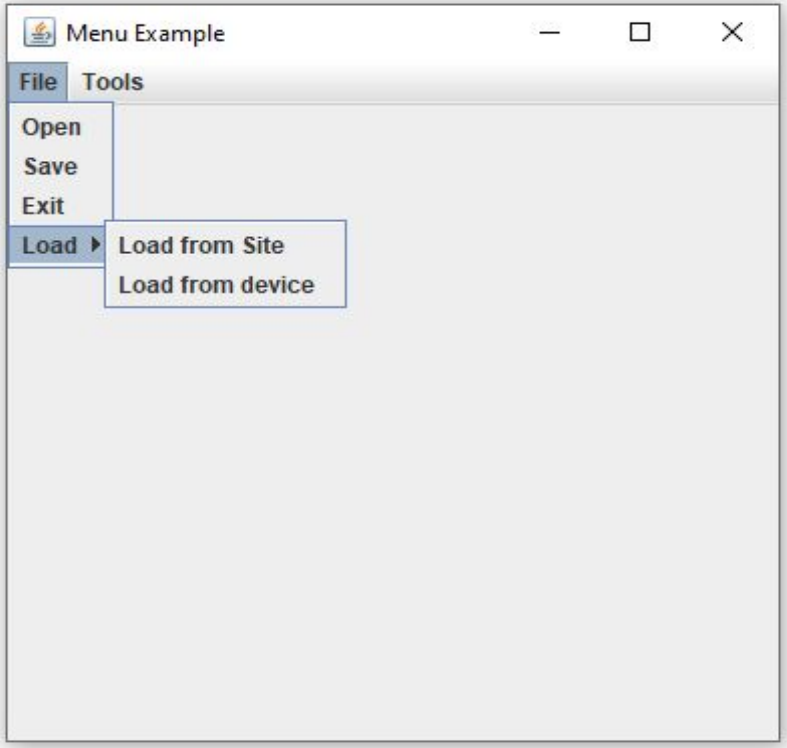
```java
public class JMenuBarExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JMenuBar Example");
        JMenuBar menubar = new JMenuBar();
        JMenu menu1 = new JMenu("File");
        JMenu menu2 = new JMenu("Edit");
        menubar.add(menu1);
        menubar.add(menu2);
        frame.setJMenuBar(menubar);
        frame.setSize(300,300);
        frame.setVisible(true);
        frame.setLayout(null);
    }
}
```



Swing Components-JMenuBar, JMenu and JMenuItem Example

JavaSwing.java | FirstSwing.java | DatabaseConn... | FrameExampl... | JCOmboBoxEx... | JMenuBarExam... | MenuExample...

```java
import javax.swing.*;

public class MenuExample2 {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Menu Example");
        JMenuBar menuBar=new JMenuBar();
        JMenu menu=new JMenu("File");
        JMenu menu2 = new JMenu("Tools");
        JMenu submenu=new JMenu("Load");
        JMenuItem i1=new JMenuItem("Open");
        JMenuItem i2=new JMenuItem("Save");
        JMenuItem i3=new JMenuItem("Exit");
        JMenuItem i4=new JMenuItem("Load from Site");
        JMenuItem i5=new JMenuItem("Load from device");
        JMenuItem i6 = new JMenuItem("Run");
        JMenuItem i7 = new JMenuItem("Debug");
        menu.add(i1);  menu.add(i2);   menu.add(i3);
        submenu.add(i4);  submenu.add(i5);
        menu2.add(i6);  menu2.add(i7);
        menu.add(submenu);
        menuBar.add(menu);          menuBar.add(menu2);
        frame.setJMenuBar(menuBar);
        frame.setSize(400,400);
        frame.setVisible(true);
    }
}
```

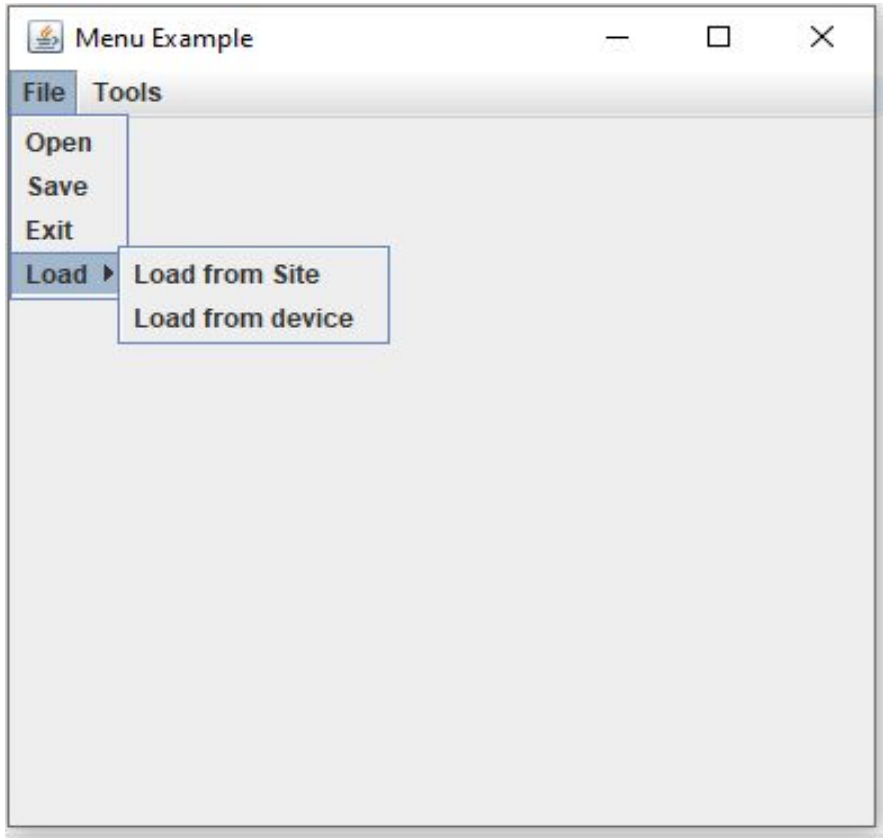


Swing Components-JMenuBar, JMenu and JMenuItem Example

- **Note: Same example with another methods**

```java
public class MenuExample2 extends JFrame{

    public MenuExample2() {
        this.setTitle("Menu Example");
        JMenuBar menuBar=new JMenuBar();
        JMenu menu=new JMenu("File");
        JMenu menu2 = new JMenu("Tools");
        JMenu submenu=new JMenu("Load");
        JMenuItem i1=new JMenuItem("Open");
        JMenuItem i2=new JMenuItem("Save");
        JMenuItem i3=new JMenuItem("Exit");
        JMenuItem i4=new JMenuItem("Load from Site");
        JMenuItem i5=new JMenuItem("Load from device");
        JMenuItem i6 = new JMenuItem("Run");
        JMenuItem i7 = new JMenuItem("Debug");
        menu.add(i1);  menu.add(i2);   menu.add(i3);
        submenu.add(i4);  submenu.add(i5);
        menu2.add(i6);  menu2.add(i7);
        menu.add(submenu);
        menuBar.add(menu);          menuBar.add(menu2);
        this.setJMenuBar(menuBar);
        this.setSize(400,400);
        this.setVisible(true);
    }
}
```

```java
public class MenuExample2MainClass {
    public static void main(String[] args) {
        new MenuExample2();
    }
}
```



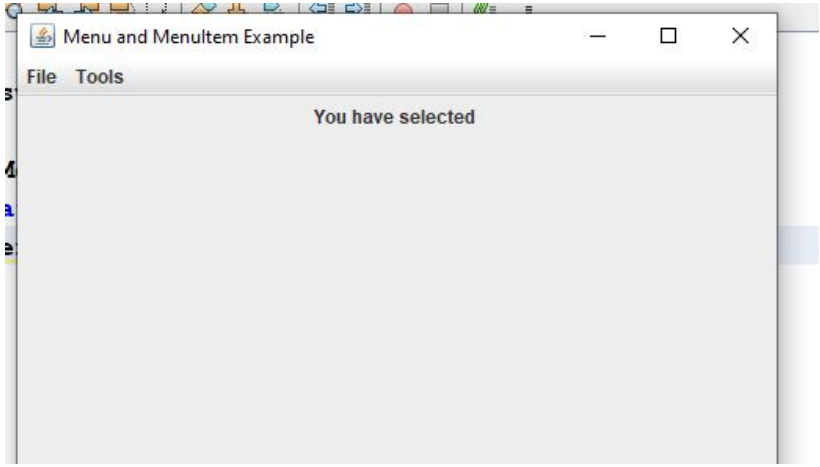
Swing Components-JMenuBar, JMenu and JMenuItem Example

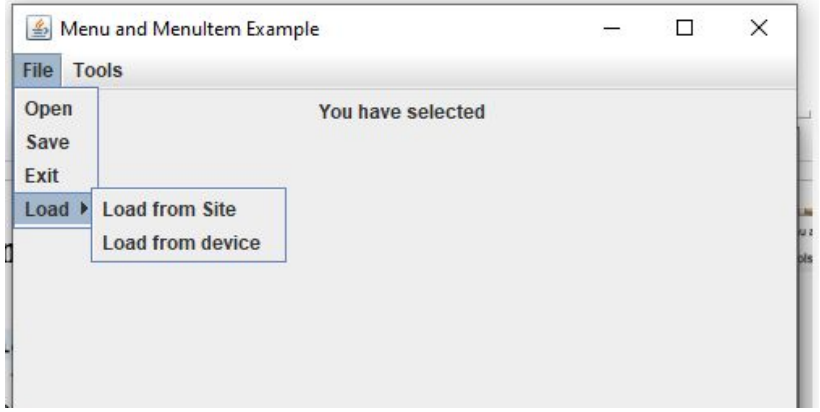
```java
public class MenuExample extends JFrame implements ActionListener{
    JMenuBar menuBar;
    JMenu menu,menu2, submenu;
    JMenuItem i1, i2, i3, i4, i5, i6, i7;
    JLabel label;
    public MenuExample(){
        this.setTitle("Menu and MenuItem Example");
        menuBar=new JMenuBar();
        menu=new JMenu("File");
        menu2 = new JMenu("Tools");
        submenu=new JMenu("Load");
        i1=new JMenuItem("Open");
        i2=new JMenuItem("Save");
        i3=new JMenuItem("Exit");
        i4=new JMenuItem("Load from Site");
        i5=new JMenuItem("Load from device");
        i6 = new JMenuItem("Run");
        i7 = new JMenuItem("Debug");
        i1.addActionListener(this);
```

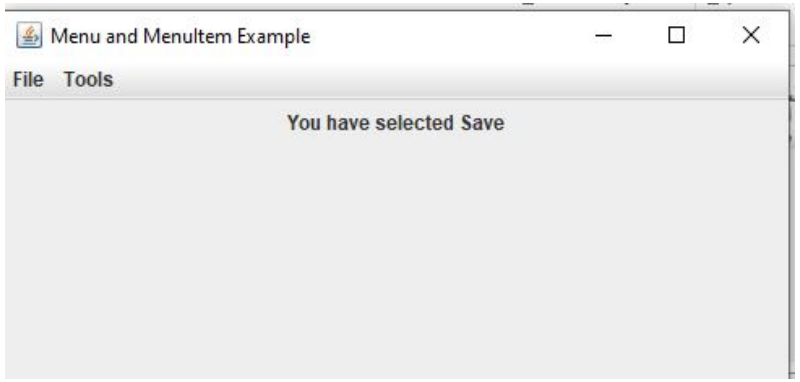
```java
        i2.addActionListener(this);
        menu.add(i1); menu.add(i2); menu.add(i3);
        submenu.add(i4); submenu.add(i5);
        menu2.add(i6);
        menu2.add(i7);
        menu.add(submenu);
        menuBar.add(menu);
        menuBar.add(menu2);
        this.setJMenuBar(menuBar);
        this.setSize(500,500);
        this.setLayout(new FlowLayout());
        this.setVisible(true);
        label= new JLabel("You have selected ");
        this.add(label);
    }
```

Swing Components-JMenuBar, JMenu and JMenuItem Example

```
    @Override
    public void actionPerformed(ActionEvent e) {
    if(e.getSource()==i1){
            label.setText("You have selected Open");
     }
     if(e.getSource()==i2){
            label.setText("You have selected Save");
     }
    }
}

public class MenuMain {
    public static void main(String[] args) {
        new MenuExample();
    }
}
```







Swing Components-JOptionPane

- The JOptionPane class is used to **provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box.**
- These dialog boxes are **used to display information or get input from the user.**
- Syntax for **show message dialog:**

```
JOptionPane.showMessageDialog(parentComponent, message, title, messageType, icon);
```

where **parentComponent** is generally set null which represents the parent component on which the dialog message is to be displayed, **message** contains a string value that is displayed on the dialog body, **title** contains a string value that appears as the title of the dialog box, messageType defines the type of message that the dialog defines, **icon** contains the icon to be displayed in the dialog that helps the user identify the kind of message that is being displayed

Swing Components-JOptionPane Example

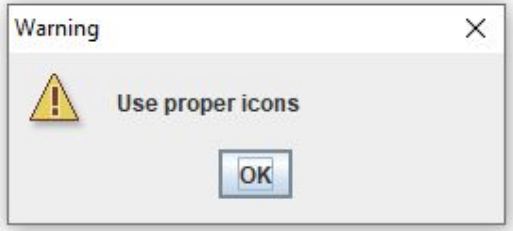
```java
import javax.swing.JOptionPane;

public class JOptionPaneExample {
    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null, "I am learning Swing", "Welcome",JOptionPane.PLAIN_MESSAGE);
    }
}
```

```java
public class JOptionPaneExample {
    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null, "I am learning Swing", "Welcome",JOptionPane.INFORMATION_MESSAGE);
    }
}
```

Swing Components-JOptionPane Example

```java
public class JOptionPaneExample {
    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null, "Are you understanding???", "Question",JOptionPane.QUESTION_MESSAGE);
    }
}
```

```java
public class JOptionPaneExample {
    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null, "Use proper icons", "Warning",JOptionPane.WARNING_MESSAGE);
    }
}
```



Swing Components-JOptionPane

❖ Syntax for **confirm dialog:**

JOptionPane.showConfirmDialog(parentComponent, message, title, messageType, icon);

❖ This function returns value 0 on yes, 1 on no, 2 on cancel and -1 on close

```
public class JOptionPaneExample {
    public static void main(String[] args) {
        int value=JOptionPane.showConfirmDialog(null, "Are you well???", "Health Test",JOptionPane.YES_NO_CANCEL_OPTION);
        System.out.println(value);
    }
}
```

Swing Components-JOptionPane

Input Dialog

❖ For input dialog, we use showinputDialog(). This method returns the value entered on the input dialog box.

```
public class JOptionPaneExample {
    public static void main(String[] args) {
        String value=JOptionPane.showInputDialog("What is your name???");
        System.out.println(value);
    }
}
```

JList

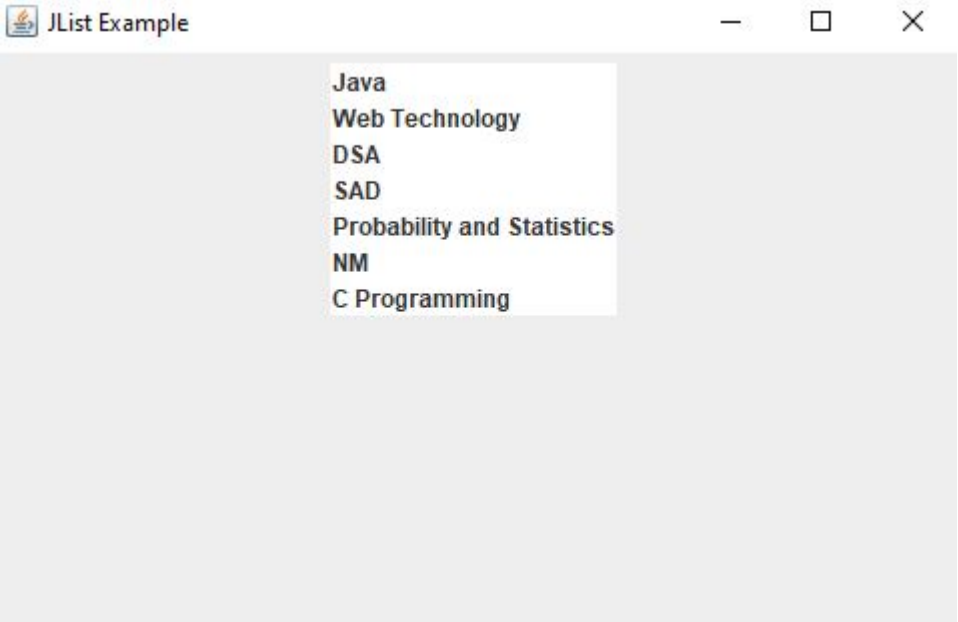
- JList is part of Java Swing package . JList is a component that displays a set of Objects and allows the user to select one or more items .
- JList inherits JComponent class. JList is a easy way to display an array of Vectors .
- Constructor for JList are :
  - **JList()**: creates an empty blank list
  - **JList(E[ ] lst)** : creates an new list with the elements of the array.
  - **JList(ListModel d)**: creates a new list with the specified List Model
  - **JList(Vector lst)** : creates a new list with the elements of the vector

Jlist-Commonly Used Methods

Methods	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return the data model that holds a list of items displayed by the JList component.
void setListData(Object[] listData)	It is used to create a read-only ListModel from an array of objects.

JList-Example

```java
public class JListExample extends JFrame {
    JList<String> subjectList;
    public JListExample(){
        this.setTitle("JList Example");
        this.setSize(500, 500);
        this.setLayout(new FlowLayout());

        String[] mySubjects ={"Java", "Web Technology",
            "DSA","SAD","Probability and Statistics","NM","C Programming"};
        subjectList = new JList<>(mySubjects);
        this.add(subjectList);
        this.setVisible(true);
    }
}
```



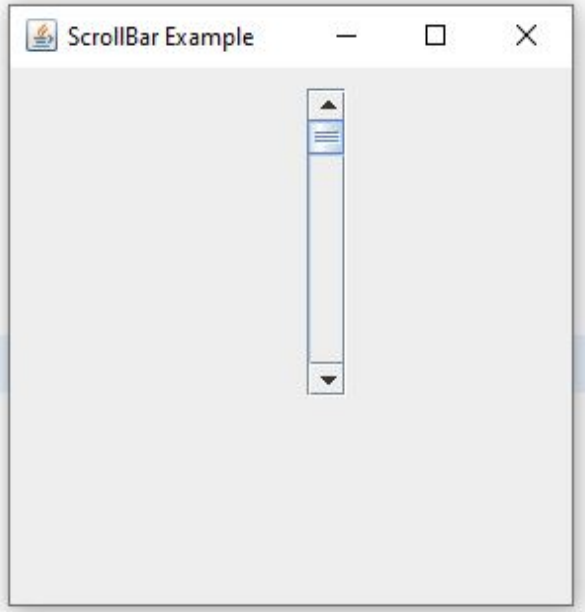
Swing Components-JScrollBar

- The object of JScrollbar class is used to add horizontal and vertical scrollbar.
- It is an implementation of a scrollbar. It inherits JComponent class.

Constructor	Description
JScrollBar()	Creates a vertical scrollbar with the initial values.
JScrollBar(int orientation)	Creates a scrollbar with the specified orientation and the initial values.
JScrollBar(int orientation, int value, int extent, int min, int max)	Creates a scrollbar with the specified orientation, value, extent, minimum, and maximum.

Swing Components-JScrollBar Example

```java
import javax.swing.JFrame;
import javax.swing.JScrollBar;

public class JScrollBarExample{
    public static void main(String[] args) {
        JFrame frame = new JFrame("ScrollBar Example");
        JScrollBar scroll = new JScrollBar();
        scroll.setBounds(150,10,20,150);
        frame.add(scroll);
        frame.setSize(300, 300);
        frame.setLayout(null);
        frame.setVisible(true);
    }
}
```



Swing Components-JScrollPane Example

- A JScrollPane is used to make scrollable view of a component.
- When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

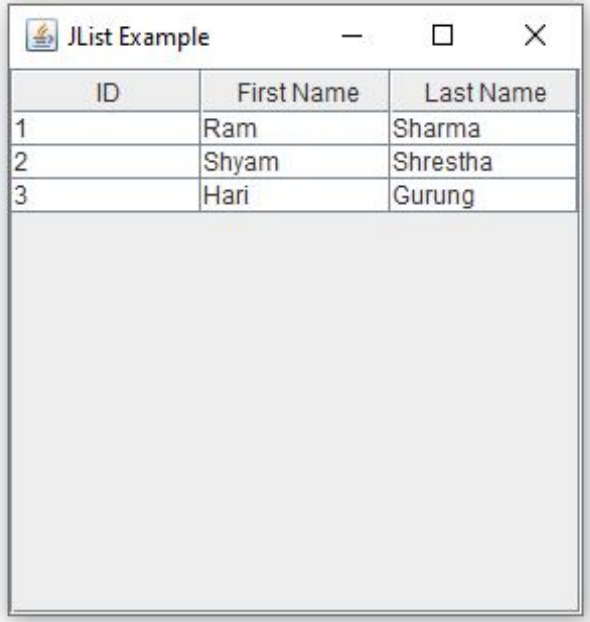
Constructor	Purpose
JScrollPane()	It creates a scroll pane. The Component parameter, when present, sets the scroll pane's client. The two int parameters, when present, set the vertical and horizontal scroll bar policies (respectively).
JScrollPane(Component)	
JScrollPane(int, int)	
JScrollPane(Component, int, int)	

Swing Components-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.
- Constructors in JTable:
  - **JTable():** A table is created with empty cells.
  - **JTable(int rows, int cols):** Creates a table of size rows * cols.
  - **JTable(Object[][] data, Object []Column):** A table is created with the specified name where []Column defines the column names.

Swing Components-JTable

```java
import javax.swing.*;

public class JTableExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JList Example");
        String[] cols = {"ID", "First Name", "Last Name"};
        String[][] data = {
                            {"1","Ram","Sharma"},
                            {"2","Shyam","Shrestha"},
                            {"3","Hari","Gurung"}
                          };
        JTable table = new JTable(data,cols);
        JScrollPane bar = new JScrollPane(table);
        frame.add(bar);
        frame.setVisible(true);
        frame.setSize(300, 300);

    }
```



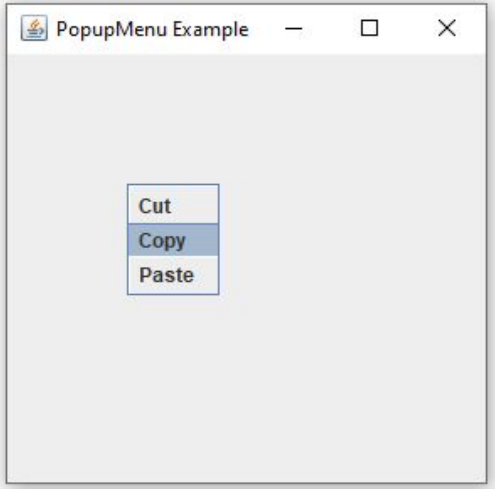
Java Swing Component-JPopUpMenu

- JPopupMenu generates a small window that pops up and displays a series of choices.
- JPopupMenu can be used to generate a small window at any position within the container.
- Constructor of the class are :
  - **JPopupMenu()** : creates a popup menu with empty name
  - **JPopupMenu(String name)** : creates a popup menu with specified title.

Java Swing Component-JPopUpMenu Example

```java
public class JPopUpMenuMain {
    public static void main(String[] args) {
        new JPopUpMenuExample();
    }
}
```

```java
public class JPopUpMenuExample implements MouseListener{
    JPopupMenu popupmenu;JFrame f;
    public JPopUpMenuExample() {
        f= new JFrame("PopupMenu Example");
        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(this);
        f.add(popupmenu);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    @Override
    public void mouseClicked(MouseEvent e) {
        if(e.getButton()==MouseEvent.BUTTON3) {
            popupmenu.show(f , e.getX(), e.getY());
        }
    }
    @Override
```



Java Swing Component-JSlider

❖ The Java JSlider class is used to create the slider. By using JSlider, a user can select a value from a specific range.

Constructor	Description
JSlider()	creates a slider with the initial value of 50 and range of 0 to 100.
JSlider(int orientation)	creates a slider with the specified orientation set by either JSlider.HORIZONTAL or JSlider.VERTICAL with the range 0 to 100 and initial value 50.
JSlider(int min, int max)	creates a horizontal slider using the given min and max.
JSlider(int min, int max, int value)	creates a horizontal slider using the given min, max and value.
JSlider(int orientation, int min, int max, int value)	creates a slider using the given orientation, min, max and value.

Java Swing Component-JSlider

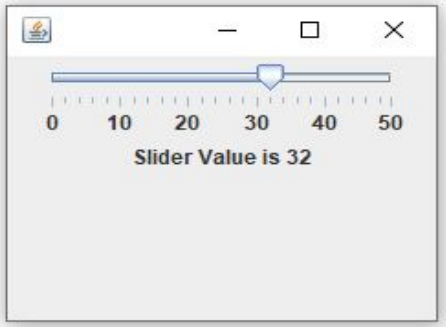
❖ Commonly used methods:

1. setOrientation(int n) : sets the orientation of the slider to the specified value
2. setValue(int v) : sets the value of the slider to the specified value
3. setPaintTicks(boolean b) : the boolean value determines whether the ticks are painted on the slider or not
4. setMajorTickSpacing(int n) : sets the spacing for major ticks .
5. setMinorTickSpacing(int n) : sets the spacing for minor ticks .
6. setMaximum(int m) : set the maximum value for the slider
7. setMinimum(int m) : sets the minimum value for the slider

Java Swing Component-JSlider Example

```java
public class SliderExampleMain {
    public static void main(String[] args) {
        new SliderExample();
    }
}
```

```java
public class SliderExample extends JFrame implements ChangeListener{
    JLabel lbl;JSlider slider;
    public SliderExample() {
        this.setTitle("Slider Example");
        slider = new JSlider(JSlider.HORIZONTAL, 0, 50, 25);
        lbl = new JLabel();
        slider.setMinorTickSpacing(2);
        slider.setMajorTickSpacing(10);
        slider.setPaintTicks(true);
        slider.setPaintLabels(true);
        JPanel panel=new JPanel();
        panel.add(slider);
        panel.add(lbl);
        this.add(panel);
        this.setVisible(true);
        this.setSize(250, 200);
        slider.addChangeListener(this);
    }
    @Override
    public void stateChanged(ChangeEvent e) {
        lbl.setText("Slider Value is "+ slider.getValue());
    }
}
```



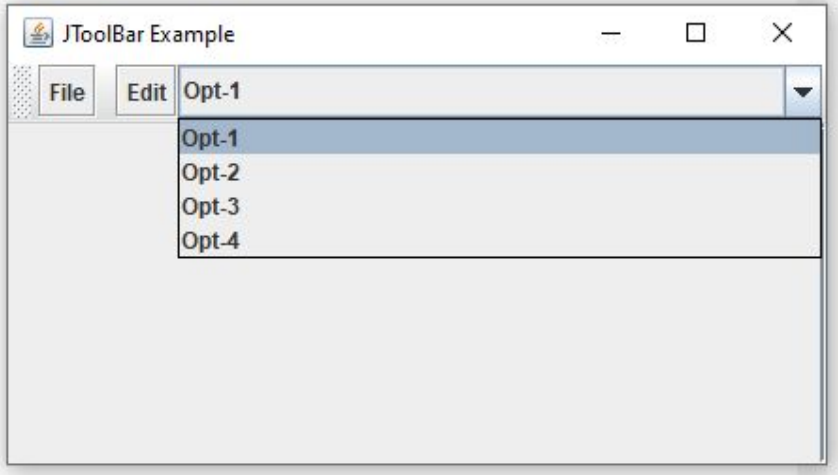
Java Swing Components-JToolBar

- JToolBar container allows us to group other components, usually buttons with icons in a row or column.
- JToolBar provides a component which is useful for displaying commonly used actions or controls.

Constructor	Description
JToolBar()	It creates a new tool bar; orientation defaults to HORIZONTAL.
JToolBar(int orientation)	It creates a new tool bar with the specified orientation.
JToolBar(String name)	It creates a new tool bar with the specified name.
JToolBar(String name, int orientation)	It creates a new tool bar with a specified name and orientation.

Java Swing Components-JToolBar Example

```java
import java.awt.BorderLayout;
import java.awt.Container;

import javax.swing.*;

public class JToolbarExample {
    public static void main(final String args[]) {
        JFrame myframe = new JFrame("JToolBar Example");
        myframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JToolBar toolbar = new JToolBar();
        toolbar.setRollover(true);
        JButton button = new JButton("File");
        toolbar.add(button);
        toolbar.addSeparator();
        toolbar.add(new JButton("Edit"));
        toolbar.add(new JComboBox(new String[] { "Opt-1", "Opt-2", "Opt-3", "Opt-4" }));
        Container contentPane = myframe.getContentPane();
        contentPane.add(toolbar, BorderLayout.NORTH);
        JTextArea textArea = new JTextArea();
        JScrollPane mypane = new JScrollPane(textArea);
        contentPane.add(mypane, BorderLayout.EAST);
        myframe.setSize(450, 250);
        myframe.setVisible(true);
    }
}
```



JTabbedPane

- The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon.
- It inherits JComponent class.
- Commonly used constructors are:

JTabbedPane()

//Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top

JTabbedPane(int tabPlacement)

//Creates an empty TabbedPane with a specified tab placement

JTabbedPane(int tabPlacement, int tabLayoutPolicy)

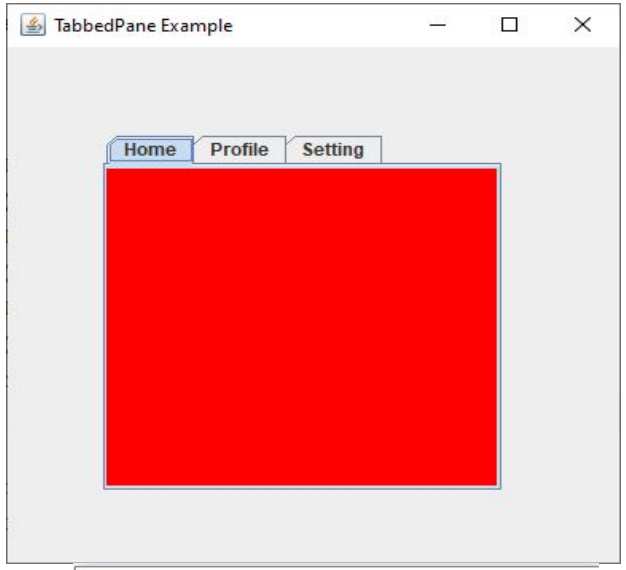
//Creates an empty TabbedPane with a specified tab placement and tab layout policy.

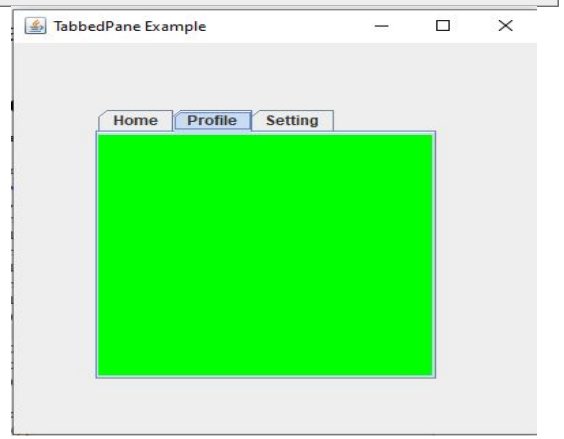
JTabbedPane

❖ After creating the object of JTabbedPane, we add components on the pane as:

tabbedPaneObject.add(String tabName, Component comp);

```java
public class TabbedPaneExample {
    public TabbedPaneExample() {
        JFrame frame = new JFrame("TabbedPane Example");
        JTabbedPane pane = new JTabbedPane();
        pane.setBounds(60, 60, 250, 250);
        JPanel pnl1 = new JPanel();
        pnl1.setBackground(Color.red);
        JPanel pnl2 = new JPanel();
        pnl2.setBackground(Color.green);
        JPanel pnl3 = new JPanel();
        pnl3.setBackground(Color.cyan);
        pane.add("Home", pnl1); pane.add("Profile",pnl2);pane.add("Setting",pnl3);
        frame.add(pane);        frame.setLayout(null);
        frame.setSize(400, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        new TabbedPaneExample();
    }
}
```





Layout Managers

- The Layout Managers are used to arrange components in a particular manner
- LayoutManager is an interface that is implemented by all the classes of layout managers.
- There are following classes that represents the layout managers:

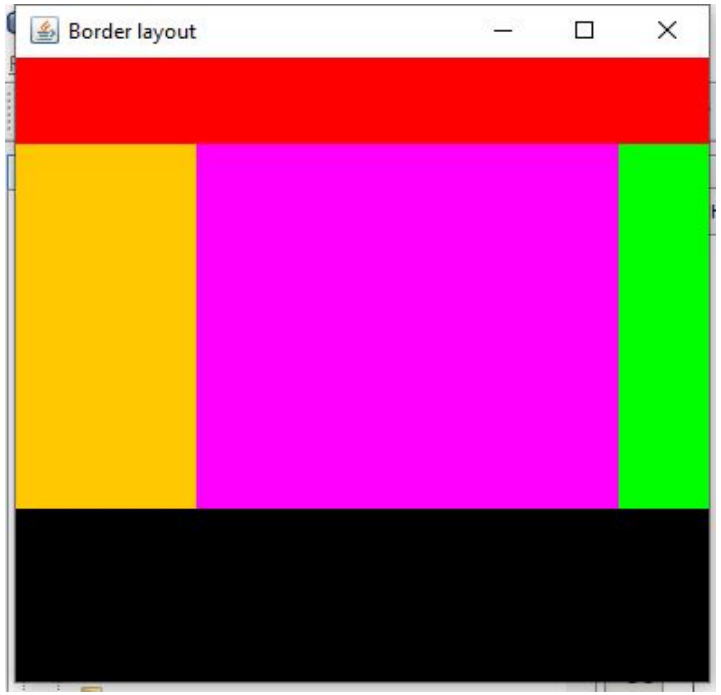
a. java.awt.BorderLayout
b. java.awt.FlowLayout
c. java.awt.GridLayout
d. java.awt.CardLayout
e. java.awt.GridBagLayout
f. javax.swing.BoxLayout
g. javax.swing.GroupLayout
h. javax.swing.ScrollPaneLayout
i. javax.swing.SpringLayout

Border Layout

- The BorderLayout is used to arrange the components in five regions: north, south, east, west and center.
- All extra space is placed in center area
- Each region (area) may contain one component only. It is the default layout of frame or window.
- BorderLayout provides five constants for each region: NORTH, SOUTH, EAST, WEST
- On frame resize, top and bottom will expand horizontally ; left and right will expand vertically

Border Layout-Example

```java
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class JavaBorderLayout {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Border layout");
        frame.setSize(400, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
        frame.setLayout(new BorderLayout());

        JPanel p1 = new JPanel();
        JPanel p2 = new JPanel();
        JPanel p3 = new JPanel();
        JPanel p4 = new JPanel();
        JPanel p5 = new JPanel();
        p1.setBackground(Color.red);
        p1.setPreferredSize(new Dimension(100,50)); //setting dimension of p1
        p2.setBackground(Color.BLACK);
        p2.setPreferredSize(new Dimension(50,100));
        p3.setBackground(Color.GREEN);
        p3.setPreferredSize(new Dimension(50,100));
        p4.setBackground(Color.ORANGE);
        p4.setPreferredSize(new Dimension(100,50));
        p5.setBackground(Color.MAGENTA);

        frame.add(p1, BorderLayout.NORTH);
        frame.add(p2, BorderLayout.SOUTH);
        frame.add(p3, BorderLayout.EAST);
        frame.add(p4, BorderLayout.WEST);
        frame.add(p5, BorderLayout.CENTER);
    }
}
```



Border Layout-Example

- In order to specify gaps between components, we need to pass horizontal and vertical gap (in pixels) in the BorderLayout Constructor.

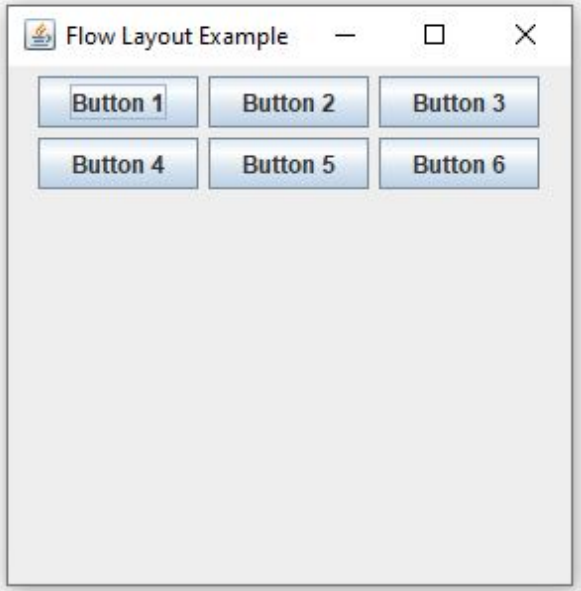
```
frame.setLayout(new BorderLayout(10,10));
```

Flow Layout

- The FlowLayout is used to arrange the components in a row, one aêer another (in a flow). It is the default layout of applet or panel
- FlowLayout provides five constants for each region: LEFT, RIGHT, CENTER, LEADING, TRAILING
- Constructors of FlowLayout class
  a. FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
  b. FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
  c. FlowLayout(int align, int hgap, int vgap): creates a flow layout with the given alignment and the given horizontal and vertical gap.

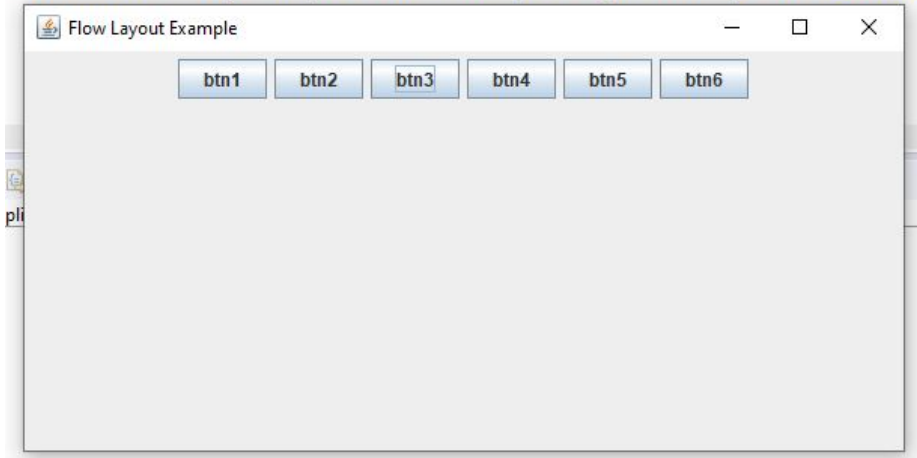
Flow Layout-Example

```
public class FlowLayoutExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Flow Layout Example");
        JButton btn1 = new JButton("Button 1");
        JButton btn2 = new JButton("Button 2");
        JButton btn3 = new JButton("Button 3");
        JButton btn4 = new JButton("Button 4");
        JButton btn5 = new JButton("Button 5");
        JButton btn6 = new JButton("Button 6");
        frame.add(btn1);frame.add(btn2);frame.add(btn3);
        frame.add(btn4);frame.add(btn5);frame.add(btn6);
        frame.setSize(300, 300);
        frame.setVisible(true);
        frame.setLayout(new FlowLayout());
    }
}
```



Flow Layout-Example

```
frame.setLayout(new FlowLayout(FlowLayout.CENTER));
```




```
frame.setLayout(new FlowLayout(FlowLayout.TRAILING));
```

```
frame.setLayout(new FlowLayout(FlowLayout.LEADING));
```

```
frame.setLayout(new FlowLayout(FlowLayout.LEFT));
```

```
frame.setLayout(new FlowLayout(FlowLayout.RIGHT));
```

Grid Layout

- The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.
- Constructors of GridLayout class

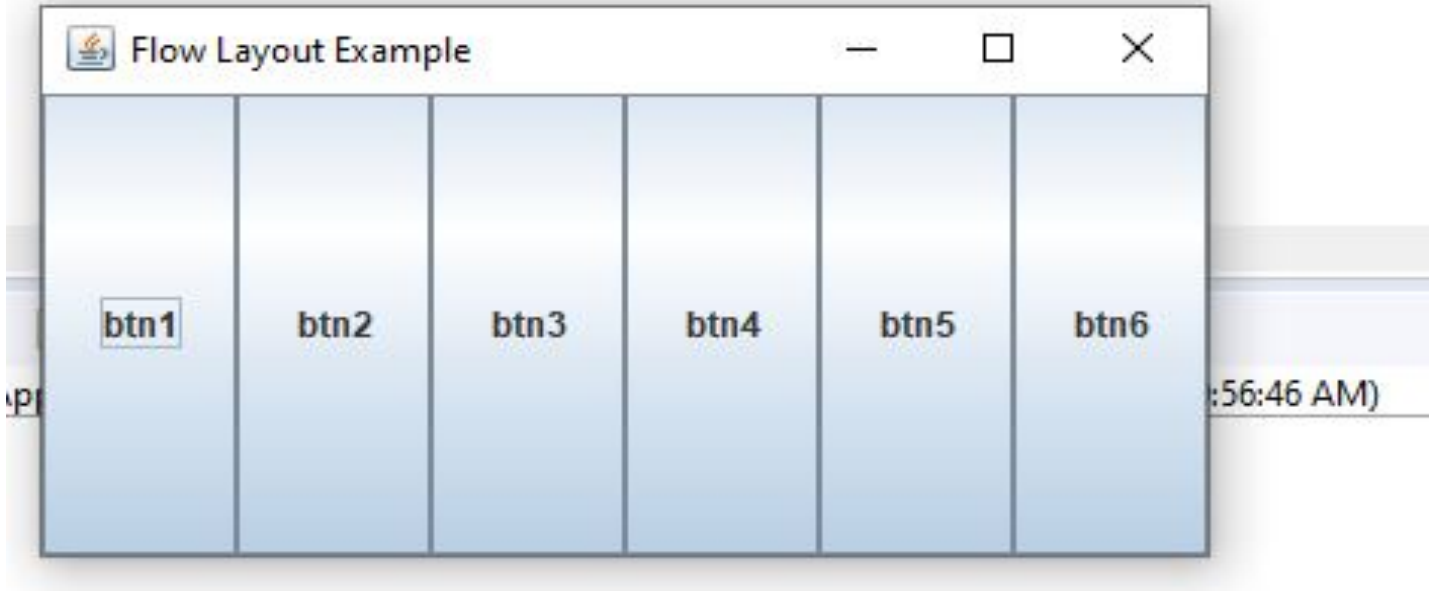
a. **GridLayout():** creates a grid layout with one column per component in a row.

b. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.

c. **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.

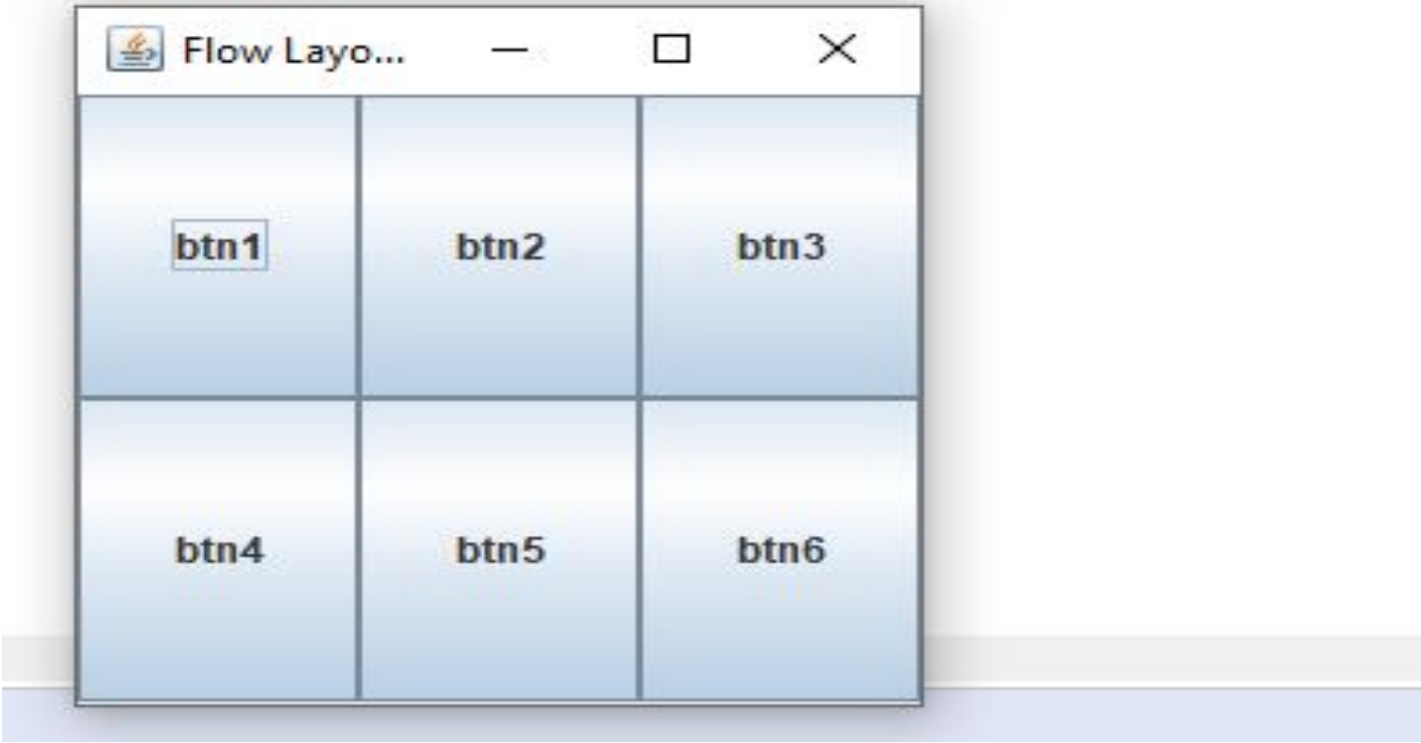
Grid Layout-Example

```
frame.setLayout(new GridLayout());
```



Grid Layout-Example

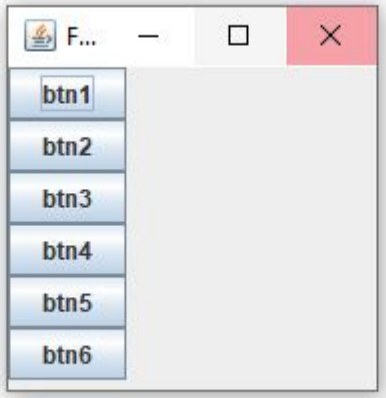
```
frame.setLayout(new GridLayout(2,3));
```



BoxLayout

- The **Java BoxLayout class** is used to arrange the components either vertically or horizontally.
- Constructor Available:
  - **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.

```
frame.setLayout(new BoxLayout(frame.getContentPane(), BoxLayout.Y_AXIS));
```



Displaying Information In a Component

- Displaying text in frame
- We can draw a message string directly onto a frame, but that is not considered good programming practice.
- In Java, frames are really **designed to be containers for components such as a menu bar and other user interface elements**.
- We normally draw on another component which we add to the frame.

Displaying Information In a Component

```java
public class DisplayingComponentExample extends JPanel{
    public static void main(String[] args) {
        //Create a Frame
                JFrame frame=new JFrame("Draw a Text");
                frame.setSize(250,250);
                frame.getContentPane().setBackground(Color.white);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                DisplayingComponentExample obj = new DisplayingComponentExample();
                frame.add(obj);
                frame.setVisible(true);
            }
            //Function to draw a text using drawString method
            @Override
            public void paintComponent(Graphics g)
            {
                g.setColor(Color.black);
                Font myFont=new Font("TimesRoman",Font.BOLD,30);
                g.setFont(myFont);
                g.drawString("Hello World",50,75);
            }
    }
```

Working with 2D Shapes

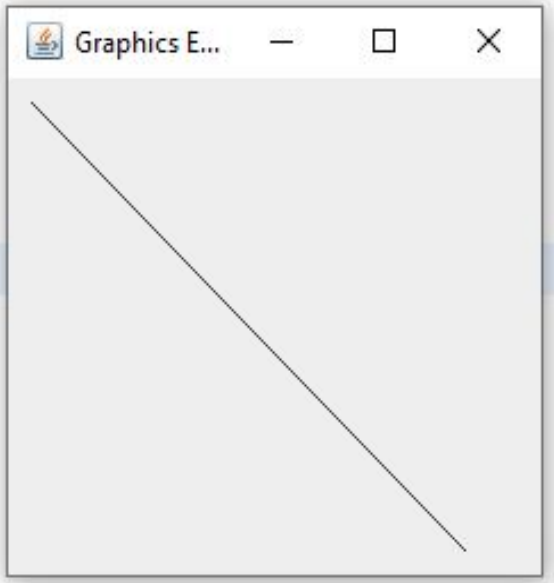
- java.awt.Graphics class provides many methods for graphics programming.
- The Graphics class is the abstract super class for all graphics contexts which allow an application to draw onto components that can be realized on various devices, or onto off-screen images as well.
- Starting with Java 1.0 this class has methods to draw graphical components.
- Java 1.2 introduced Java 2D library which implements a powerful set of graphical operations.
- In order to draw shapes in the Java 2D library, we need to obtain the object of the **Graphics2D class** which is the sub class of Graphics class

Working with 2D Shapes

- The Java 2D library organizes geometric shapes(Line2D, rectangle2D, Ellipse2D) in an object-oriented fashion.
- These classes all implement the Shape interface.
- To draw a shape, we first create an object of a class that implements the Shape interface and call the draw method of the Graphics2D class.

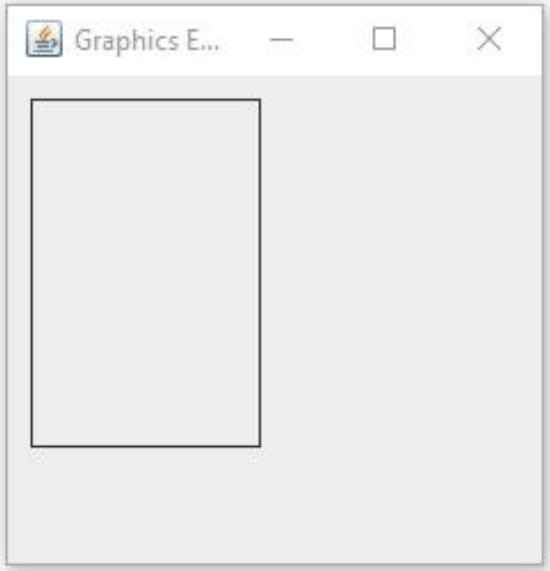
Working with 2D Shapes-Example

```
public class GraphicsExample extends JPanel{
    public static void main(String[] args) {
        //Create a Frame
                JFrame frame=new JFrame("Graphics Example");
                frame.setSize(250,250);
                frame.getContentPane().setBackground(Color.white);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                GraphicsExample obj = new GraphicsExample();
                frame.add(obj);
                frame.setVisible(true);
            }
            //Function to draw a text using drawString method
            @Override
            public void paintComponent(Graphics g)
            {
                Graphics2D g2d = (Graphics2D) g;
                g2d.drawLine(10, 10, 200, 200);
            }
}
```



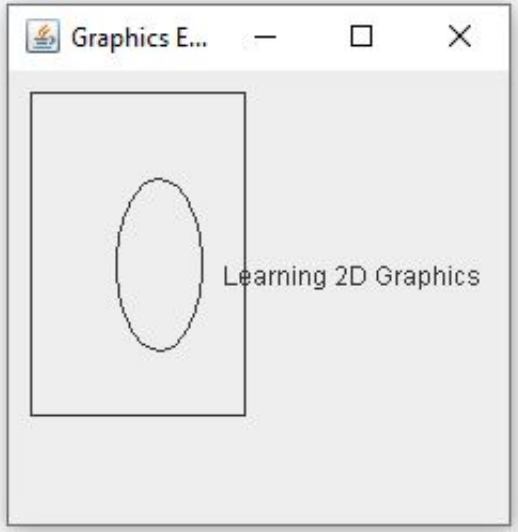
Working with 2D Shapes-Example

```
public class GraphicsExample extends JPanel{
    public static void main(String[] args) {
        //Create a Frame
                JFrame frame=new JFrame("Graphics Example");
                frame.setSize(250,250);
                frame.getContentPane().setBackground(Color.white);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                GraphicsExample obj = new GraphicsExample();
                frame.add(obj);
                frame.setVisible(true);
            }
            //Function to draw a text using drawString method
            @Override
            public void paintComponent(Graphics g)
            {
                Graphics2D g2d = (Graphics2D) g;
                g2d.drawRect(10, 10, 100, 150);
            }
}
```



Working with 2D Shapes-Example

```java
public class GraphicsExample extends JPanel{
    public static void main(String[] args) {
        //Create a Frame
                JFrame frame=new JFrame("Graphics Example");
                frame.setSize(250,250);
                frame.getContentPane().setBackground(Color.white);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                GraphicsExample obj = new GraphicsExample();
                frame.add(obj);
                frame.setVisible(true);
            }
            //Function to draw a text using drawString method
            @Override
            public void paintComponent(Graphics g)
            {
                Graphics2D g2d = (Graphics2D) g;
                g2d.drawRect(10, 10, 100, 150);
                g2d.drawOval(50, 50, 40, 80);
                g2d.drawString("Learning 2D Graphics", 100, 100);
            }
}
```



Working with 2D Shapes-Example

```java
public class GraphicsExample extends JPanel{
    public static void main(String[] args) {
        //Create a Frame
                JFrame frame=new JFrame("Smile Example");
                frame.setSize(250,250);
                frame.getContentPane().setBackground(Color.white);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                GraphicsExample obj = new GraphicsExample();
                frame.add(obj);
                frame.setVisible(true);
            }
            //Function to draw a text using drawString method
            @Override
            public void paintComponent(Graphics g)
            {
                Graphics2D g2d = (Graphics2D) g;
                g2d.setColor(Color.YELLOW);
                  g2d.fillOval(10, 10, 200, 200);
                  // draw Eyes
                  g2d.setColor(Color.BLACK);
                  g2d.fillOval(55, 65, 30, 30);
                  g2d.fillOval(135, 65, 30, 30);
                  // draw Mouth
                  g2d.fillOval(50, 110, 120, 60);
                  // adding smile
                  g2d.setColor(Color.YELLOW);
                  g2d.fillRect(50, 110, 120, 30);
                  g2d.fillOval(50, 120, 120, 40);
            }
}
```

Using Color

- The setPaint() method of the Graphics2D class lets us select a color that is used for all subsequent drawing operations on the graphics context.

```
g.setPaint(Color.blue);
g.drawString("Advanced Java Programming", 35, 100);
```

- We can also fill the interiors of a closed shapes such as rectangles or ellipses with a color for which we can use the fill method.

```
public void paintComponent(Graphics g)
{
g.setColor(Color.GREEN);
g.fillRect(10, 10, 100, 150);
}
```

Using Color

- The java.awt.Color class offers predefined constants for the following 13 standard colors:
  - BLACK,BLUE,CYAN,DARK_GRAY, GREEN, LIGHT_GRAY, MAGNETA, ORANGE, PINK, RED, WHITE, YELLOW
- Also we can use the color constructor for redness, blueness and greenness.

```
g.setColor(new Color(255, 0, 0));
```

- In order to set the background color, we can use the setBackground() method.

```
Component.setBackground(Color.RED);
```

Using Special Fonts for Text

- To find out the fonts that are available on a particular computer, we call the getAvailableFontFamilyNames() methods of the GraphicsEnvironment class.

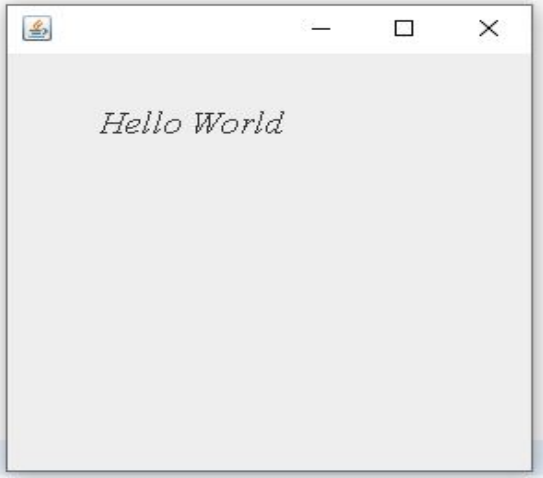
  GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamily Names();

```
public class FontExample {
    public static void main(String[] args) {
        String[] availableFontFamily =
                GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames();
        for(int i=0;i<availableFontFamily.length;i++) {
            System.out.println(availableFontFamily[i]);
        }
    }
}
```

Using Special Fonts for Text

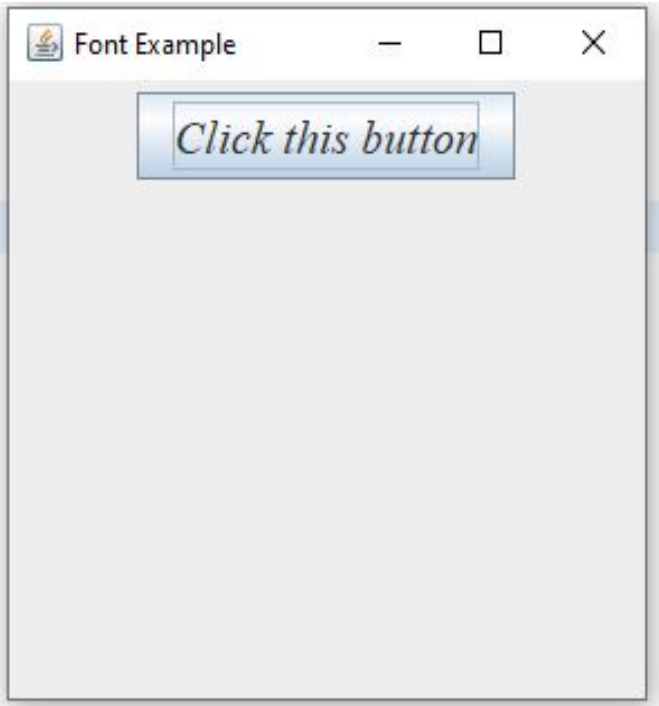
- The Font class defines three static style identifiers: PLAIN, BOLD, and ITALIC.
- We can use these values on all fonts. The point size determines the size of the font on a display.
- If a given point size isn't available, Font substitutes a default size.

```
public class FontExample extends JPanel{
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        Font font = new Font(Font.SERIF, Font.ITALIC, 20);
        FontExample ex = new FontExample();
        ex.setFont(font);
        frame.add(ex);
        frame.setSize(300, 300);
        frame.setVisible(true);
        }
        @Override
        public void paintComponent(Graphics g) {
            g.drawString("Hello World", 50, 50);
    }
}
```



Using Special Fonts for Text-Example

```java
public class FontExample{
    public static void main(String[] args) {
            JFrame frame = new JFrame();
            frame.setTitle("Font Example");
            JButton btn = new JButton("Click this button");
            Font font = new Font(Font.SERIF, Font.ITALIC, 20);
            btn.setFont(font);
            frame.add(btn);
            frame.setSize(300,300);
            frame.setVisible(true);
            frame.setLayout(new FlowLayout());
        }
}
```

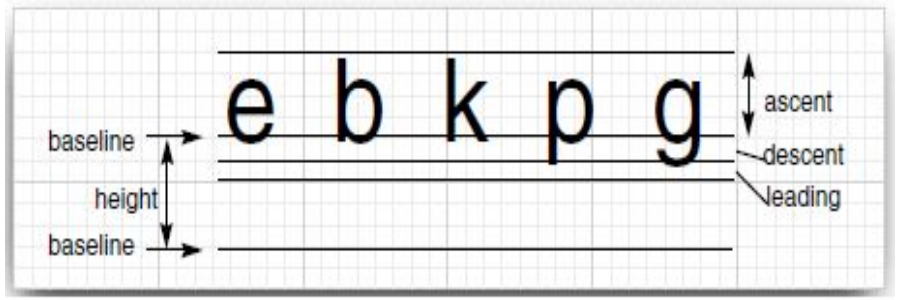


Using Special Fonts for Text-Font Metrics

- To get detailed size and spacing information for text rendered in a font, we can ask for a java.awt.font.LineMetrics object.
- Different systems will have different real fonts available; the available fonts may not match the font you request.
- Furthermore, the measurements of different characters within a single font may be different, especially in multilingual text.
- Thus, a LineMetrics object presents information about a particular set of text in a particular font on a particular system, not general information about a font.

Using Special Fonts for Text-Typesetting

- The **baseline** is the imaginary line where, for example, the bottom of a character like “e” rests.
- The **ascent** is the distance from the baseline to the top of an ascender, which is the upper part of a letter.
- The **descent** is the distance from the baseline to a descender, which is the lower portion of a letter.
- **Leading** is the space between the descent of one line and the ascent of the next line.
- The **height** of a font is the distance between successive baselines, which is the same as descent + leading + ascent.



Using Special Fonts for Text-Typesetting

```java
public class FontExample extends JPanel{
            static Font font;
    public static void main(String[] args) {
            JFrame frame = new JFrame();
            frame.setTitle("Font Example");
            FontExample exmp = new FontExample();
            font = new Font(Font.SERIF, Font.BOLD, 20);
            exmp.setFont(font);
            frame.add(exmp);
            frame.setSize(300,300);
            frame.setVisible(true);
        }
    @Override
    public void paintComponent(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        //g.drawString("I am learning Java", 75, 100);
        FontRenderContext context = g2d.getFontRenderContext();
        Rectangle2D bounds =  font.getStringBounds("I am learning Java", context);
          // (x,y) = top left corner of text
          double x = (getWidth() - bounds.getWidth()) / 2;
          double y = (getHeight() - bounds.getHeight()) / 2;
          // add ascent to y to reach the baseline
          double ascent = -bounds.getY();
          double baseY = y + ascent;
          g2d.drawString("I am learning Java", (int) x, (int) baseY);
          Rectangle2D rect2D =new Rectangle2D.Double(x, y, bounds.getWidth(), bounds.getHeight());
          g2d.draw(rect2D);
    }
}
```

Activate Windows

Displaying Image

- We can display image using the Graphics class.
- There are various ways of reading images such as ImageIcon, ToolKit, BufferedReader, etc.

```java
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class DisplayingImage extends JPanel{
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setTitle("Drawing Image");
        DisplayingImage di = new DisplayingImage();
        frame.add(di);
        frame.setSize(300,300);
        frame.setVisible(true);


    }
    @Override
    public void paintComponent(Graphics g) {
        Image img = new ImageIcon("C:\\Users\\Sudar\\eclipse-workspace\\JavaSwing\\src\\images.jpg").getImage();
        g.drawImage(img, 50,50,this);
    }
}
```

Displaying Image-Example

```
public class DisplayingImageExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Diaplaying Image");
        ImageIcon ic = new ImageIcon("C:\\Users\\Sudar\\eclipse-workspace\\JavaSwing\\src\\images.jpg");
        frame.add(new JLabel(ic));
        frame.setSize(300, 300);
        frame.setVisible(true);
        frame.pack();
        //pack() sizes the frame so that all its contents are at or above their preferred sizes.
    }
}
```

Displaying Image using ImageIO class

- ❖ ImageIO class can be used to display an image from the system on a frame.
- ❖ By using ImageIO.read(File file), the image is read from the system.

```java
public class ShowImageExample{
    public static void main(String[] args) throws IOException {
        File file = new File("C:\\Users\\Sudar\\eclipse-workspace\\JavaSwing\\src\\images.jpg");
        BufferedImage bufferedImage = ImageIO.read(file);
        ImageIcon imageIcon = new ImageIcon(bufferedImage);
        JFrame jFrame = new JFrame("Displaying Image");
        jFrame.setLayout(new FlowLayout());
        jFrame.setSize(300, 300);
        JLabel jLabel = new JLabel();
        jLabel.setIcon(imageIcon);
        jFrame.add(jLabel);
        jFrame.setVisible(true);
        jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Event Handling in Java Swing

- Change in the state of an object is known as **Event**, i.e., event describes the change in the state of the source.
- Events are generated as a result of user interaction with the graphical user interface components.
- For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from the list, and scrolling the page are the activities that causes an event to occur.

Event Handling in Java Swing

1. **Event:** An event is a signal to the program that something has happened. It can be triggered by typing in a text field, selecting an item from the menu etc.
2. **Event handler:** The code that performs a task in response to an event. is called event handler.
3. **Event handling:** It is process of responding to events that can occur at any time during execution of a program.
4. **Event Source:** It is an object that generates the event(s). Usually the event source is a button or the other component that the user can click but any Swing component can be an event source.
5. **Event Listener:** It is an object that watch for (i.e. listen for) events and handles them when they occur.
6. **Listener interface**: It is an interface which contains methods that the listener must implement and the source of the event invokes when the event occurs.

Java Event classes and Listener interfaces

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

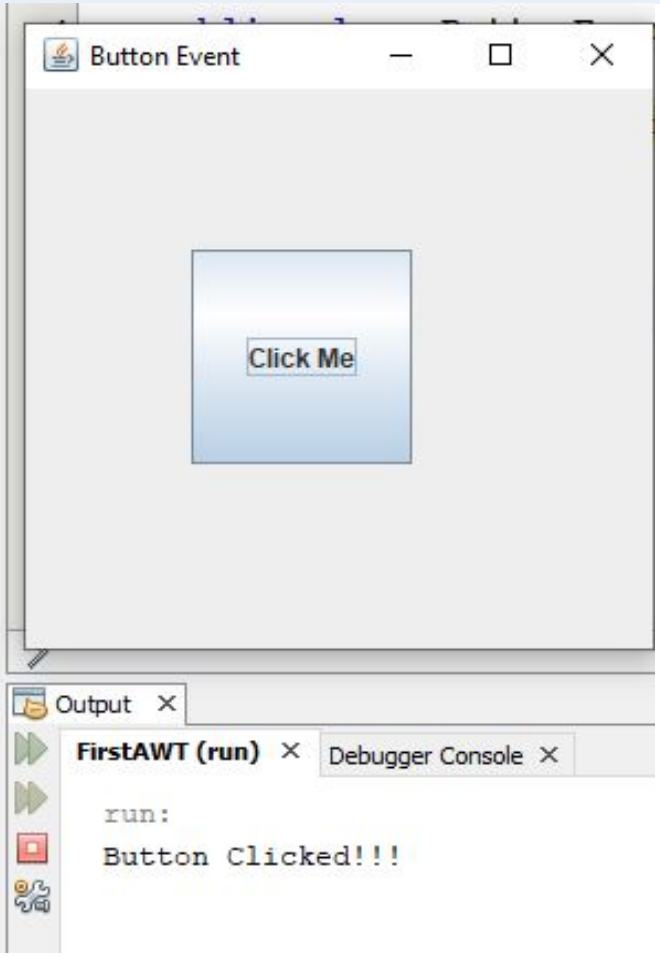
Steps to perform Event Handling

- We must first register the component with the Listener.
- For registering the component with the Listener, many classes provide the registration methods.
- For example:
  a. **Button**: public void addActionListener(ActionListener a){}
  b. **MenuItem**: public void addActionListener(ActionListener a){}
  c. **TextField**: public void addActionListener(ActionListener a){}
     public void addTextListener(TextListener a){}
  d. **TextArea**: public void addTextListener(TextListener a){}
  e. **Checkbox**: public void addItemListener(ItemListener a){}

ActionEvent Example

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

public class JButtonEvent implements ActionListener{
    public JButtonEvent(){
        JFrame frame = new JFrame("Button Event");
        frame.setSize(300,300);
        frame.setVisible(true);
        frame.setLayout(null);

        JButton btn = new JButton("Click Me");
        btn.setBounds(75, 75, 100, 100);
        btn.addActionListener(this);
        frame.add(btn);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button Clicked!!!");
    }
}
```

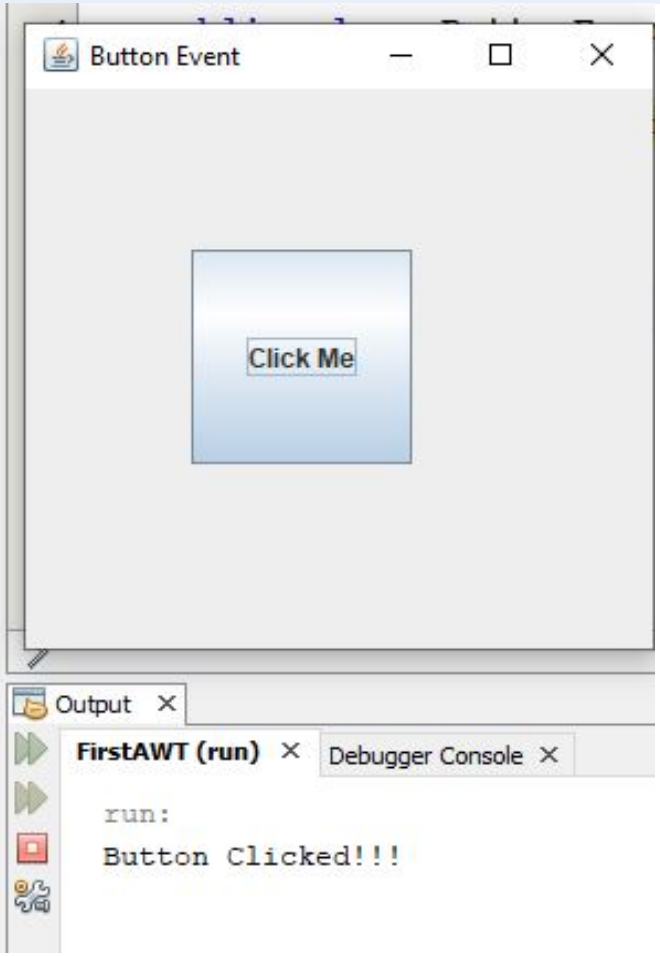
```java
public class ButtonEventMain {
    public static void main(String[] args) {
        new JButtonEvent();
    }
}
```



ActionEvent Example

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

public class JButtonEvent extends JFrame implements ActionListener{
    public JButtonEvent(){
        new JFrame("Button Event");
        this.setSize(300,300);
        this.setVisible(true);
        this.setLayout(null);

        JButton btn = new JButton("Click Me");
        btn.setBounds(75, 75, 100, 100);
        btn.addActionListener(this);
        this.add(btn);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button Clicked!!!");
    }
}
```

```java
public class ButtonEventMain {
    public static void main(String[] args) {
        new JButtonEvent();
    }
}
```



Mouse Event Handling

- The class which processes the MouseEvent should implement MouseListener Interface.
- 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.
- Methods:

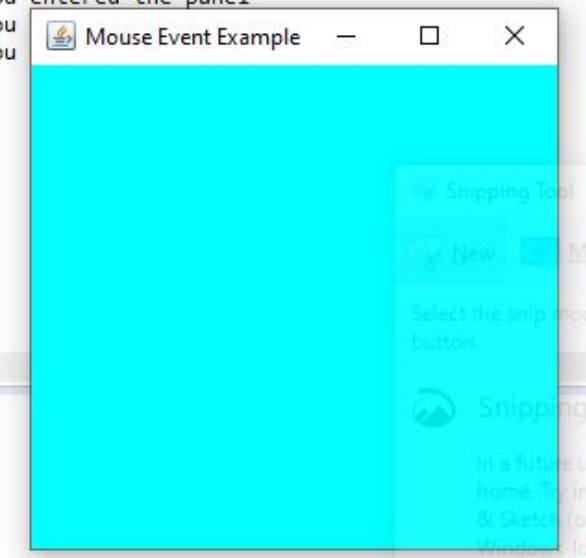
a. public abstract void mouseClicked(MouseEvent e);
b. public abstract void mouseEntered(MouseEvent e);
c. public abstract void mouseExited(MouseEvent e);
d. public abstract void mousePressed(MouseEvent e);
e. public abstract void mouseReleased(MouseEvent e);

MouseEvent Example

```java
public class MouseListenerMain {
    public static void main(String[] args) {
        new MouseListenerExample();
    }
}
```

```java
import java.awt.Color;
import java.awt.event.MouseListener;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class MouseListenerExample extends JFrame implements MouseListener{

    public MouseListenerExample() {
        this.setTitle("Mouse Event Example");
        this.setSize(300, 300);
        JPanel pnl = new JPanel();
        pnl.setBackground(Color.CYAN);
        pnl.addMouseListener(this);
        this.add(pnl);
        this.setVisible(true);
    }

    @Override
    public void mouseClicked(java.awt.event.MouseEvent e) {
        System.out.println("Panel is clicked");
    }
    @Override
    public void mouseEntered(java.awt.event.MouseEvent e) {
        System.out.println("You entered the panel");
    }
    @Override
    public void mouseExited(java.awt.event.MouseEvent e) {
        System.out.println("You left from the panel");
    }
    @Override
    public void mousePressed(java.awt.event.MouseEvent e) {
        // TODO Auto-generated method stub
    }
    @Override
    public void mouseReleased(java.awt.event.MouseEvent e) {
        // TODO Auto-generated method stub
    }
}
```

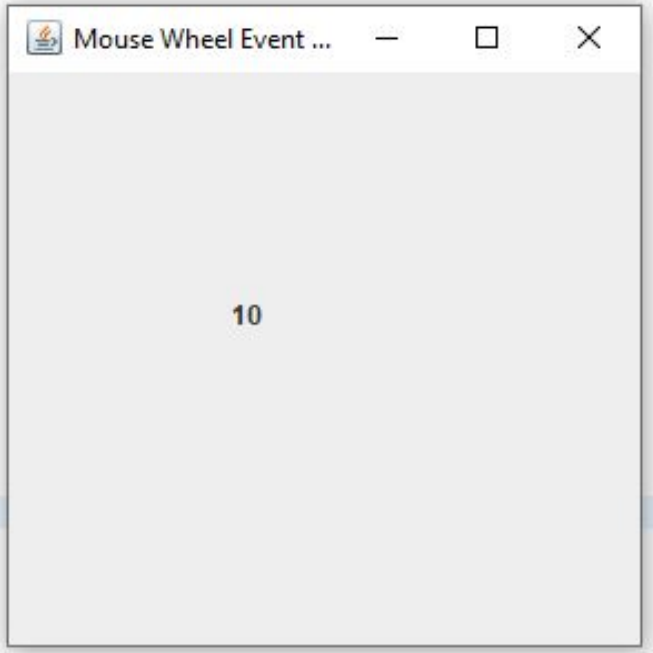
```
MouseListenerMain [Java Application] C:\Program Files\Java\jre1.8.0_74\bin\javaw.exe (Apr 8, 2023, 10:12:
You entered the panel
You left from the panel
You entered the panel
You left from the panel
You entered the panel
Panel is clicked
Panel is clicked
You left from the panel
You entered the panel
Panel is clicked
You left from the panel
You entered the panel
You left from the panel
You entered the panel
You left from the panel
You entered the panel
You
You
```



MouseWheelEvent Example

```java
public class MouseWheelExampleMain {
    public static void main(String[] args) {
        new MouseWheelExample();
    }
}
```

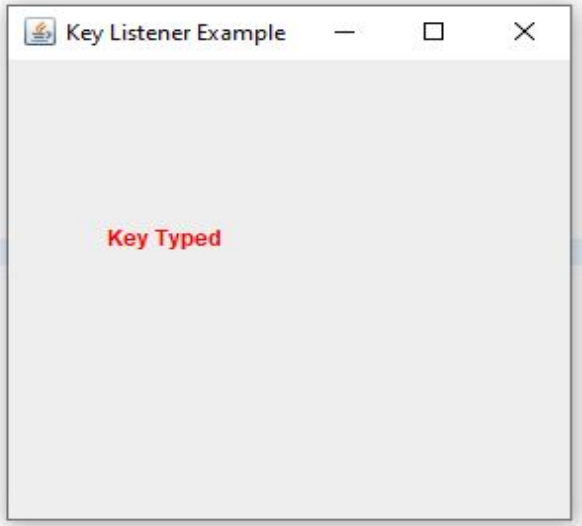
```java
public class MouseWheelExample extends JFrame implements MouseWheelListener{
    JLabel lbl;
    int count=0;
    public  MouseWheelExample() {
        lbl = new JLabel();
        this.setTitle("Mouse Wheel Event Example");
        this.addMouseWheelListener(this);
        this.add(lbl);
        lbl.setBounds(100, 100, 20, 20);
        this.setLayout(null);
        this.setSize(300, 300);
        this.setVisible(true);
    }

    @Override
    public void mouseWheelMoved(MouseWheelEvent e) {

        if(e.getWheelRotation()==1) { //returns value 1 and -1
            count++;
        }else {
            if(count>0) {
                count--;
            }

        }
        lbl.setText(String.valueOf(count));
    }

}
```



Key Event Handling

- 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, and it has three methods.

```java
public class KeyListenerExample extends JFrame implements KeyListener{
        JLabel label;
    public KeyListenerExample() {
        this.setTitle("Key Listener Example");
        this.setSize(300,300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLayout(null);
        label = new JLabel();
        label.setBounds(50, 50, 100, 100);
        label.setForeground(Color.red);
        label.setBackground(Color.red);
        this.add(label);
        this.setVisible(true);
        this.addKeyListener(this);
    }
    @Override
    public void keyTyped(KeyEvent e) {
        label.setText("Key Typed");
    }

    @Override
    public void keyPressed(KeyEvent e) {
        label.setText("Key Pressed");
     }

    @Override
    public void keyReleased(KeyEvent e) {
        label.setText("Key Released");
    }

}
```



Window Event

- For Window Event, we use WindowListener Interface.
- The Java WindowListener is notified whenever you change the state of window. It is notified against WindowEvent.

Constructor	Description
WindowEvent(Window source, int id)	Constructs a WindowEvent object.
WindowEvent(Window source, int id, int oldState, int newState)	Constructs a WindowEvent object with the specified previous and new window states.
WindowEvent(Window source, int id, Window opposite)	Constructs a WindowEvent object with the specified opposite Window.
WindowEvent(Window source, int id, Window opposite, int oldState, int newState)	Constructs a WindowEvent object.

Window Event

```java
public class WindowEventExample extends JFrame implements WindowListener{
    public WindowEventExample() {
        this.setTitle("Window Event Example");
        this.setSize(300, 300);
        this.addWindowListener(this);
        this.setVisible(true);
    }
    @Override
    public void windowActivated(WindowEvent e) {
        System.out.println("Window is activated");
    }
    @Override
    public void windowClosed(WindowEvent e) {
        System.out.println("Window is closed");
    }
    @Override
    public void windowClosing(WindowEvent e) {
        System.out.println("Window is closing");
    }
    @Override
    public void windowDeactivated(WindowEvent e) {
        // TODO Auto-generated method stub

    }
    @Override
    public void windowDeiconified(WindowEvent e) {
        // TODO Auto-generated method stub
    }
    @Override
    public void windowIconified(WindowEvent e) {
        // TODO Auto-generated method stub
    }
    @Override
    public void windowOpened(WindowEvent e) {
        // TODO Auto-generated method stub
    }
}
```

Java Adapter Classes

- Java adapter classes provide the default implementation of listener interfaces.
- If we 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** and **javax.swing.event** packages.

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

Java Adapter Classes

- ❖ java.awt.dnd Adapter classes

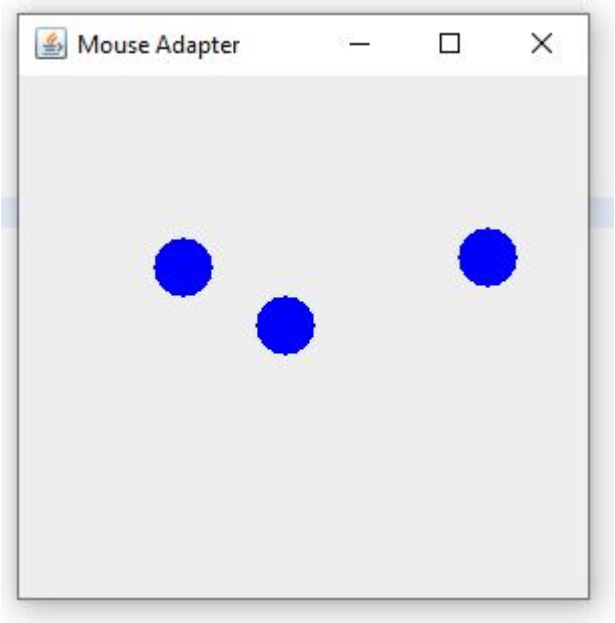
Adapter class	Listener interface
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

- ❖ javax.swing.event Adapter classes

Adapter class	Listener interface
MouseInputAdapter	MouseInputListener
InternalFrameAdapter	InternalFrameListener

Java Adapter Classes-Example(Mouse Adapter)

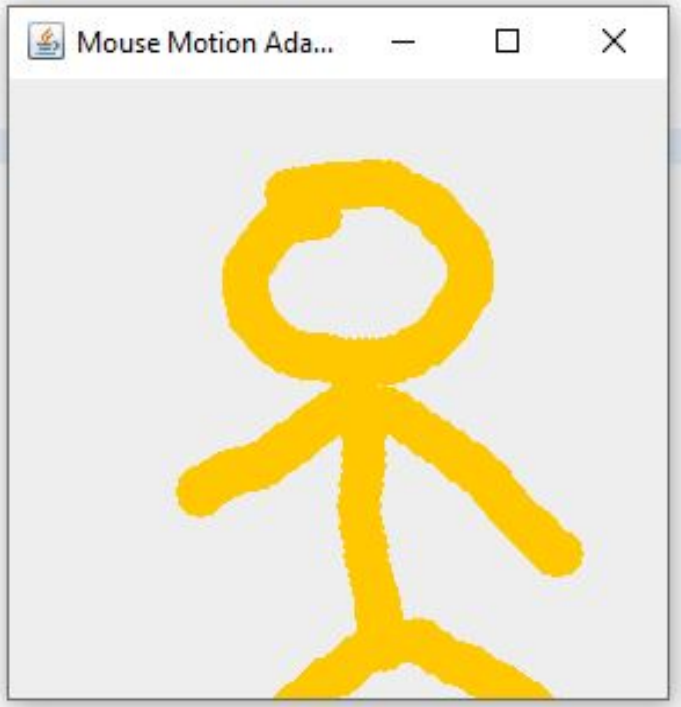
```java
public class MouseAdapterExample extends MouseAdapter {
   JFrame f;
   public MouseAdapterExample() {
        f = new JFrame ("Mouse Adapter");
        f.addMouseListener(this);
        f.setSize (300, 300);
        f.setLayout (null);
        f.setVisible (true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

   @Override
    public void mouseClicked (MouseEvent e) {
        Graphics g = f.getGraphics();
        g.setColor (Color.BLUE);
        g.fillOval (e.getX(), e.getY(), 30, 30);
    }
}
```




```java
public class MouseAdapterExampleMain {
    public static void main(String[] args) {
        new MouseAdapterExample();
    }
}
```

Java Adapter Classes-Example(Mouse Adapter)

```java
public class MouseMotionAdapterExample extends MouseMotionAdapter{
    JFrame f;
    public MouseMotionAdapterExample() {
        f = new JFrame ("Mouse Motion Adapter");
            f.setSize (300, 300);
            f.setLayout (null);
            f.setVisible (true);
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            f.addMouseMotionListener(this);
    }

    @Override
    public void mouseDragged (MouseEvent e) {
            Graphics g = f.getGraphics();
            g.setColor (Color.ORANGE);
            g.fillOval (e.getX(), e.getY(), 20, 20);
    }

}
```




```java
public class MouseMotionAdapterMain {
    public static void main(String[] args) {
        new MouseMotionAdapterExample();
    }
}
```

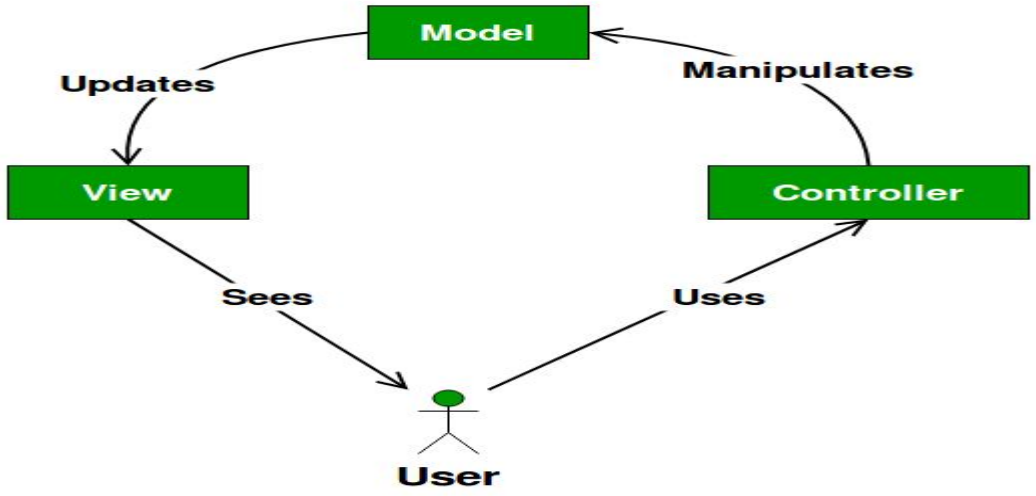
Swing and the MVC Design Pattern

Design Patterns

- A pattern serves as a guide for creating a good design.
- Patterns are based on **sound common sense** and the **application of fundamental design principles.**
- In software engineering, a design pattern **is a general repeatable solution to a commonly occurring problem in software design.**
- A design pattern provides a general reusable solution to a common design problem
- These are well-tested solutions to common problems and issues we run into in software development.
- A design pattern isn't a finished design that can be transformed directly into code.
- It is a description or template for how to solve a problem that can be used in many different situations.

MVC Pattern

- MVC stands for Model View Controller
- It is a design pattern created to solve the design problem of a rather frequent application.
- The basic idea of this pattern is to divide the program (or module) into three main modules: The Model, the View and the Controller.
- The benefits of using the Model-View-Control (MVC) pattern in our development projects is that we can **completely decouple our business and presentation layers.**



Model

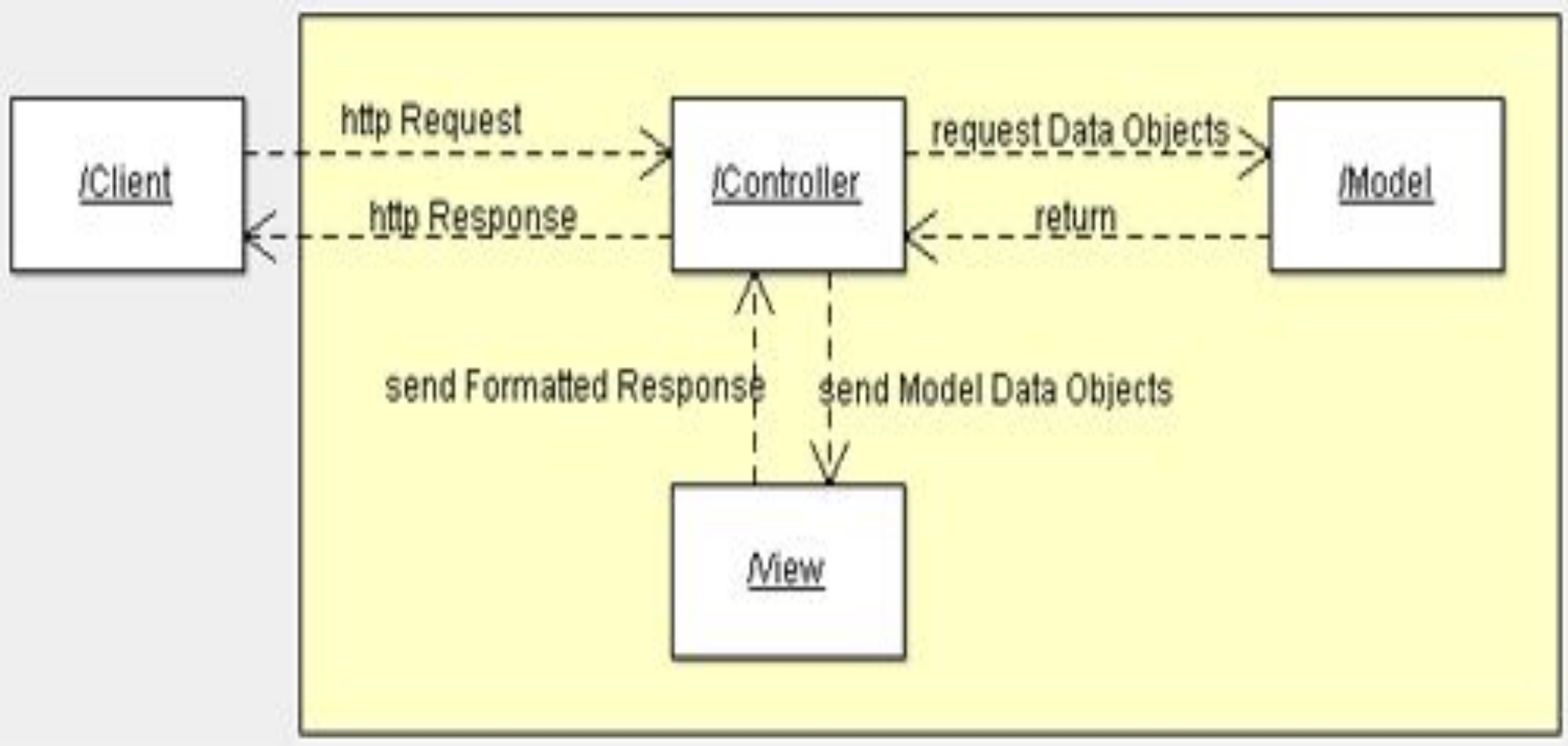
- The **Model** contains only the pure application data.
- It contains no logic describing how to present the data to a user.
- Model represent either the data that is being transferred between the View and Controller components or any other business logic-related data.
- The model is responsible for managing the data of the application. It receives user input from the controller.
- In its simplest form it could simply be some kind of data container depending on the application.
- Model objects **encapsulate the data** specific to an application and define the **logic and computation that manipulate and process that data.**

View

- The View component is used for all the UI logic of the application.
- The view means presentation of the model in a particular format.
- View represents the visualization of the data that model contains.
- View is responsible to display the data provided by the model in a specific form.
- A major purpose of view objects is to display data from the application's model objects and to enable the editing of that data.
- Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

Controller

- The Controller exists between the view and the model.
- Controllers act as an interface between Model and View components to process all the business logic and incoming requests.
- It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events
- It is responsible to handle the request and return a response to user by loading appropriate View with data from Model.
- It keeps view and model separate.


/Client
http Request
http Response
/Controller
request Data Objects
return
/Model
send Formatted Response
send Model Data Objects
/View

MVC Summary

- Model stores/represents the content
- View displays the content
- Controller handles user input

Swing and the MVC Design Pattern

Design Patterns

- A pattern serves as a guide for creating a good design.
- Patterns are based on **sound common sense** and the **application of fundamental design principles.**
- In software engineering, a design pattern **is a general repeatable solution to a commonly occurring problem in software design.**
- A design pattern provides a general reusable solution to a common design problem
- These are well-tested solutions to common problems and issues we run into in software development.
- A design pattern isn't a finished design that can be transformed directly into code.
- It is a description or template for how to solve a problem that can be used in many different situations.

Swing and the MVC Design Pattern-Example

- Follow the project attached for MVC example