Java Swing

Used to create window-based applications.

What is GUI in Java?

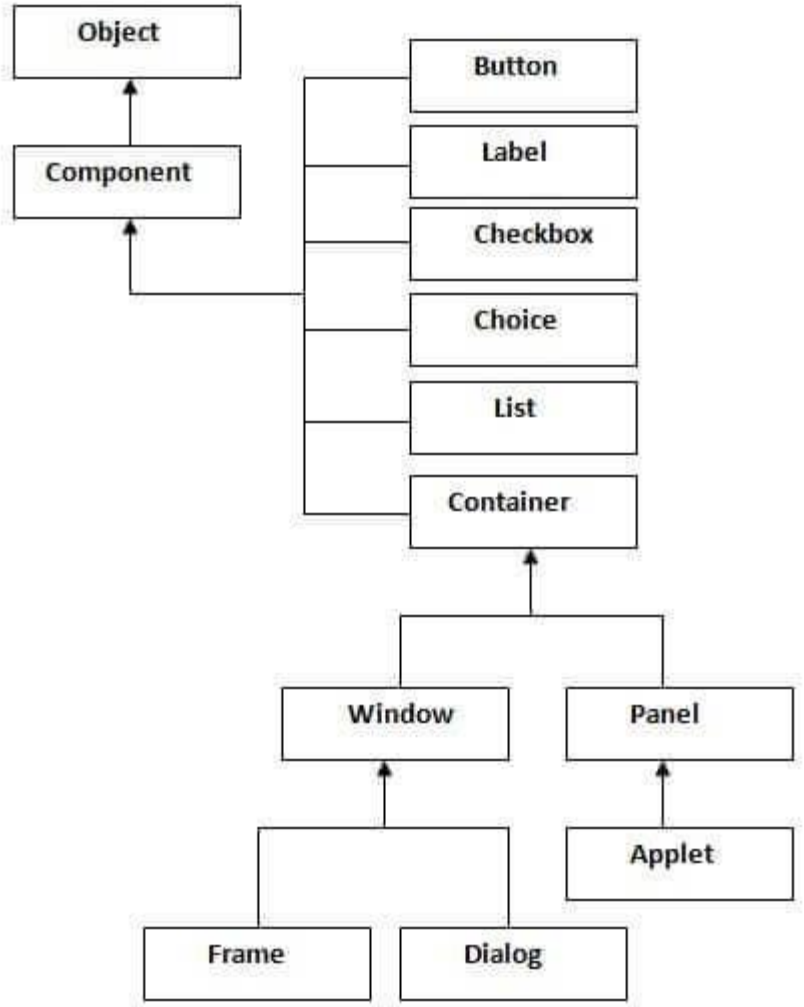
- GUI (Graphical User Interface) In Java gives programmers an easy-to-use visual experience to build Java applications.
- It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with the applications.
- Swing GUI in Java plays an important role in building easy interfaces.

Swing

- Swing in Java is a Graphical User Interface (GUI) toolkit that includes a rich set of widgets. It is a part of Java Foundation Classes(JFC), which is an API for Java programs that provide GUI.
- Swing includes packages that let you make a sophisticated set of GUI components for your Java applications and it is platform-independent.
- The Swing library is built on top of the Java Abstract Widget Toolkit (AWT), an older, platform dependent GUI toolkit.
- You can use the Java GUI components like button, textbox, etc. from the library and do not have to create the components from scratch.

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 i.e. its components are using the resources of OS.



Difference between Swing and AWT

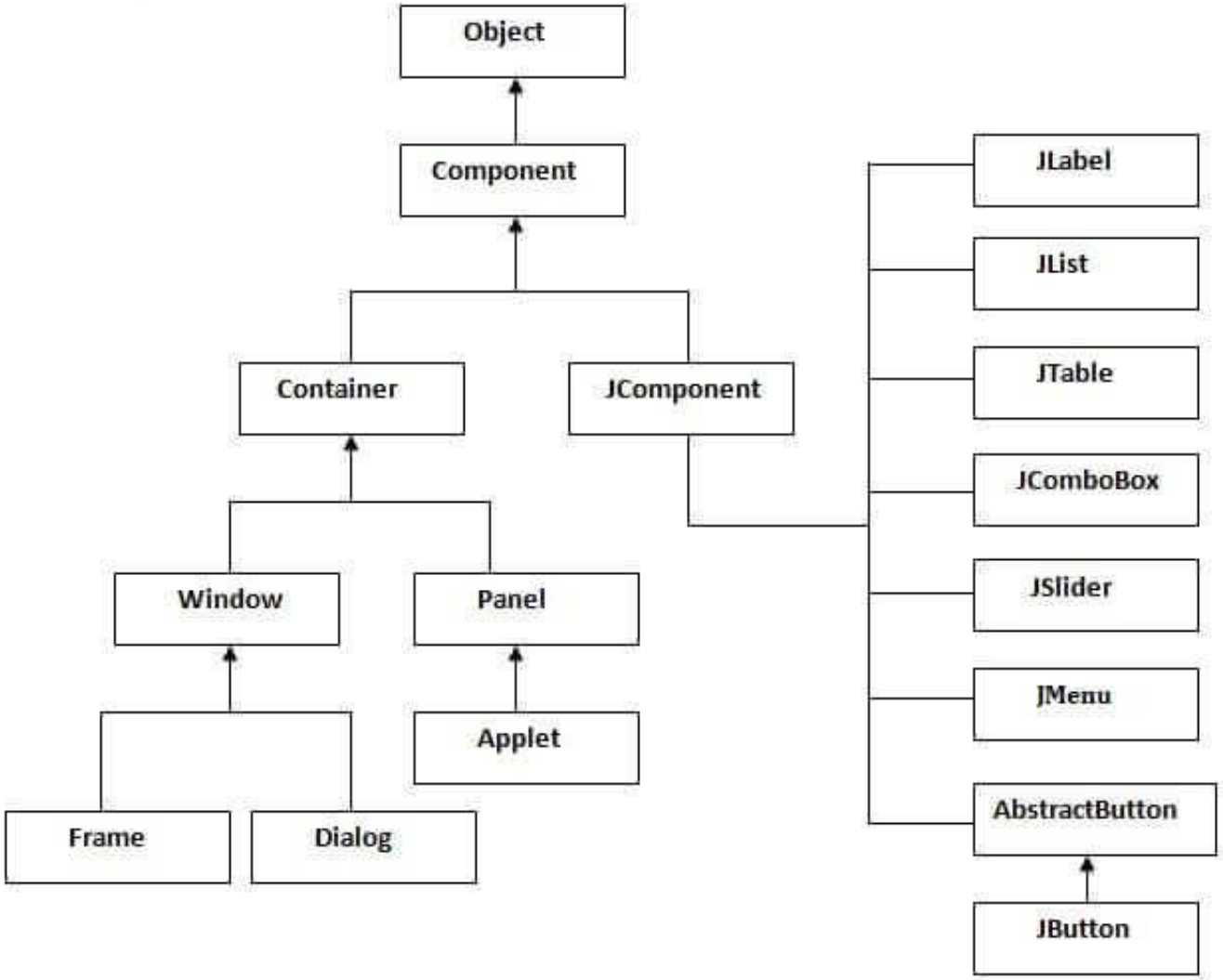
1. AWT components are platform-dependent.
2. AWT components are heavyweight.
3. AWT doesn't support pluggable look and feel.
4. AWT provides less components than Swing.
5. AWT doesn't follows MVC(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.

1. Java swing components are platform-independent.
2. Swing components are lightweight.
3. Swing supports pluggable look and feel.
4. Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5. Swing follows MVC.

Swing Features

- Light Weight – Swing components are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.
- Rich Controls – Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, and table controls.
- Highly Customizable – Swing controls can be customized in a very easy way as visual apperance is independent of internal representation.
- Pluggable look-and-feel – SWING based GUI Application look and feel can be changed at run-time, based on available values.

Swing Hierarchy



Component Class

A Component is the abstract base class for the non menu user-interface controls of SWING. Component represents an object with graphical representation

Container Class

A Container is a component that can contain other SWING components

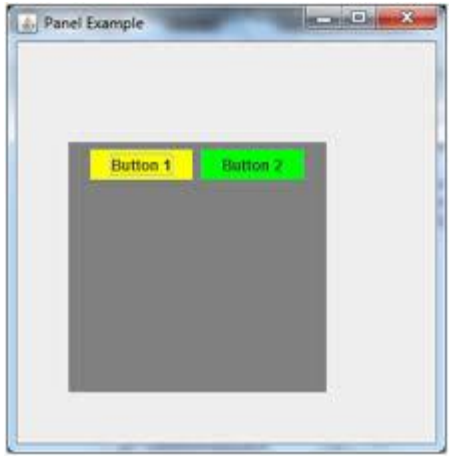
JComponent Class

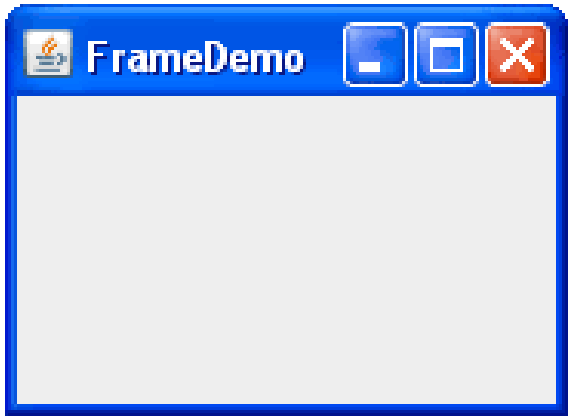
A JComponent is a base class for all SWING UI components. In order to use a SWING component that inherits from JComponent, the component must be in a containment hierarchy whose root is a top-level SWING container

Container

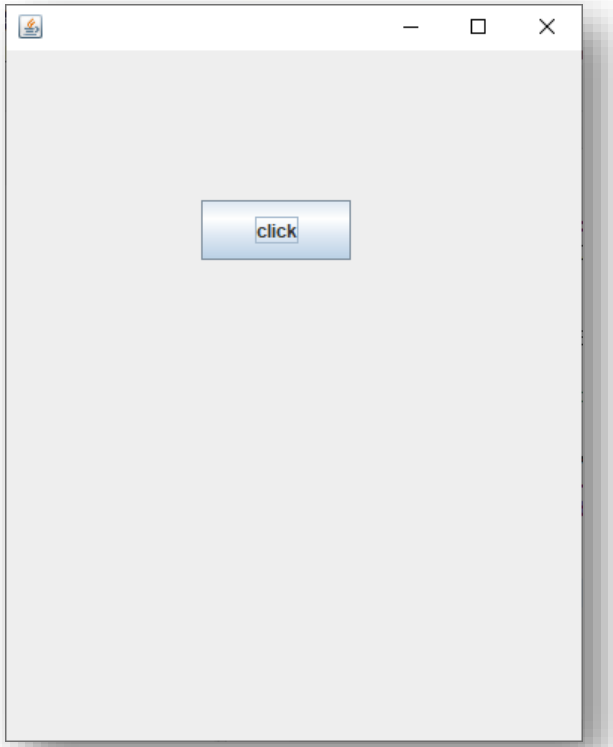
Container classes are classes that can have other components on it. So for creating a GUI, we need at least one container object. There are 3 types of containers.

1. Panel: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window.
2. Frame: It is a fully functioning window with its title and icons.
3. Dialog: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame.





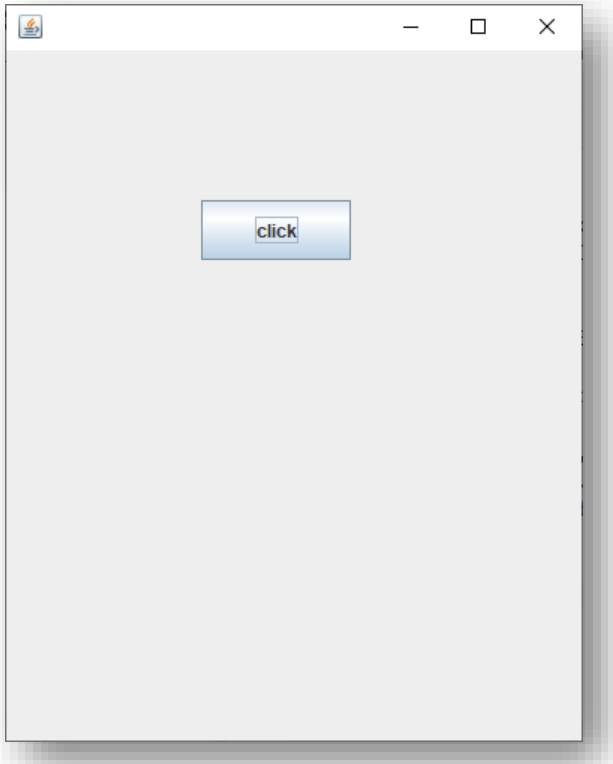
```java
import javax.swing.*;
public class Driver {
    public static void main(String[] args) {
        JFrame f=new JFrame();//creating instance of JFrame

        JButton b=new JButton("click");//creating instance of JButton
        b.setBounds(130,100,100, 40);//x axis, y axis, width, height

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

        f.setSize(400,500);//400 width and 500 height
        f.setLayout(null);//using no layout managers
        f.setVisible(true);//making the frame visible
    }
}
```



A simple Java Swing-I

We are creating one button and adding it on the JFrame object inside the main() method.

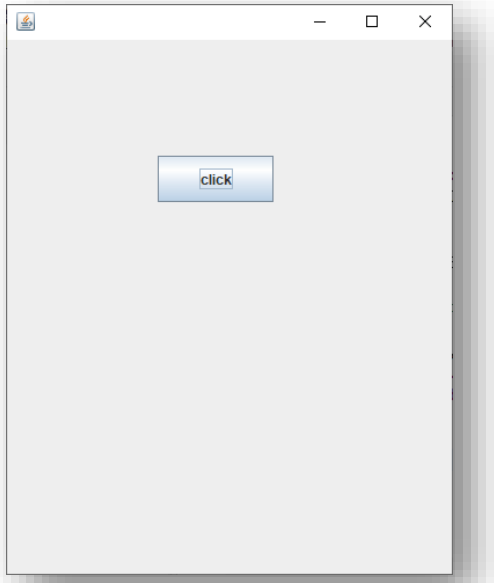
```java
import javax.swing.*;
public class Driver {
    JFrame f;
    Driver(){
    f=new JFrame();//creating instance of JFrame

    JButton b=new JButton("click");//creating instance of JButton
    b.setBounds(130,100,100, 40);

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

    f.setSize(400,500);//400 width and 500 height
    f.setLayout(null);//using no layout managers
    f.setVisible(true);//making the frame visible
    }

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



A simple Java Swing-II

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

```java
import javax.swing.*;
public class Driver extends JFrame{
//inheriting JFrame
    JFrame f;
    Driver(){
        JButton b=new JButton("click");//create button
        b.setBounds(130,100,100, 40);

        add(b);//adding button on frame
        setSize(400,500);
        setLayout(null);
        setVisible(true);
    }
    public static void main(String[] args) {
        new Driver();
    }
}
```



A simple Java Swing-III

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

Layout Manager

Java Layout Manger

- The Layout manager is used to layout (or arrange) the GUI java components inside a container.
- LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

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 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

BorderLayout(): creates a border layout but with no gaps between the components.

JBorderLayout(int hgap, int vgap): creates a border layout with the given horizontal and vertical gaps between the components.

```java
package swing1;
import java.awt.*;
import javax.swing.*;
class Border {
JFrame f;
Border(){
    f=new JFrame();
    JButton b1=new JButton("NORTH");;
    JButton b2=new JButton("SOUTH");;
    JButton b3=new JButton("EAST");;
    JButton b4=new JButton("WEST");;
    JButton b5=new JButton("CENTER");;
    f.add(b1,BorderLayout.NORTH);
    f.add(b2,BorderLayout.SOUTH);
    f.add(b3,BorderLayout.EAST);
    f.add(b4,BorderLayout.WEST);
    f.add(b5,BorderLayout.CENTER);
    f.setSize(300,300);
    f.setVisible(true);
}
}
public class swing5 {

    public static void main(String[] args) {
        new Border();

    }
}
```

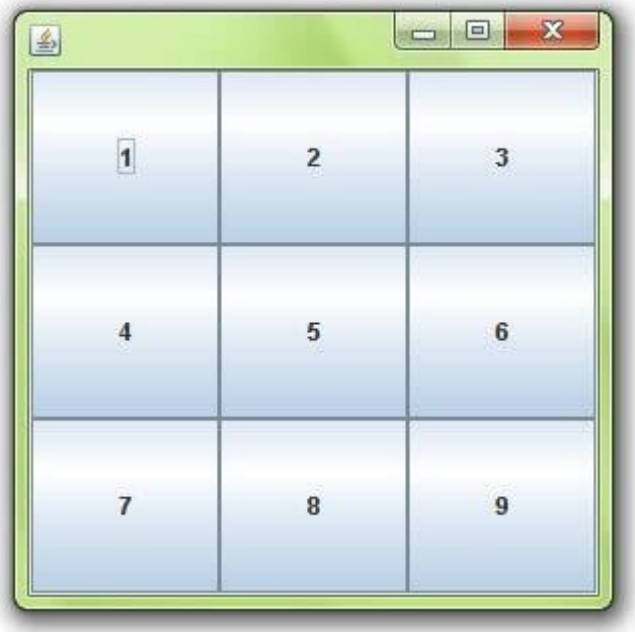
Java GridLayout

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

Constructors of GridLayout class

1. GridLayout(): creates a grid layout with one column per component in a row.
2. GridLayout(int rows, int columns): creates a grid layout with the given rows and columns but no gaps between the components.
3. 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.

```java
package swing1;
import java.awt.*;
import javax.swing.*;
class MyGridLayout{
JFrame f;
MyGridLayout(){
    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");

    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.setLayout(new GridLayout(3,3));
    //setting grid layout of 3 rows and 3 columns

    f.setSize(300,300);
    f.setVisible(true);
}  }
public class swing6 {
    public static void main(String[] args) {
        new MyGridLayout();

    }
```



The layout is set by using *setLayout()* method.

GridLayout(): It Creates a grid layout with a default of one column per component, in a single row Or in the specified row and column

Java FlowLayout

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

Fields of FlowLayout class

1. public static final int LEFT
2. public static final int RIGHT
3. public static final int CENTER
4. public static final int LEADING
5. public static final int TRAILING

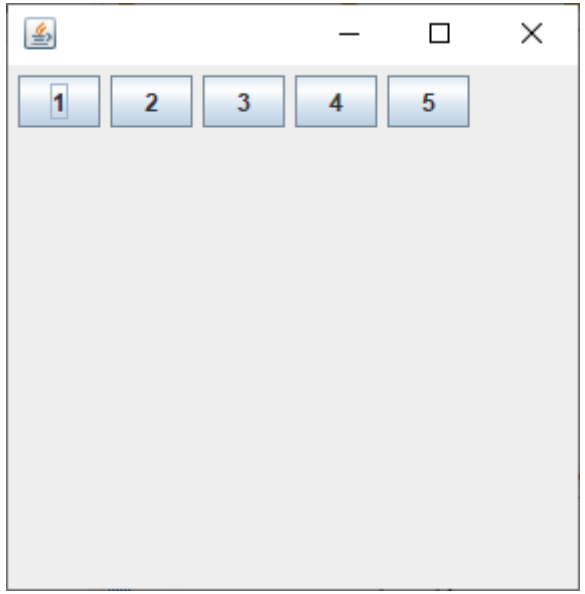
Constructors of FlowLayout class

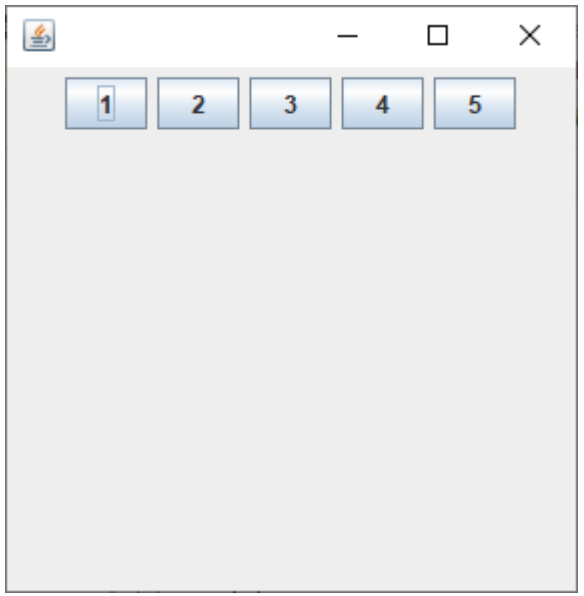
FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.

FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.

FlowLayout(int align, int hgap, int vgap): creates a flow layout with the given alignment and the given horizontal and vertical gap.

```java
package swing1;
import java.awt.*;
import javax.swing.*;
class MyFlowLayout{
JFrame f;
MyFlowLayout(){
    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");

    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

    f.setLayout(new FlowLayout(FlowLayout.RIGHT));
    //setting flow layout of right alignment

    f.setSize(300,300);
    f.setVisible(true);
}
}
public class swing7 {

    public static void main(String[] args) {
        new MyFlowLayout();

    }}
```





LEADING value specifies the components to be left-aligned and the TRAILING value specifies the components to be right-aligned.

Java BoxLayout

The BoxLayout is used to arrange the components either vertically or horizontally. For this purpose, BoxLayout provides four constants.

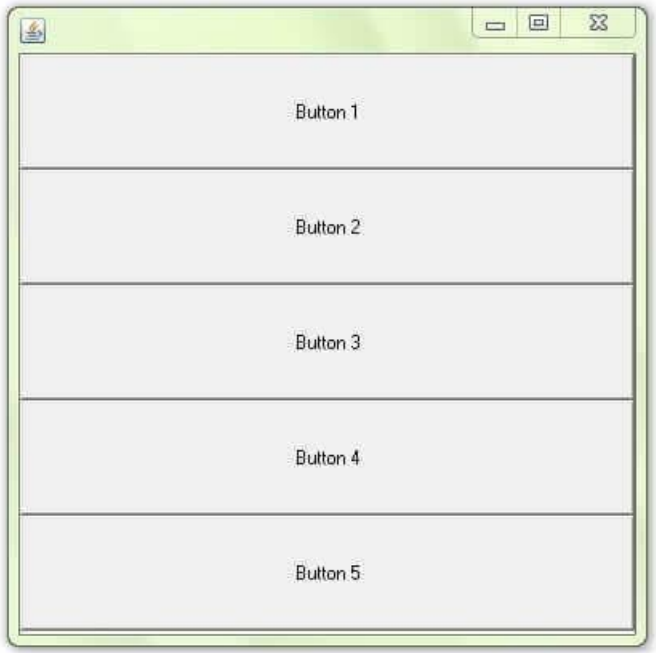
Fields of BoxLayout class

1. public static final int X_AXIS
2. public static final int Y_AXIS
3. public static final int LINE_AXIS
4. public static final int PAGE_AXIS

Constructors of BoxLayout class

BoxLayout(Container c, int axis): creates a box layout that arranges the components with the given axis.

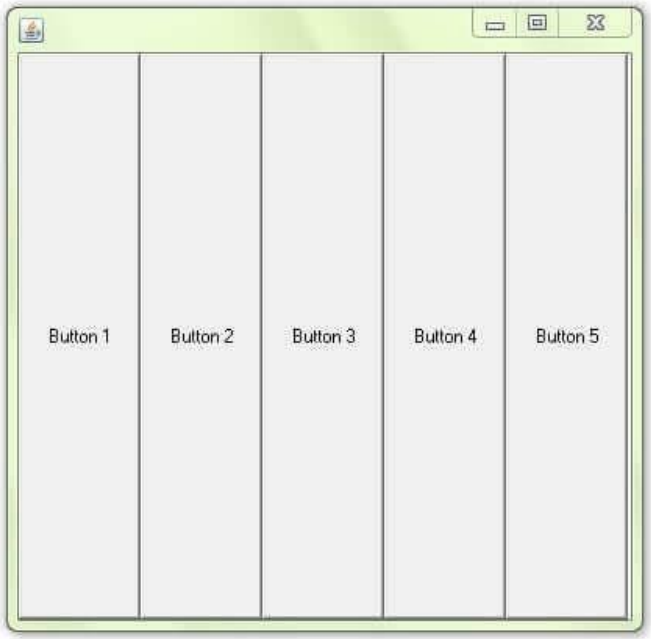
```java
package swing1;
import java.awt.*;
import javax.swing.*;

class BoxLayoutExample1 extends Frame {
 Button buttons[];

 public BoxLayoutExample1 () {
   buttons = new Button [5];

   for (int i = 0;i<5;i++) {
      buttons[i] = new Button ("Button " + (i + 1));
      add (buttons[i]);
    }

setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
setSize(400,400);
setVisible(true);
}

}
public class swing8 {

    public static void main(String[] args) {
        BoxLayoutExample1 b=new BoxLayoutExample1();

    }

}
```



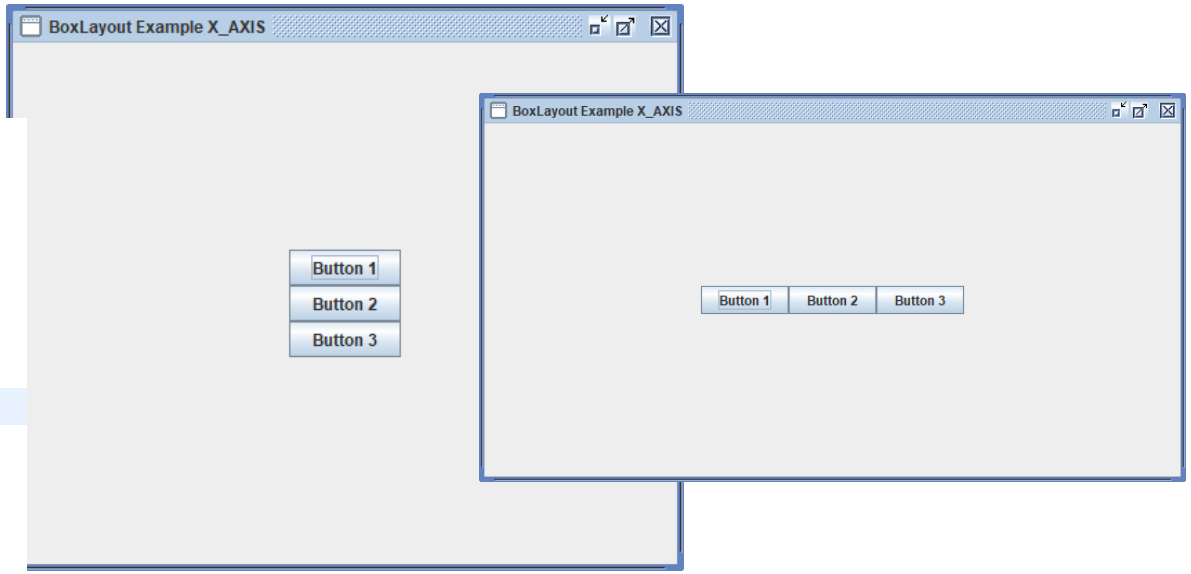
Frame is an **AWT** component , where as **JFrame** is a **Swing** component . You can also that see **JFrame** extends **Frame**.

JButton[] buttons = new JButton[10];

```
1  package swing1;
2
3
4  import java.awt.*;
5  import javax.swing.*;
6
7  class BoxLayoutExample2 extends Frame {
8   Button buttons[];
9
10  public BoxLayoutExample2() {
11    buttons = new Button [5];
12
13    for (int i = 0;i<5;i++) {
14       buttons[i] = new Button ("Button " + (i + 1));
15       add (buttons[i]);
16     }
17
18 setLayout (new BoxLayout(this, BoxLayout.X_AXIS));
19 setSize(400,400);
20 setVisible(true);
21 }
22 }
23 public class swing9 {
24
25     public static void main(String[] args) {
26         BoxLayoutExample2 b=new BoxLayoutExample2();
27
28     }
29
30 }
31
```



```java
public class swingBoxlayout {
    public static void main(String[] args) {
        // Create and set up a frame window
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame("BoxLayout Example X_AXIS");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Set the panel to add buttons
        JPanel panel = new JPanel();
        // Set the BoxLayout to be X_AXIS: from left to right
        BoxLayout boxlayout = new BoxLayout(panel, BoxLayout.X_AXIS);
        // Set the Boxayout to be Y_AXIS from top to down
        //BoxLayout boxlayout = new BoxLayout(panel, BoxLayout.Y_AXIS);
        panel.setLayout(boxlayout);
        // Set border for the panel
        panel.setBorder(new EmptyBorder(new Insets(150, 200, 150, 200)));
        //panel.setBorder(new EmptyBorder(new Insets(50, 80, 50, 80)));
         // Define new buttons
        JButton jb1 = new JButton("Button 1");
        JButton jb2 = new JButton("Button 2");
        JButton jb3 = new JButton("Button 3");
        // Add buttons to the frame (and spaces between buttons)
        panel.add(jb1);
        panel.add(jb2);
        panel.add(jb3);
        // Set size for the frame
        //frame.setSize(300, 300);
        // Set the window to be visible as the default to be false
        frame.add(panel);
        frame.pack();
        frame.setVisible(true);

    }
```




```java
package swing1;
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.BoxLayout;
import javax.swing.Box;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import java.awt.Insets;
import java.awt.Dimension;
```

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout class

1. CardLayout(): creates a card layout with zero horizontal and vertical gap.
2. CardLayout(int hgap, int vgap): creates a card layout with the given horizontal and vertical gap.

Commonly used methods of CardLayout class

- public void next(Container parent): is used to flip to the next card of the given container.
- public void previous(Container parent): is used to flip to the previous card of the given container.
- public void first(Container parent): is used to flip to the first card of the given container.
- public void last(Container parent): is used to flip to the last card of the given container.
- public void show(Container parent, String name): is used to flip to the specified card with the given name.

Java CardLayout

- GridBagLayout
- GroupLayout
- SpringLayout
- ScrollpaneLayout

Similarly can refer more layouts

JOptionPane

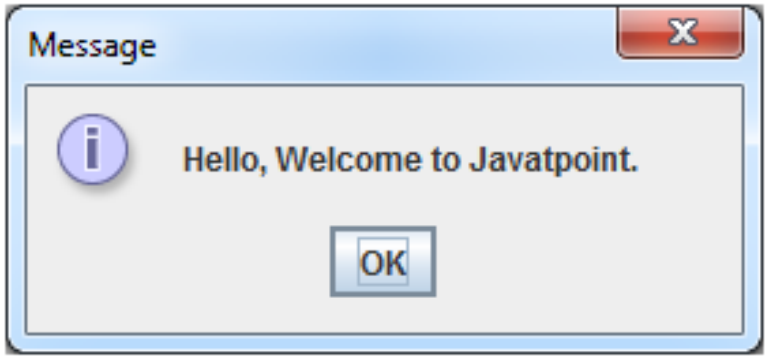
The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. The JOptionPane class inherits JComponent class.

Constructor	Description
JOptionPane()	It is used to create a JOptionPane with a test message.
JOptionPane(Object message)	It is used to create an instance of JOptionPane to display a message.
JOptionPane(Object message, int messageType	It is used to create an instance of JOptionPane to display a message with specified message type and default options.

Methods	Description
JDialog createDialog(String title)	It is used to create and return a new parentless JDialog with the specified title.
static void showMessageDialog(Component parentComponent, Object message)	It is used to create an information-message dialog titled "Message".
static void showMessageDialog(Component parentComponent, Object message, String title, int messageType)	It is used to create a message dialog with given title and messageType.
static int showConfirmDialog(Component parentComponent, Object message)	It is used to create a dialog with the options Yes, No and Cancel; with the title, Select an Option.
static String showInputDialog(Component parentComponent, Object message)	It is used to show a question-message dialog requesting input from the user parented to parentComponent.
void setInputValue(Object newValue)	It is used to set the input value that was selected or input by the user.

Message Dialog Box

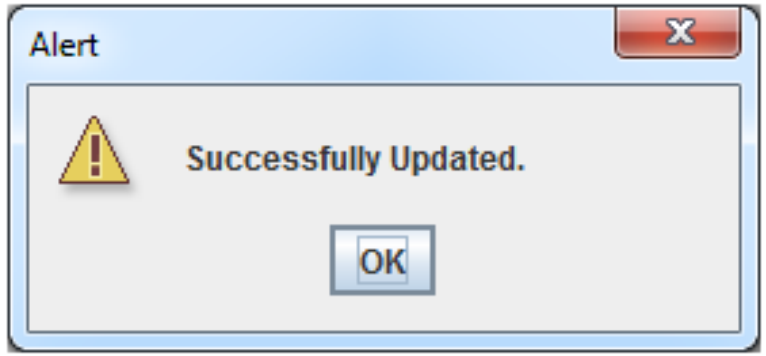
```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Hello, Welcome to Javatpoint.");
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```



public class JOptionPane **extends** JComponent **implements** Accessible

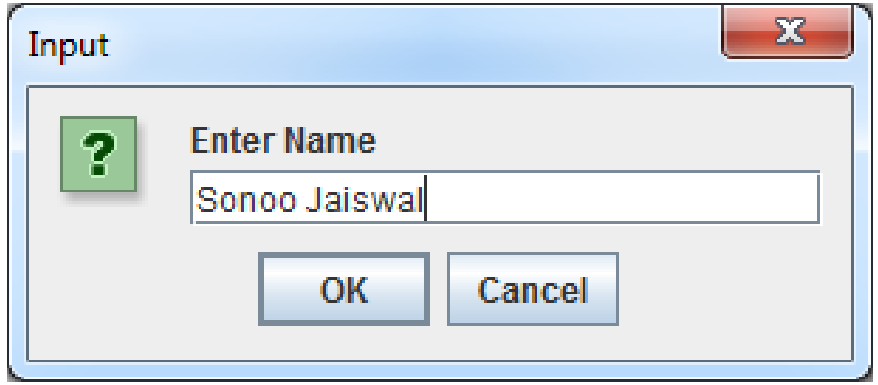
Confirm Dialog Box

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Successfully Updated.","Alert",JOptionPane.WARNING_MESSAGE);
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```



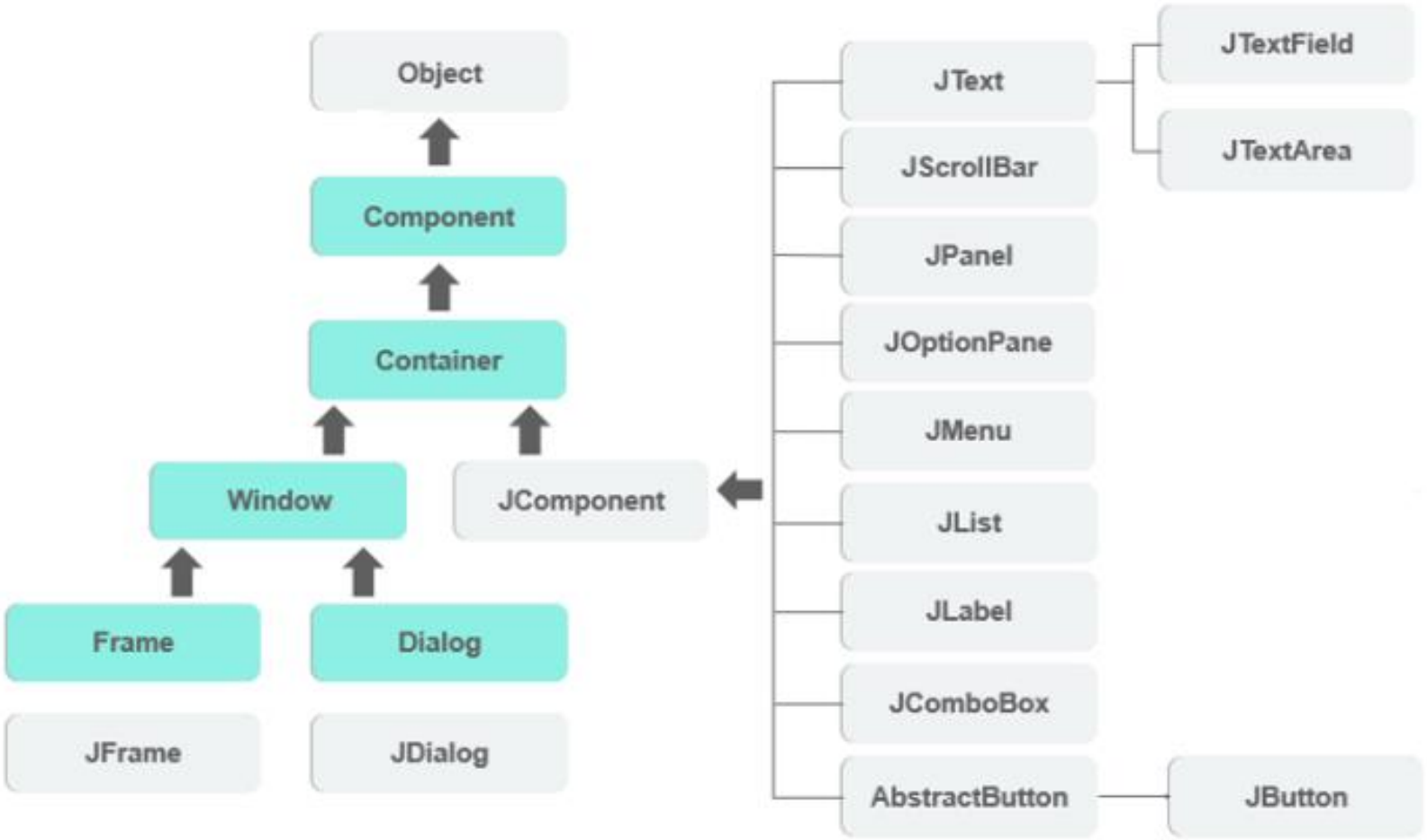
Input Dialog Box

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    String name=JOptionPane.showInputDialog(f,"Enter Name");
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```



• Jcomponent Class

Java Swing Class Hierarchy



JComponent

The JComponent class is the base class of all Swing components except top-level containers. Swing components whose names begin with "J" are descendants of the JComponent class. For example, JButton, JScrollPane, JPanel, JTable etc. But, JFrame and JDialog don't inherit JComponent class because they are the child of top-level containers.

The JComponent class extends the Container class which itself extends Component. The Container class has support for adding components to the container.

Java Swing Class Hierarchy

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

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.

Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

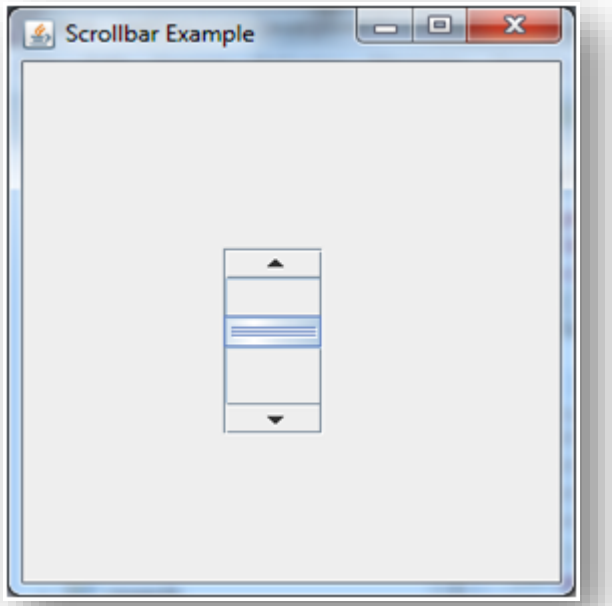
JScrollBar

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.

Sample Code

```
import javax.swing.*;
class ScrollBarExample
{
ScrollBarExample(){
   JFrame f= new JFrame("Scrollbar Example");
 JScrollBar s=new JScrollBar();
s.setBounds(100,100, 50,100);
f.add(s);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[])
{
new ScrollBarExample();
}}
```



setBounds: The first two **arguments** are x and y coordinates of the top-left corner of the component, the third argument is the width of the component and the fourth argument is the height of the component.

JMenuBar, JMenu and JMenuItem

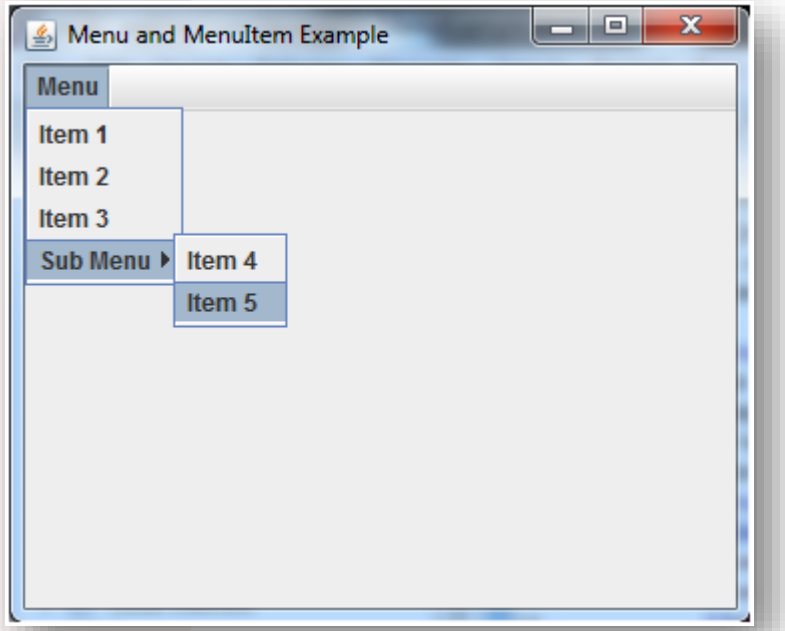
- The JMenuBar class is used to display menubar on the window or frame. It may have several menus.
- The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.
- The object of JMenuItem class adds a simple labeled menu item. The items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

JMenuBar, JMenu and JMenuItem

- public class JMenuBar extends JComponent implements MenuElement, Accessible
- public class JMenu extends JMenuItem implements MenuElement, Accessible
- public class JMenuItem extends AbstractButton implements Accessible, MenuElement

Sample Code

```
class MenuExample
{
        JMenu menu, submenu;
        JMenuItem i1, i2, i3, i4, i5;
        MenuExample(){
        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);
}
```



JFrame Constructor

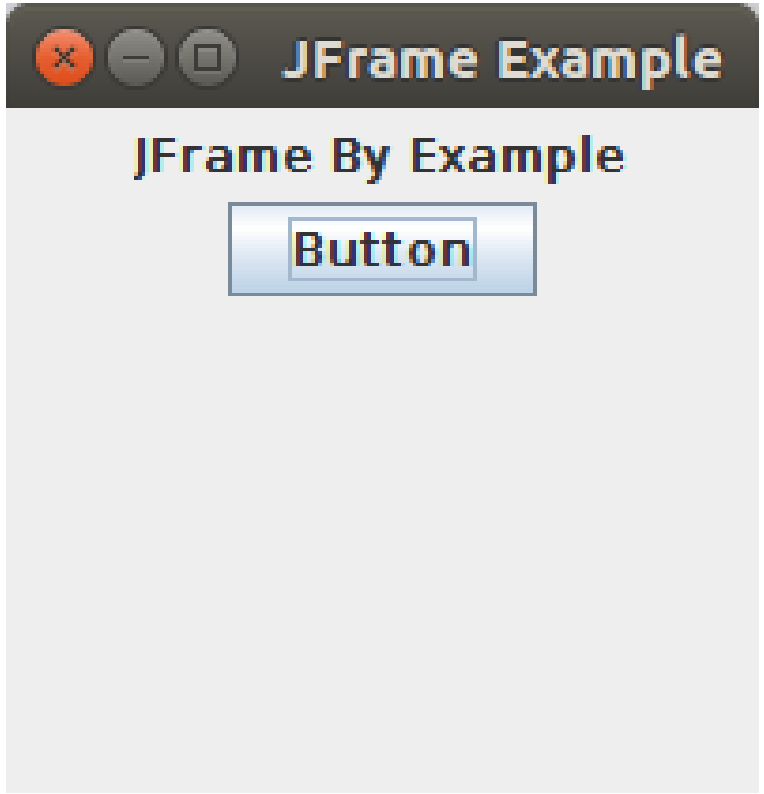
Constructor	Description
JFrame()	It constructs a new frame that is initially invisible.
JFrame(GraphicsConfiguration gc)	It creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.
JFrame(String title)	It creates a new, initially invisible Frame with the specified title.
JFrame(String title, GraphicsConfiguration gc)	It creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

JFrame Methods

Modifier and Type	Method	Description
protected void	addImpl(Component comp, Object constraints, int index)	Adds the specified child Component.
protected JRootPane	createRootPane()	Called by the constructor methods to create the default rootPane.
protected void	frameInit()	Called by the constructors to init the JFrame properly.
void	setContentPane(Containe contentPane)	It sets the contentPane property
static void	setDefaultLookAndFeelDecorated(boolean defaultLookAndFeelDecorated)	Provides a hint as to whether or not newly created JFrames should have their Window decorations (such as borders, widgets to close the window, title...) provided by the current look and feel.
void	setIconImage(Image image)	It sets the image to be displayed as the icon for this window.
void	setJMenuBar(JMenuBar menubar)	It sets the menubar for this frame.
void	setLayeredPane(JLayeredPane layeredPane)	It sets the layeredPane property.
JRootPane	getRootPane()	It returns the rootPane object for this frame.
TransferHandler	getTransferHandler()	It gets the transferHandler property.

Sample Code

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.Jpanel;
public class JFrameExample {
    public static void main(String s[]) {
        JFrame frame = new JFrame("JFrame Example");
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        JLabel label = new JLabel("JFrame By Example");
        JButton button = new JButton();
        button.setText("Button");
        panel.add(label);
        panel.add(button);
        frame.add(panel);
        frame.setSize(200, 300);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```



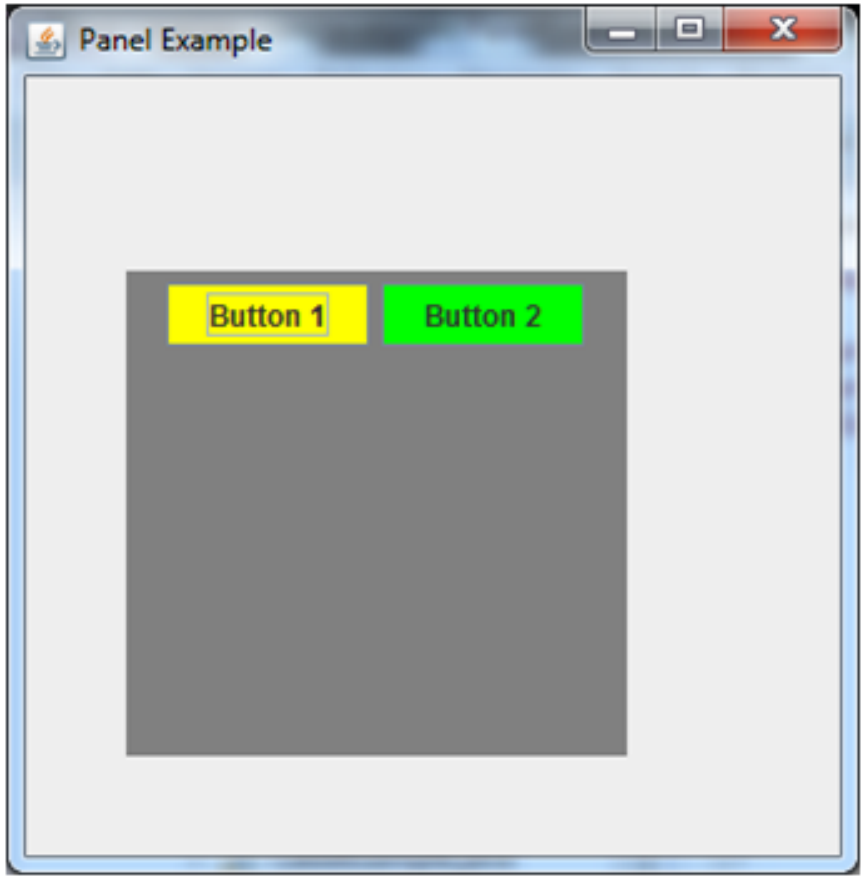
JPanel

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

It doesn't have title bar.

Sample Code

```
import java.awt.*;
import javax.swing.*;
public class PanelExample {
    PanelExample()
      {
      JFrame f= new JFrame("Panel Example");
      JPanel panel=new JPanel();
      panel.setBounds(40,80,200,200);
      panel.setBackground(Color.gray);
      JButton b1=new JButton("Button 1");
      b1.setBounds(50,100,80,30);
      b1.setBackground(Color.yellow);
      JButton b2=new JButton("Button 2");
      b2.setBounds(100,100,80,30);
      b2.setBackground(Color.green);
      panel.add(b1); panel.add(b2);
      f.add(panel);
            f.setSize(400,400);
            f.setLayout(null);
            f.setVisible(true);
      }
      public static void main(String args[])
      {
      new PanelExample();
      }
  }
```



Modifier and Type	Method	Description
void	setActionMap(ActionMap am)	It sets the ActionMap to am.
void	setBackground(Color bg)	It sets the background color of this component.
void	setFont(Font font)	It sets the font for this component.
void	setMaximumSize(Dimension maximumSize)	It sets the maximum size of this component to a constant value.
void	setMinimumSize(Dimension minimumSize)	It sets the minimum size of this component to a constant value.
protected void	setUI(ComponentUI newUI)	It sets the look and feel delegate for this component.
void	setVisible(boolean aFlag)	It makes the component visible or invisible.
void	setForeground(Color fg)	It sets the foreground color of this component.
String	getToolTipText(MouseEvent event)	It returns the string to be used as the tooltip for event.
Container	getTopLevelAncestor()	It returns the top-level ancestor of this component (either the containing Window or Applet), or null if this component has not been added to any container.
TransferHandler	getTransferHandler()	It gets the transferHandler property.

JButton

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

Commonly used Constructors:

JButton()- It creates a button with no text and icon.

JButton(String s)- It creates a button with the specified text.

JButton(Icon i)- It creates a button with the specified icon object.

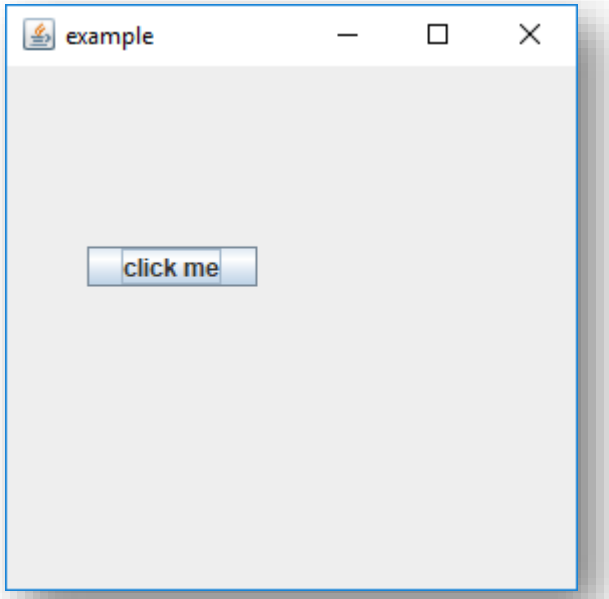
JButton

Commonly used Methods of AbstractButton class:

1. void setText(String s) - It is used to set specified text on button
2. String getText() - It is used to return the text of the button.
3. void setEnabled(boolean b) - It is used to enable or disable the button.
4. void setIcon(Icon b) - It is used to set the specified Icon on the button.
5. Icon getIcon() - It is used to get the Icon of the button.
6. void setMnemonic(int a) - It is used to set the mnemonic on the button.
7. void addActionListener(ActionListener a) - It is used to add the action listener to this object.

Sample code

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



JLabel

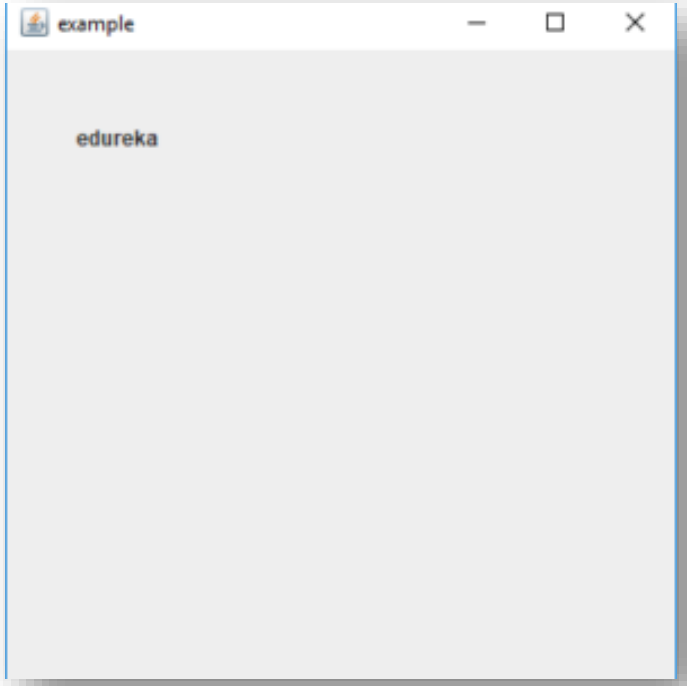
The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

Methods	Description
String getText()	t returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

Sample code

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



JTextField

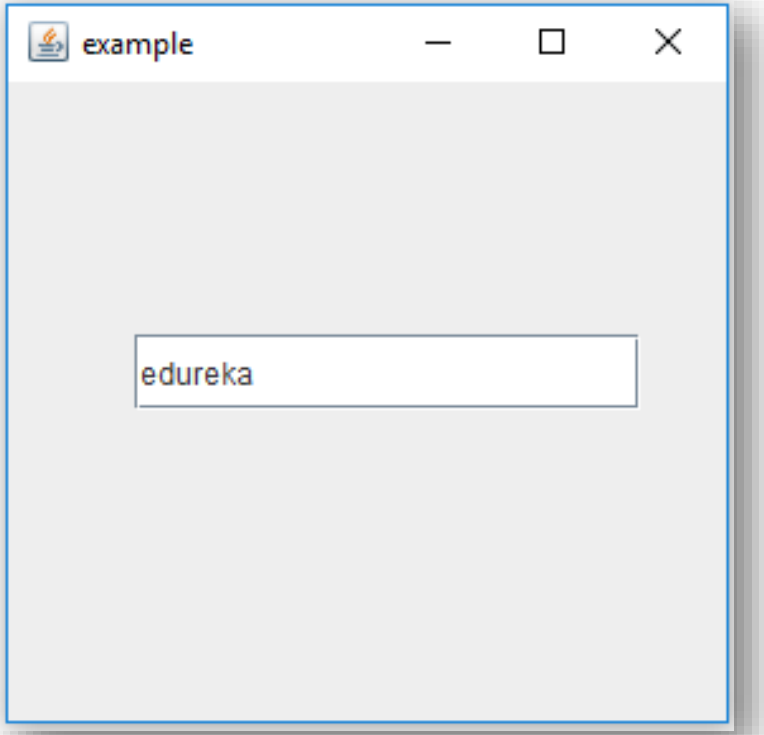
The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Methods	Description
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	It is used to set the current font.
void removeActionListener(ActionListener l)	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

Sample code

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



JTextArea

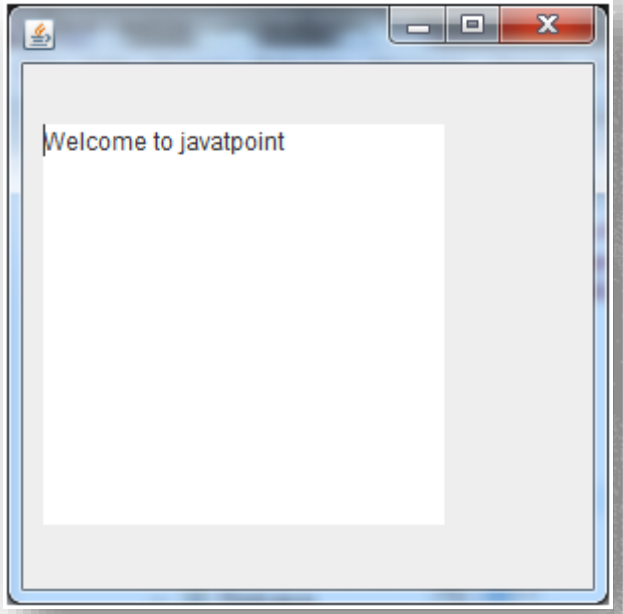
The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class.

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.

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.

Sample code

```
import javax.swing.*;
public class TextAreaExample
{
    TextAreaExample(){
       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);
    }
public static void main(String args[])
   {
  new TextAreaExample();
   }}
```



JCheckBox

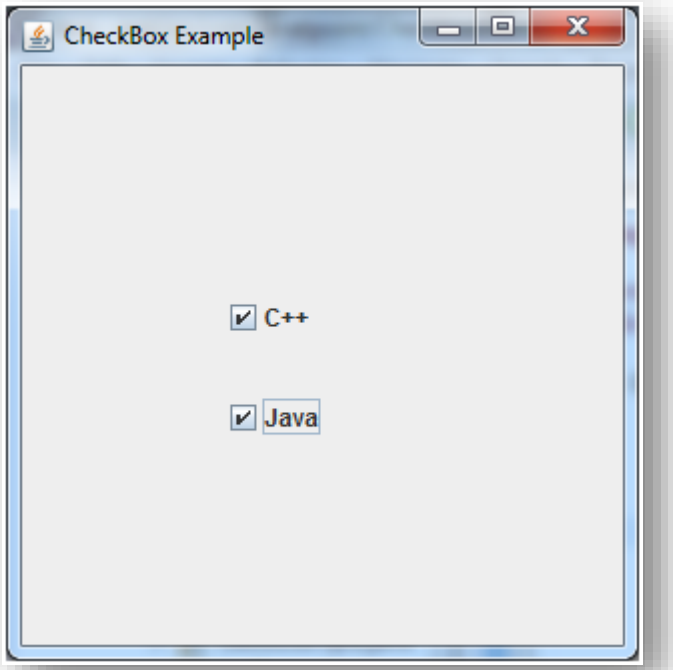
The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits JToggleButton class.

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.

Methods	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.
protected String paramString()	It returns a string representation of this JCheckBox.

Sample code

```
import javax.swing.*;
public class CheckBoxExample
{
    CheckBoxExample(){
       JFrame f= new JFrame("CheckBox Example");
       JCheckBox checkBox1 = new JCheckBox("C++");
       checkBox1.setBounds(100,100, 50,50);
       JCheckBox checkBox2 = new JCheckBox("Java", true);
       checkBox2.setBounds(100,150, 50,50);
       f.add(checkBox1);
       f.add(checkBox2);
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
    }
public static void main(String args[])
   {
   new CheckBoxExample();
   }}
```



JRadioButton

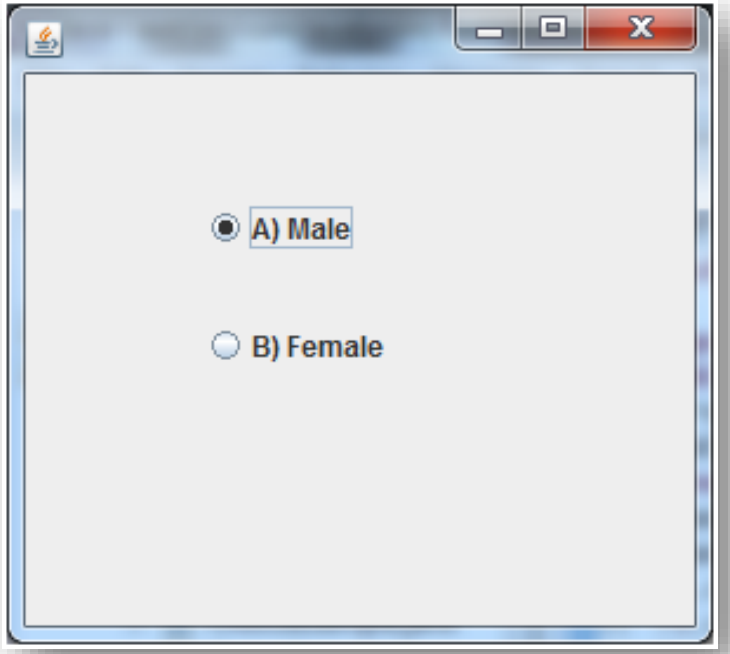
The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

Constructor	Description
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected status.

Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Sample code

```
import javax.swing.*;
public class RadioButtonExample {
JFrame f;
RadioButtonExample(){
f=new JFrame();
JRadioButton r1=new JRadioButton("A) Male");
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);
}
public static void main(String[] args) {
    new RadioButtonExample();
}
}
```



JComboBox

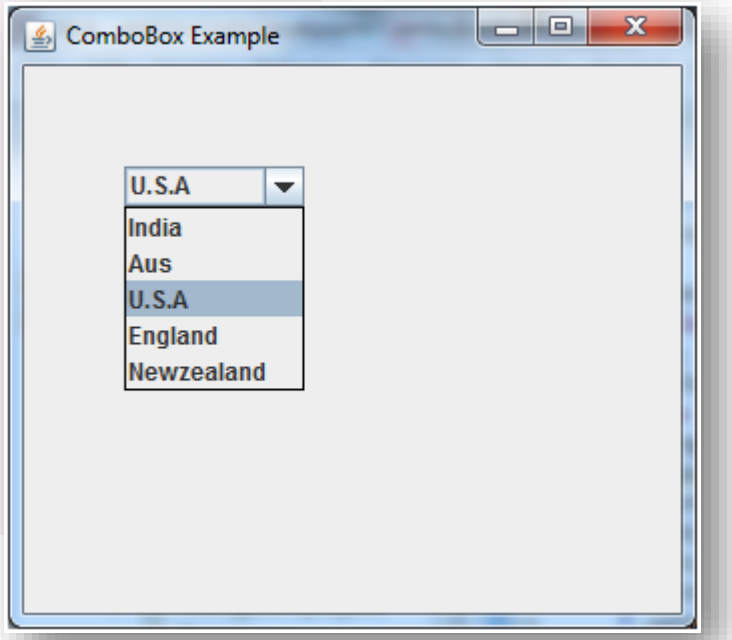
The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array.
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

Methods	Description
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox is editable.
void addActionListener(ActionListener a)	It is used to add the ActionListener.
void addItemListener(ItemListener i)	It is used to add the ItemListener.

Sample code

```
import javax.swing.*;
public class ComboBoxExample {
JFrame f;
ComboBoxExample(){
    f=new JFrame("ComboBox Example");
    String country[]={"India","Aus","U.S.A","England","Newzealand"};
    JComboBox cb=new JComboBox(country);
    cb.setBounds(50, 50,90,20);
    f.add(cb);
    f.setLayout(null);
    f.setSize(400,500);
    f.setVisible(true);
}
public static void main(String[] args) {
    new ComboBoxExample();
}
}
```

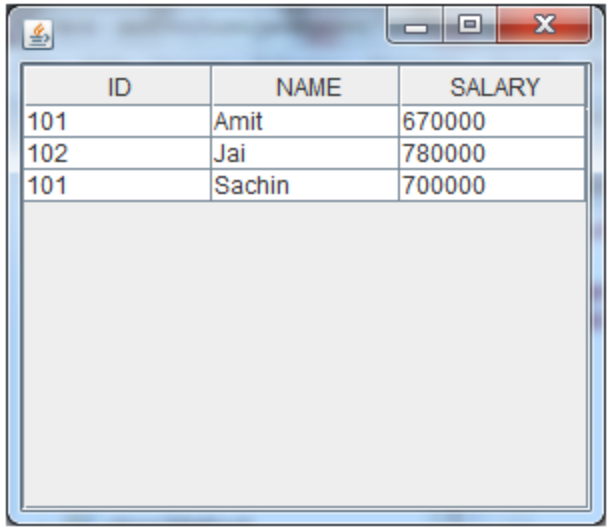


JTable

The JTable class is used to display data in tabular form. It is composed of rows and columns.

Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

```
import javax.swing.*;
public class TableExample {
    JFrame f;
    TableExample(){
    f=new JFrame();
    String data[][]={ {"101","Amit","670000"},
                      {"102","Jai","780000"},
                      {"101","Sachin","700000"}};
    String column[]={"ID","NAME","SALARY"};
    JTable jt=new JTable(data,column);
    jt.setBounds(30,40,200,300);
    JScrollPane sp=new JScrollPane(jt);
    f.add(sp);
    f.setSize(300,400);
    f.setVisible(true);
}
public static void main(String[] args) {
    new TableExample();
}
}
```



Sample code

JList

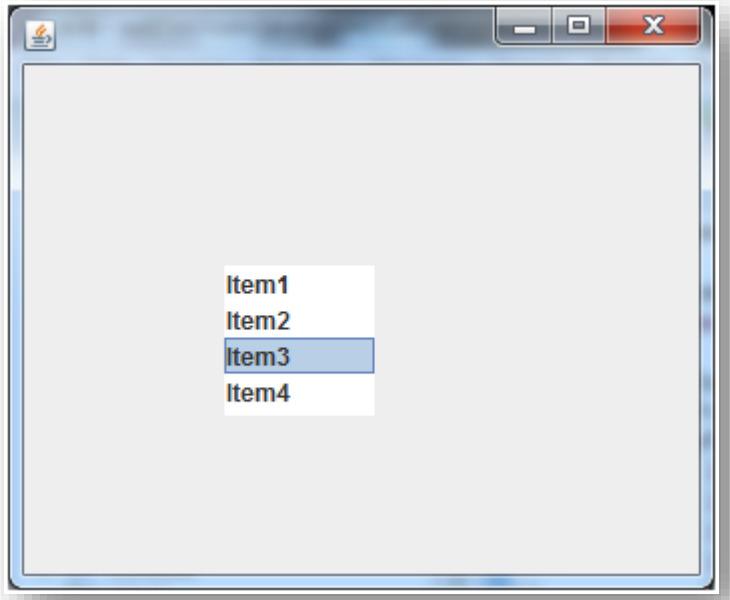
The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

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.

Sample Code

```
import javax.swing.*;
public class ListExample
{
    ListExample(){
       JFrame f= new JFrame();
       DefaultListModel<String> l1 = new DefaultListModel<>();
        l1.addElement("Item1");
        l1.addElement("Item2");
        l1.addElement("Item3");
        l1.addElement("Item4");
        JList<String> list = new JList<>(l1);
        list.setBounds(100,100, 75,75);
        f.add(list);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
public static void main(String args[])
   {
  new ListExample();
   }}
```



Event Handling

Intro

What is an Event?

- 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.

Types of Event

The events can be broadly classified into two categories:

- Foreground Events – These events require direct interaction of the user. They are generated as consequences of a person interacting with the graphical components in the Graphical User Interface.

  For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page, etc.

- Background Events – These events require the interaction of the end user. Operating system interrupts, hardware or software failure, timer expiration, and operation completion are some examples of background events

What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has a code which is known as an event handler, that is executed when an event occurs.

Java uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.

The Delegation Event Model has the following key participants.

- Source – The source is an object on which the event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide us with classes for the source object.
- Listener – It is also known as event handler. The listener is responsible for generating a response to an event. From the point of view of Java implementation, the listener is also an object. The listener waits till it receives an event. Once the event is received, the listener processes the event and then returns.

Steps Involved in Event Handling

1. The user clicks the button and the event is generated.
2. The object of concerned event class is created automatically and information about the source and the event get populated within the same object.
3. Event object is forwarded to the method of the registered listener class.
4. The method is gets executed and returns.

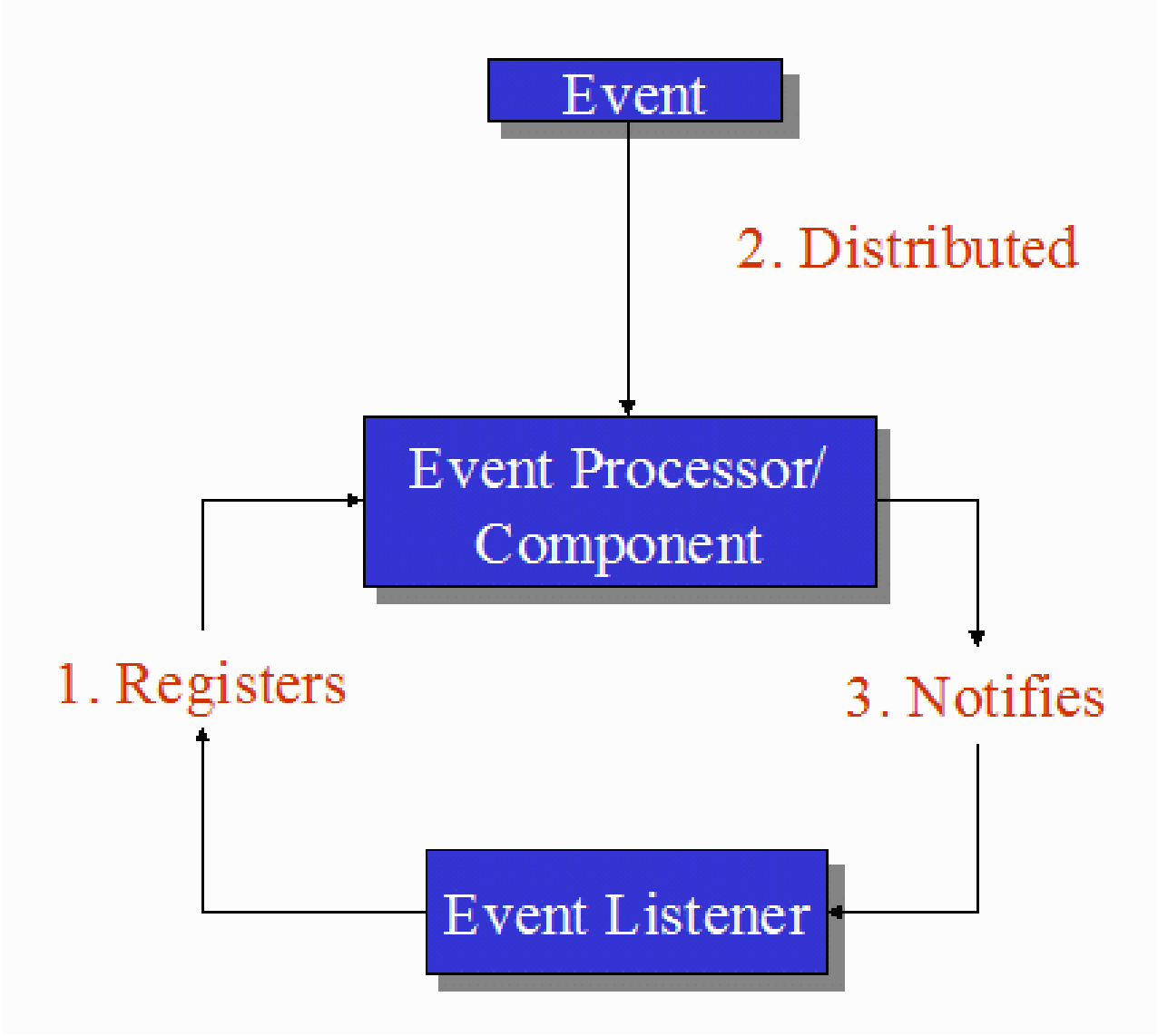
Important Event Classes and Interface

ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved,clicked,pressed or released and also when it enters or exits a component	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener

Important Event Classes and Interface

MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener
AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

The Java Event Listener Pattern



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

Registration Methods

Register the component with the Listener

Java ActionListener Interface

- 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().
- actionPerformed() method
- The actionPerformed() method is invoked automatically whenever you click on the registered component.

```
1.public abstract void actionPerformed(ActionEvent e);
```

Two ways to write action listener

- **Implement the ActionListener interface in the class:**
- **Using Anonymous class**

- **Implement the ActionListener interface in the class:**

Implement the ActionListener interface in the class:

Steps:

1.public class ActionListenerExample Implements ActionListener

2) Register the component with the Listener:

component.addActionListener(instanceOfListenerclass);

3) Override the actionPerformed() method:

```
public void actionPerformed(ActionEvent e){
        //Write the code here
}
```

Java ActionListener Example: Using Anonymous class

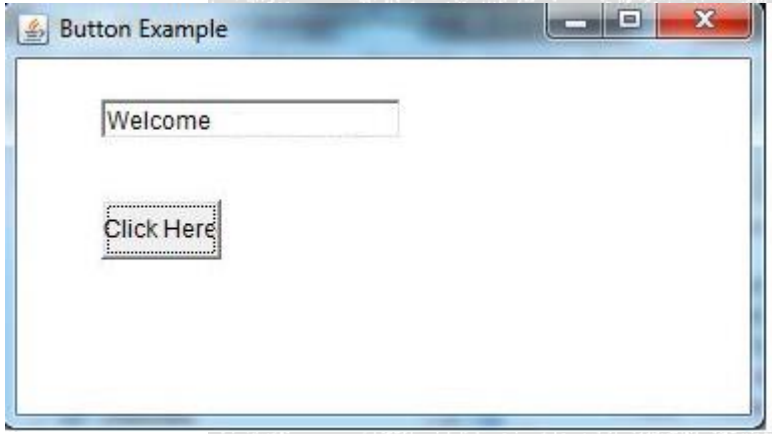
- **Anonymous classes** enable you to make your code more concise. They enable you to declare and instantiate a **class** at the same time..

We can also use the anonymous class to implement the ActionListener. It is the shorthand way, so you do not need to follow the 3 steps:

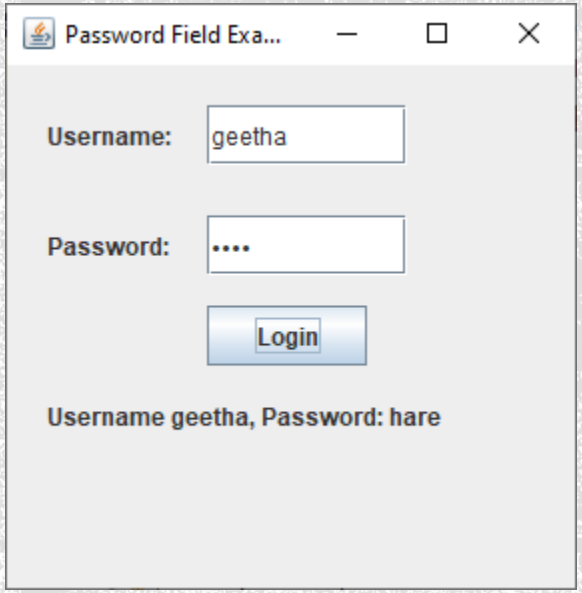
```
b.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent e){
        tf.setText("Welcome .");

}
});
```

```
interface Eatable{
 void eat();
}
class TestAnnonymousInner1{
public static void main(String args[]){
Eatable e=new Eatable(){
public void eat(){
      System.out.println("nice fruits");}};
e.eat();
 }
}
```

```
1.import java.awt.*;
2.import java.awt.event.*;
3.//1st step
4.public class ActionListenerExample implements ActionListener{
5.public static void main(String[] args) {
6.    Frame f=new Frame("ActionListener Example");
7.    final TextField tf=new TextField();
8.    tf.setBounds(50,50, 150,20);
9.    Button b=new Button("Click Here");
10.    b.setBounds(50,100,60,30);
11.    //2nd step
12.    b.addActionListener(this);
13.    f.add(b);f.add(tf);
14.    f.setSize(400,400);
15.    f.setLayout(null);
16.    f.setVisible(true);
17.}
18.//3rd step
19.public void actionPerformed(ActionEvent e){
20.            tf.setText("Welcome .");
21.}
22.}
```



```java
package swing1;
import javax.swing.*;
import java.awt.event.*;
public class eventbutton {

    public static void main(String[] args) {
        JFrame f=new JFrame("Password Field Example");
        final JLabel label = new JLabel();
        label.setBounds(20,150, 200,50);
        final JPasswordField value = new JPasswordField();
        value.setBounds(100,75,100,30);
        JLabel l1=new JLabel("Username:");
        l1.setBounds(20,20, 80,30);
        JLabel l2=new JLabel("Password:");
        l2.setBounds(20,75, 80,30);
        JButton b = new JButton("Login");
        b.setBounds(100,120, 80,30);
        final JTextField text = new JTextField();
        text.setBounds(100,20, 100,30);
        f.add(value); f.add(l1); f.add(label); f.add(l2); f.add(b); f.add(text);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
              String data = "Username " + text.getText();
              data += ", Password: "
                + new String(value.getPassword());
              label.setText(data);
              }
            });
        }
    }
```

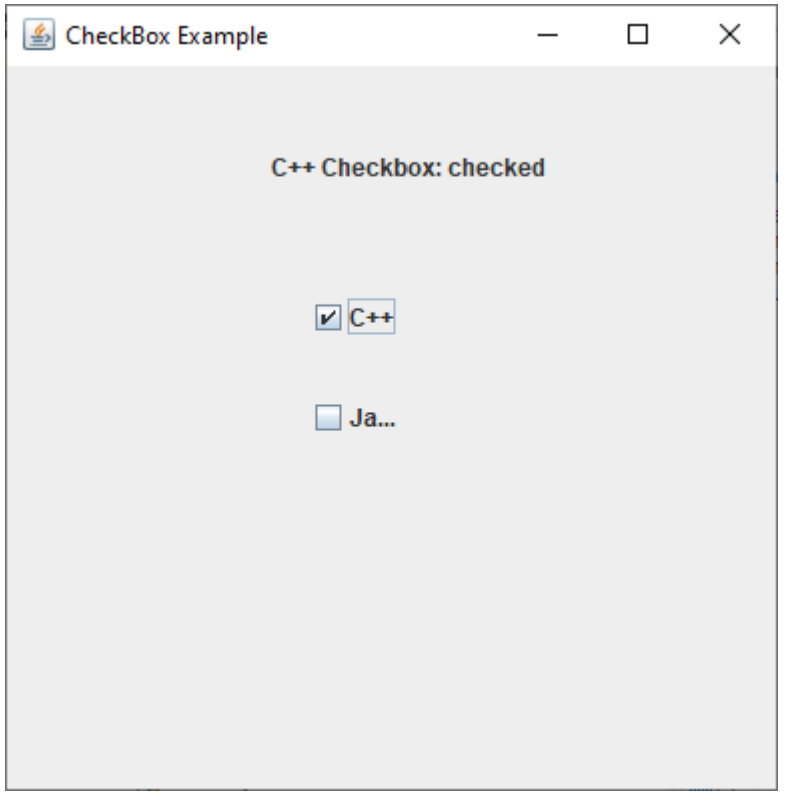


Java ItemListener is notified on clicking checkbox

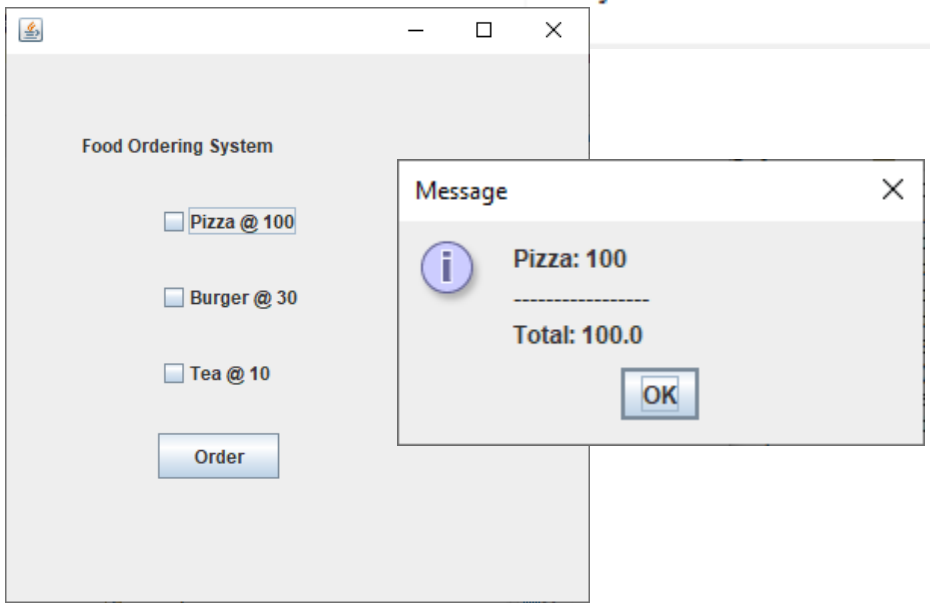
- The Java ItemListener is notified whenever you click on the checkbox. It is notified against ItemEvent. The ItemListener interface is found in java.awt.event package. It has only one method: itemStateChanged().
- itemStateChanged() method
- The itemStateChanged() method is invoked automatically whenever you click or unclick on the registered checkbox component.

```
1.public abstract void itemStateChanged(ItemEvent e);
```

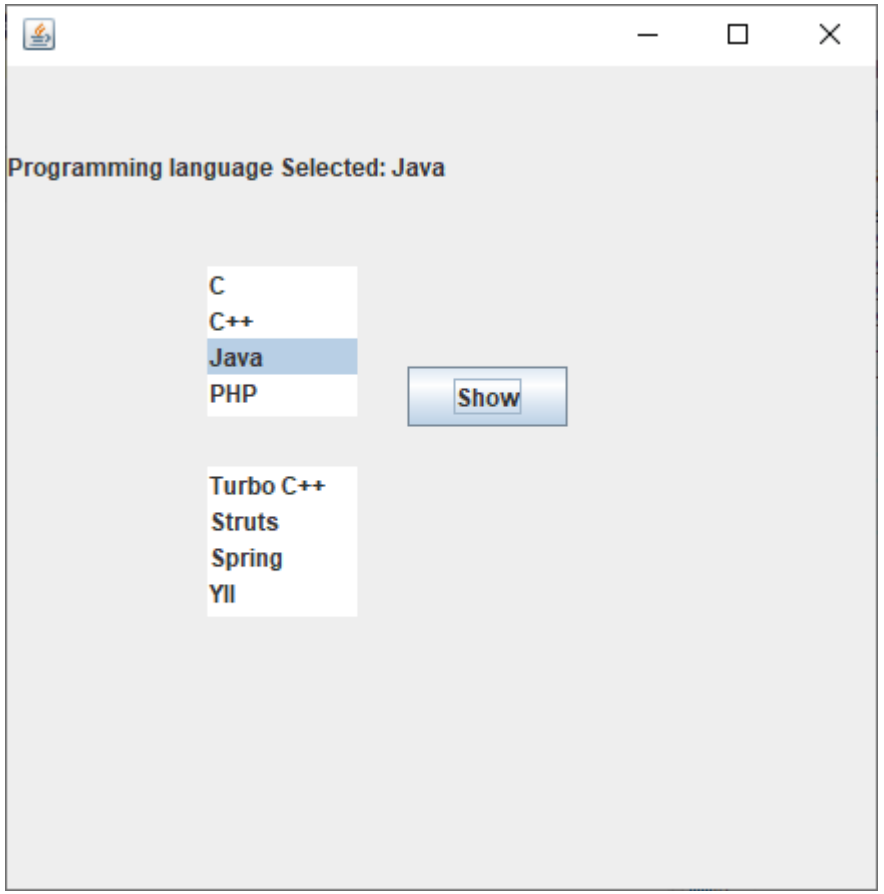
```java
package swing1;
import javax.swing.*;
import java.awt.event.*;
public class CheckBoxExample {
    CheckBoxExample(){
        JFrame f= new JFrame("CheckBox Example");
        final JLabel label = new JLabel();
        label.setHorizontalAlignment(JLabel.CENTER);
        label.setSize(400,100);
        JCheckBox checkbox1 = new JCheckBox("C++");
        checkbox1.setBounds(150,100, 50,50);
        JCheckBox checkbox2 = new JCheckBox("Java");
        checkbox2.setBounds(150,150, 50,50);
        f.add(checkbox1); f.add(checkbox2); f.add(label);
        checkbox1.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                label.setText("C++ Checkbox: "
                + (e.getStateChange()==1?"checked":"unchecked"));
            }
          });
        checkbox2.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                label.setText("Java Checkbox: "
                + (e.getStateChange()==1?"checked":"unchecked"));
            }
          });
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
     }
    public static void main(String[] args) {
        new CheckBoxExample();

    }
```

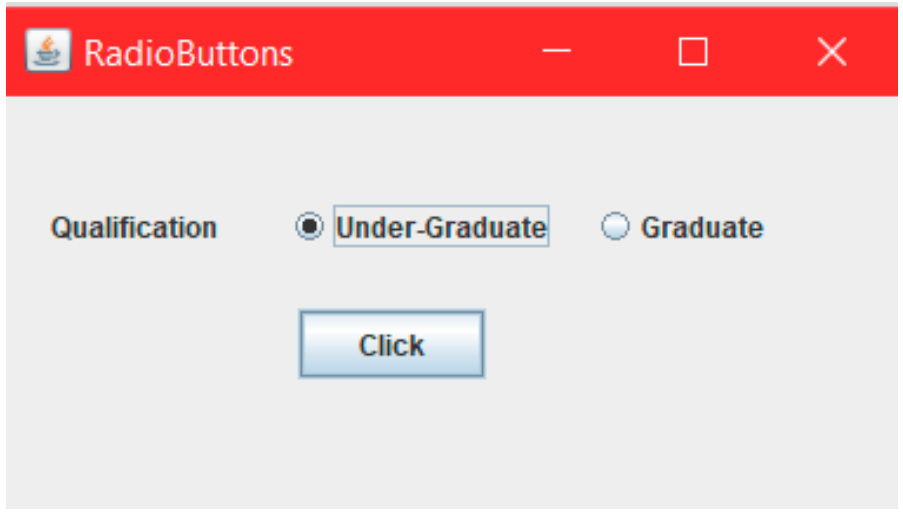


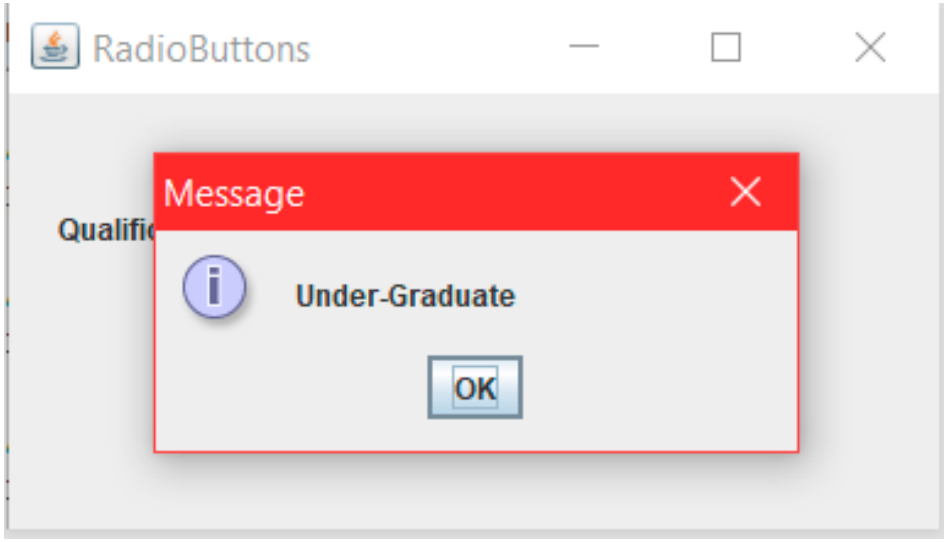
```java
package swing1;
import javax.swing.*;
import java.awt.event.*;

public class CheckBoxExample2 extends JFrame implements ActionListener{
    JLabel l;
    JCheckBox cb1,cb2,cb3;
    JButton b;
    CheckBoxExample2(){
        l=new JLabel("Food Ordering System");
        l.setBounds(50,50,300,20);
        cb1=new JCheckBox("Pizza @ 100");
        cb1.setBounds(100,100,150,20);
        cb2=new JCheckBox("Burger @ 30");
        cb2.setBounds(100,150,150,20);
        cb3=new JCheckBox("Tea @ 10");
        cb3.setBounds(100,200,150,20);
        b=new JButton("Order");
        b.setBounds(100,250,80,30);
        b.addActionListener(this);
        add(l);add(cb1);add(cb2);add(cb3);add(b);
        setSize(400,400);
        setLayout(null);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
    public void actionPerformed(ActionEvent e){
        float amount=0;
        String msg="";
        if(cb1.isSelected()){
            amount+=100;
            msg="Pizza: 100\n";
        }
        if(cb2.isSelected()){
            amount+=30;
            msg+="Burger: 30\n";
        }
        if(cb3.isSelected()){
            amount+=10;
            msg+="Tea: 10\n";
        }
        msg+="-----------------\n";
        JOptionPane.showMessageDialog(this,msg+"Total: "+amount);
    }
    public static void main(String[] args) {
        new CheckBoxExample2();
    }
```



```
kage swing1;
ort javax.swing.*;
ort java.awt.event.*;

lic class ListDuplicateex {
  ListDuplicateex(){
      JFrame f= new JFrame();
      final JLabel label = new JLabel();
      label.setSize(500,100);
      JButton b=new JButton("Show");
      b.setBounds(200,150,80,30);
      final DefaultListModel<String> l1 = new DefaultListModel<>(
      l1.addElement("C");
      l1.addElement("C++");
      l1.addElement("Java");
      l1.addElement("PHP");
      final JList<String> list1 = new JList<>(l1);
      list1.setBounds(100,100, 75,75);
      DefaultListModel<String> l2 = new DefaultListModel<>();
      l2.addElement("Turbo C++");
      l2.addElement("Struts");
      l2.addElement("Spring");
      l2.addElement("YII");
      final JList<String> list2 = new JList<>(l2);
      list2.setBounds(100,200, 75,75);
      f.add(list1); f.add(list2); f.add(b); f.add(label);
      f.setSize(450,450);
      f.setLayout(null);
      f.setVisible(true);
      b.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          String data = "";
          if (list1.getSelectedIndex() != -1) {
            data = "Programming language Selected: " + list1.getSelectedValue();
            label.setText(data);
          }
          if(list2.getSelectedIndex() != -1){
            data += ", FrameWork Selected: ";
            for(Object frame :list2.getSelectedValues()){
              data += frame + " ";
            }
          }
          label.setText(data);
        }
      });
    }
  public static void main(String[] args) {
       new ListDuplicateex();

  }
```


Programming language Selected: Java
C
C++
Java
PHP
Show
Turbo C++
Struts
Spring
YII


RadioButtons
Qualification
Under-Graduate
Graduate
Click


RadioButtons
Message
Under-Graduate
OK

```
1  package swing1;
2
3
4  //Java program to show JRadioButton Example.
6  import java.awt.*;
9
10 class RadioButtonEvent  extends JFrame {
11
12  // Declaration of object of JRadioButton class.
13  JRadioButton jRadioButton1;
14  // Declaration of object of JRadioButton class.
15  JRadioButton jRadioButton2;
16  // Declaration of object of JButton class.
17  JButton jButton;
18  // Declaration of object of ButtonGroup class.
19  ButtonGroup G1;
20  // Declaration of object of  JLabel  class.
21  JLabel L1;
22  // Constructor of Demo class.
```

Radio Button Even class To be continued…

```
23  public RadioButtonEvent()
24  {
25      this.setLayout(null);
26      // Initialization of object of "JRadioButton" class.
27      jRadioButton1 = new JRadioButton();
28      jRadioButton2 = new JRadioButton();
29      jButton = new JButton("Click");
30      G1 = new ButtonGroup();
31      // Initialization of object of " JLabel" class.
32      L1 = new JLabel("Qualification");
33      // setText(...) function is used to set text of radio button.
34      // Setting text of "jRadioButton2".
35      jRadioButton1.setText("Under-Graduate");
36      jRadioButton2.setText("Graduate");
37      jRadioButton1.setBounds(120, 30, 120, 50);
38      // Setting Bounds of "jRadioButton4".
39      jRadioButton2.setBounds(250, 30, 80, 50);
40      // Setting Bounds of "jButton".
41      jButton.setBounds(125, 90, 80, 30);
42      // Setting Bounds of JLabel "L2".
43      L1.setBounds(20, 30, 150, 50);
44         // Adding "jRadioButton2" on JFrame.
45      this.add(jRadioButton1);
46      // Adding "jRadioButton4" on JFrame.
47      this.add(jRadioButton2);
48      // Adding "jButton" on JFrame.
49      this.add(jButton);
50      // Adding JLabel "L2" on JFrame.
51      this.add(L1);
52      // Adding "jRadioButton1" and "jRadioButton3" in a Button Group "G2".
53      G1.add(jRadioButton1);
54      G1.add(jRadioButton2);
55
```

```
    jButton.addActionListener(new ActionListener() {
        // Anonymous class.
        public void actionPerformed(ActionEvent e)
        {
            // Override Method

            // Declaration of String class Objects.
            String qual = " ";

            // If condition to check if jRadioButton2 is selected.
            if (jRadioButton1.isSelected()) {

                qual = "Under-Graduate";
            }

            else if (jRadioButton2.isSelected()) {

                qual = "Graduate";
            }
            else {

                qual = "NO Button selected";
            }

            // MessageDialog to show information selected radion buttons.
            JOptionPane.showMessageDialog(RadioButtonEvent .this, qual);
        }
    });
 }
}
```

```
94  class RadioButton {
95   // Driver code.
96   public static void main(String args[])
97   { // Creating object of demo class.
98       RadioButtonEvent  f = new RadioButtonEvent();
99
100      // Setting Bounds of JFrame.
101      f.setBounds(100, 100, 400, 200);
102
103      // Setting Title of frame.
104      f.setTitle("RadioButtons");
105
106      // Setting Visible status of frame as true.
107      f.setVisible(true);
108  }
109 }
```

Let’s see some of the listeners

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().

How to write ActionListener

1. Implement the ActionListener interface in the class:

   ```
   public class ActionListenerExample Implements ActionListener
   ```

2. Register the component with the Listener:

   ```
   component.addActionListener(instanceOfListenerclass);
   ```

3. Override the actionPerformed() method:

   ```
   public void actionPerformed(ActionEvent e){
           //Write the code here
   }
   ```

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.

Methods of MouseListener interface

- public abstract void mouseClicked(MouseEvent e);
- public abstract void mouseEntered(MouseEvent e);
- public abstract void mouseExited(MouseEvent e);
- public abstract void mousePressed(MouseEvent e);
- public abstract void mouseReleased(MouseEvent e);

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.

Methods of MouseMotionListener

- public abstract void mouseDragged(MouseEvent e);
- public abstract void mouseMoved(MouseEvent e);

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().

itemStateChanged() method

The itemStateChanged() method is invoked automatically whenever you click or unclick on the registered checkbox component.

```
public abstract void itemStateChanged(ItemEvent e);
```

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.

Methods of KeyListener

- public abstract void keyPressed(KeyEvent e);
- public abstract void keyReleased(KeyEvent e);
- public abstract void keyTyped(KeyEvent e);

WindowListener

The Java WindowListener is notified whenever you change the state of window. It is notified against WindowEvent. The WindowListener interface is found in java.awt.event package. It has three methods.

Methods of WindowListener

- public abstract void windowActivated(WindowEvent e);
- public abstract void windowClosed(WindowEvent e);
- public abstract void windowClosing(WindowEvent e);
- public abstract void windowDeactivated(WindowEvent e);
- public abstract void windowDeiconified(WindowEvent e);
- public abstract void windowIconified(WindowEvent e);
- public abstract void windowOpened(WindowEvent e);