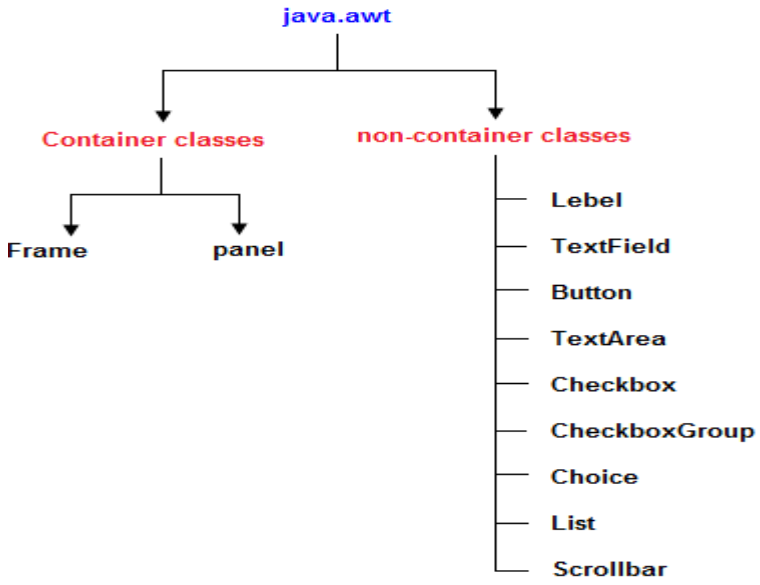
UNIT-V

GUI Programming with Swings

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

AWT Introduction

- 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.
- The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.
- The predefined classes of awt package are classified into following types:



Limitations of AWT

- The buttons of AWT does not support pictures.
- It is heavyweight in nature.
- Two very important components trees and tables are not present.
- Extensibility is not possible as it is platform dependent

Swing Introduction

- Java Swing is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

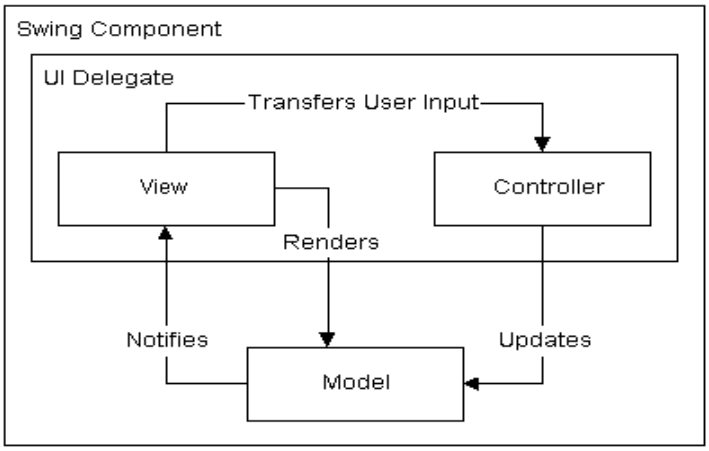
There are many differences between java awt and swing that are given below.

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.

MVC Architecture

- The MVC design pattern consists of three modules model, view and controller.
- **Model** :The model represents the state (data) and business logic of the application. For example-in case of a check box, the model contains a field which indicates whether the box is checked or unchecked.
- **View** :The view module is responsible to display data i.e. it represents the presentation.The view determines how a component has displayed on the screen, including any aspects of view that are affected by the current state of the model.

- **Controller**:The controller determines how the component will react to the user.Controller The controller module acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.



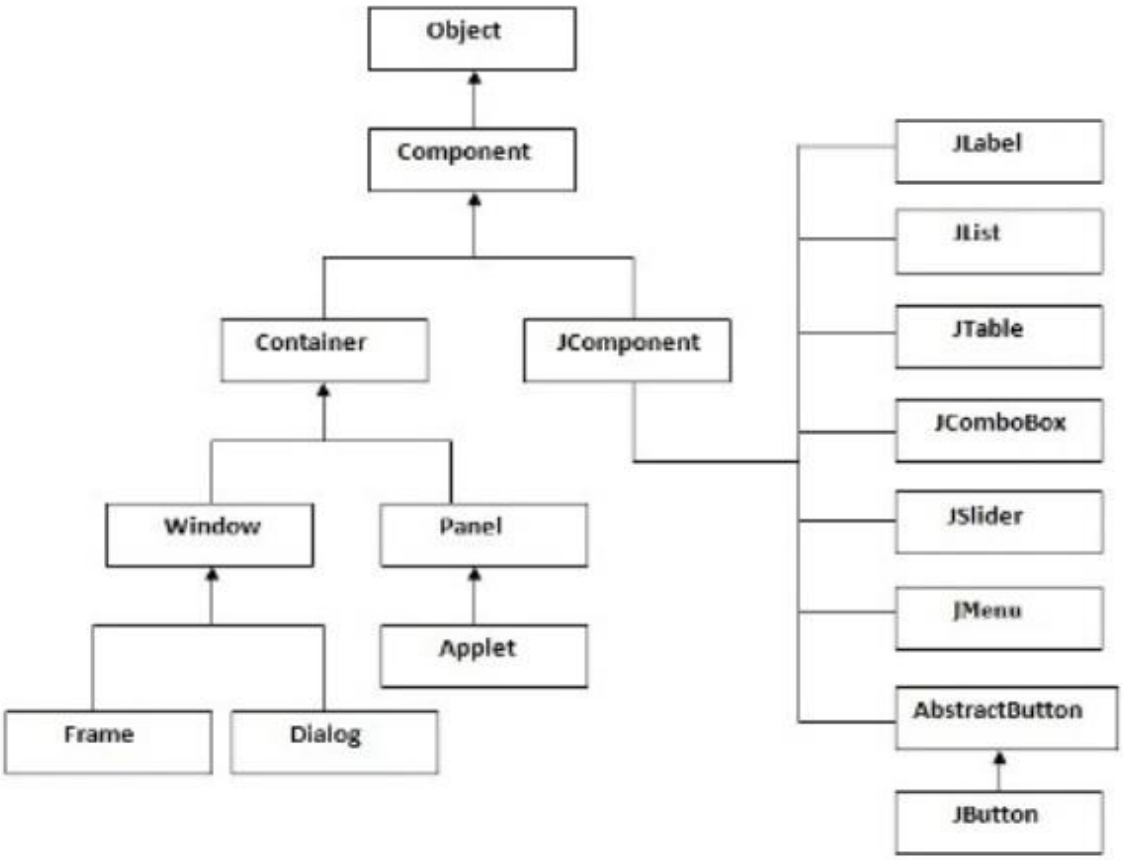
Advantage of MVC Architecture

- Navigation control is centralized Now only controller contains the logic to determine the next page.
- Easy to maintain
- Easy to extend
- Easy to test
- Better separation of concerns

Disadvantage of MVC Architecture

- We need to write the controller code self. If we change the controller code, we need to recompile the class and redeploy the application.

Hierarchy of Java Swing classes



Component Class

- A component is an object having a graphical representation that can be displayed on the screen and that can interact with the user.
- Examples of components are the buttons, checkboxes, and scrollbars of a typical graphical user interface.

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.

public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

JPanel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

JDialog

- The JDialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Dialog class.
- Unlike JFrame, it doesn't have maximize and minimize buttons.

JFrame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

There are two ways to create a frame:

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

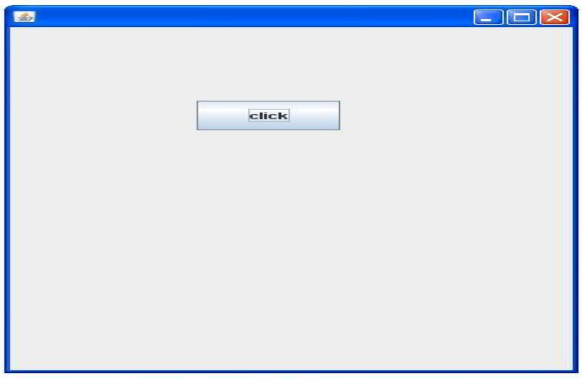
2.By extending Frame class (inheritance)

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

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

```
        JFrame f=new JFrame();
        JButton b=new JButton("click");
        f.add(b);
        f.setSize(400,500);
        f.setVisible(true);
    }
}
```

Output:



2.By extending Frame class (inheritance)

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

```
import javax.swing.*;
public class Simple2 extends JFrame
{   //inheriting JFrame
        JFrame f;
        Simple2()
        {
            JButton b=new JButton("click");    //create button
            add(b);    //adding button on frame
            setSize(400,500);
            setVisible(true);
        }
        public static void main(String[] args)
        {
             new Simple2();
        }
}
```

Java LayoutManagers

The LayoutManagers 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:

1. BorderLayout
2. FlowLayout
3. GridLayout
4. CardLayout
5. GridBagLayout

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:

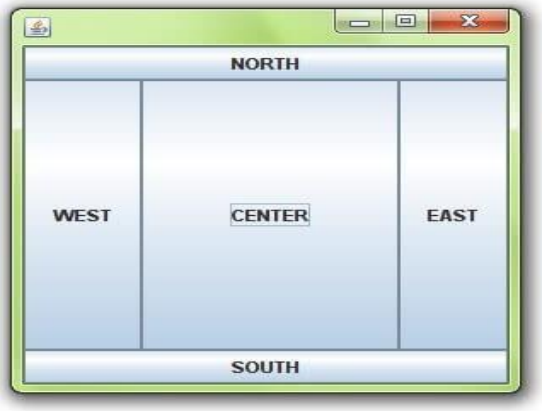
- public static final int NORTH
- public static final int SOUTH
- public static final int EAST
- public static final int WEST
- public static final int CENTER

Example

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

Output:



FlowLayout

- This layout is used to arrange the GUI components in a sequential flow (that means one after another in horizontal way)
- You can also set flow layout of components like flow from left, flow from right.

FlowLayout Left

```
Frame f=new Frame();
f.setLayout(new FlowLayout(FlowLayout.LEFT));
```

FlowLayout Right

```
Frame f=new Frame();
f.setLayout(new FlowLayout(FlowLayout.RIGHT));
```

Example

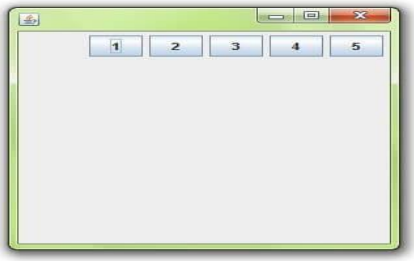
```
import java.awt.*;
import javax.swing.*;
public class MyFlowLayout
{
        MyFlowLayout()
        {
           JFrame f=new JFrame();
           JButton b1=new JButton("1");
           JButton b2=new JButton("2");
           JButton b3=new JButton("3");
           JButton b4=new JButton("4");
           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));

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

Output:



GridLayout

This layout is used to arrange the GUI components in the table format.

Example

```
import java.awt.*;
import javax.swing.*;
public class MyGridLayout
{
    MyGridLayout()
    {
        JFrame 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));

        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args)
```

```
    {
        new MyGridLayout();
    }
}
```

Output

CardLayout

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

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

Example of CardLayout class

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CardLayoutExample extends JFrame implements ActionListener
{
```

```
CardLayout card;
JButton b1,b2,b3;
Container c;
CardLayoutExample()
{
      c=getContentPane();
      card=new CardLayout(40,30);
     //create CardLayout object with 40 hor space and 30 ver space
      c.setLayout(card);
      b1=new JButton("Apple");
      b2=new JButton("Boy");
      b3=new JButton("Cat");
      b1.addActionListener(this);
      b2.addActionListener(this);
      b3.addActionListener(this);
      c.add("a",b1);c.add("b",b2);c.add("c",b3);
}
public void actionPerformed(ActionEvent e)
   {
            card.next(c);
   }
   public static void main(String[] args)
   {
            CardLayoutExample cl=new CardLayoutExample();
            cl.setSize(400,400);
            cl.setVisible(true);
            cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
   }
}
```

Output:

GridBagLayout

- The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.
- The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of constraints object we arrange component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.

Fields

Modifier and Type	Field	Description
double[]	columnWeights	It is used to hold the overrides to the column weights.
int[]	columnWidths	It is used to hold the overrides to the column minimum width.
protected Hashtable<Component,GridBagConstraints>	comptable	It is used to maintains the association between a component and its gridbag constraints.
protected GridBagConstraints	defaultConstraints	It is used to hold a gridbag constraints instance containing the default values.
protected GridBagLayoutInfo	layoutInfo	It is used to hold the layout information for the gridbag.
protected static int	MAXGRIDSIZE	No longer in use just for backward compatibility
protected static int	MINSIZE	It is smallest grid that can be laid out by the grid bag layout.
protected static int	PREFERREDSIZE	It is preferred grid size that can be laid out by the grid bag layout.
int[]	rowHeights	It is used to hold the overrides to the row minimum heights.
double[]	rowWeights	It is used to hold the overrides to the row weights.

Useful Methods

Modifier and Type	Method	Description
void	addLayoutComponent(Component comp, Object constraints)	It adds specified component to the layout, using the specified constraints object.
void	addLayoutComponent(String name, Component comp)	It has no effect, since this layout manager does not use a per-component string.
protected void	adjustForGravity(GridBagConstraints constraints, Rectangle r)	It adjusts the x, y, width, and height fields to the correct values depending on the constraint geometry and pads.
protected void	arrangeGrid(Container parent)	Lays out the grid.
protected void	ArrangeGrid(Container parent)	This method is obsolete and supplied for backwards compatibility
GridBagConstraints	getConstraints(Component comp)	It is for getting the constraints for the specified component.
float	getLayoutAlignmentX(Container parent)	It returns the alignment along the x axis.
float	getLayoutAlignmentY(Container parent)	It returns the alignment along the y axis.
int[][]	getLayoutDimensions()	It determines column widths and row heights for the layout grid.
protected GridBagLayoutInfo	getLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.

protected GridBagLayoutInfo	GetLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.
Point	getLayoutOrigin()	It determines the origin of the layout area, in the graphics coordinate space of the target container.
double[][]	getLayoutWeights()	It determines the weights of the layout grid's columns and rows.
protected Dimension	getMinSize(Container parent, GridBagLayoutInfo info)	It figures out the minimum size of the master based on the information from getLayoutInfo.
protected Dimension	GetMinSize(Container parent, GridBagLayoutInfo info)	This method is obsolete and supplied for backwards compatibility only

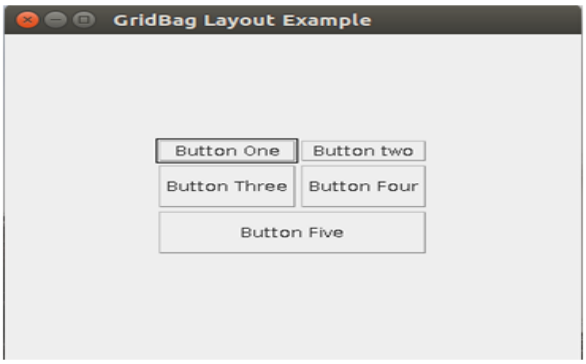
Example

```
import java.awt.Button;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import javax.swing.*;
public class GridBagLayoutExample extends JFrame
{
   public static void main(String[] args)
  {
        GridBagLayoutExample a = new GridBagLayoutExample();
  }
  public GridBagLayoutExample()
  {
        GridBagLayoutgrid = new GridBagLayout();
        GridBagConstraints gbc = new GridBagConstraints();
        setLayout(grid);
        setTitle("GridBag Layout Example");
        GridBagLayout layout = new GridBagLayout();
        this.setLayout(layout);
```

```
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridx = 0;
        gbc.gridy = 0;
        this.add(new Button("Button One"), gbc);
        gbc.gridx = 1;
        gbc.gridy = 0;
        this.add(new Button("Button two"), gbc);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.ipady = 20;
        gbc.gridx = 0;
        gbc.gridy = 1;
        this.add(new Button("Button Three"), gbc);
        gbc.gridx = 1;
        gbc.gridy = 1;
        this.add(new Button("Button Four"), gbc);
        gbc.gridx = 0;
        gbc.gridy = 2;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridwidth = 2;
        this.add(new Button("Button Five"), gbc);
        setSize(300, 300);
        setPreferredSize(getSize());
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

}
```

Output:



Event Handling

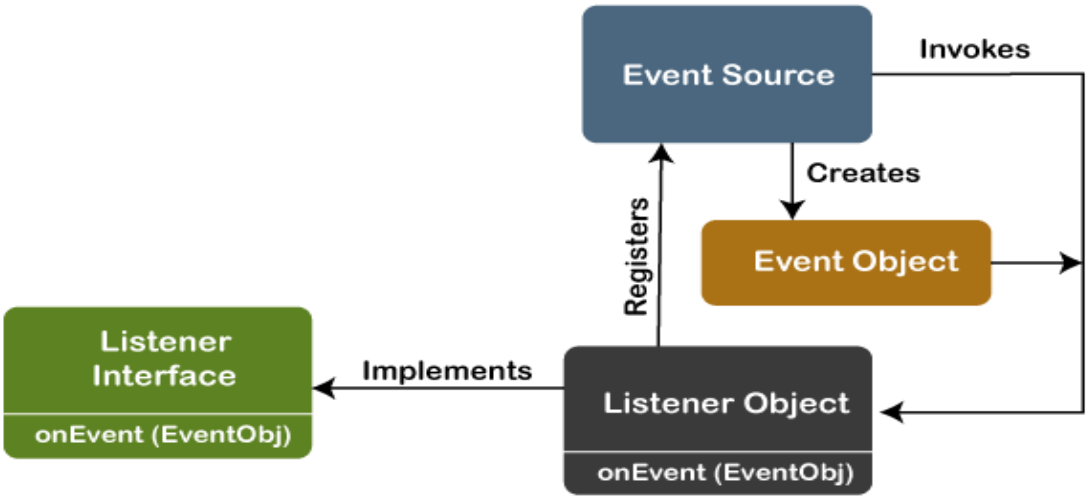
- In general we can not perform any operation on dummy GUI screen even any button click or select any item.
- To perform some operation on these dummy GUI screen you need some predefined classes and interfaces.
- All these type of classes and interfaces are available in **java.awt.event** package.
- Changing the state of an object is known as an **event**.
- The process of handling the request in GUI screen is known as **event handling** (event represent an action). It will be changes component to component.

Note: In event handling mechanism event represent an action class and Listener represent an interface. Listener interface always contains abstract methods so here you need to write your own logic.

Delegation Event Model in Java

- The Delegation Event model is defined to handle events in GUI programming languages. The GUI stands for Graphical User Interface, where a user graphically/visually interacts with the system.
- The GUI programming is inherently event-driven; whenever a user initiates an activity such as a mouse activity, clicks, scrolling, etc., each is known as an event that is mapped to a code to respond to functionality to the user. This is known as event handling.

The below image demonstrates the event processing.



In this model, a source generates an event and forwards it to one or more listeners. The listener waits until it receives an event. Once it receives the event, it is processed by the listener and returns it.

Basically, an Event Model is based on the following three components:

- Events
- Events Sources
- Events Listeners

Events

The Events are the objects that define state change in a source. An event can be generated as a reaction of a user while interacting with GUI elements.

Some of the event generation activities are moving the mouse pointer, clicking on a button, pressing the keyboard key, selecting an item from the list, and so on. We can also consider many other user operations as events.

Event Sources

A source is an object that causes and generates an event. It generates an event when the internal state of the object is changed. The sources are allowed to generate several different types of events.

A source must register a listener to receive notifications for a specific event. Each event contains its registration method. Below is an example:

Example

public void addTypeListener (TypeListener e1)

- From the above syntax, the Type is the name of the event, and e1 is a reference to the event listener.
- For example, for a keyboard event listener, the method will be called as **addKeyListener()**.
- For the mouse event listener, the method will be called as **addMouseMotionListener()**.
- When an event is triggered using the respected source, all the events will be notified to registered listeners and receive the event object.
- This process is known as event multicasting.

Event Listeners

It is also known as event handler.Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received , the listener process the event and then returns.

Event classes and Listener interfaces

Event Classes	Description	Listener 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 exit 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
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 orset 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

Registration Methods

For registering the component with the Listener, many classes provide the registration methods.

For example:

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)
{
}
```

Steps to perform Event Handling

Following steps are required to perform event handling:

- Implement the Listener interface and overrides its methods
- Register the component with the Listener
- The User clicks the button and the event is generated.
- Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
- Event object is forwarded to the method of registered listener class.
- The method is now get executed and returns.

Syntax to Handle the Event

```
class className implements XXXListener
{
 .......
 .......
}
addcomponentobject.addXXXListener(this);
.......
// override abstract method of given interface and write proper logic
public void methodName(XXXEvent e)
{
  .......
  .......
}
    .......
}
```

Event Handling for Mouse

For handling event for mouse you need MouseEvent class and MouseListener interface.

GUI Component	Event class	Listener Interface
Mouse	MouseEvent	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

The signature of 5 methods found in MouseListener interface are given below:

1. public abstract void mouseClicked(MouseEvent e);
2. public abstract void mouseEntered(MouseEvent e);
3. public abstract void mouseExited(MouseEvent e);
4. public abstract void mousePressed(MouseEvent e);
5. public abstract void mouseReleased(MouseEvent e);

Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener
{
        Label l;
        MouseListenerExample()
        {
            addMouseListener(this);
            l=new Label();
            add(l);
            setSize(300,300);
            setVisible(true);
        }
        public void mouseClicked(MouseEvent e)
        {
            l.setText("Mouse Clicked");
        }
        public void mouseEntered(MouseEvent e)
        {
            l.setText("Mouse Entered");
        }
        public void mouseExited(MouseEvent e)
        {
            l.setText("Mouse Exited");
        }
        public void mousePressed(MouseEvent e)
        {
            l.setText("Mouse Pressed");
        }
```

```
    public void mouseReleased(MouseEvent e)
    {
        l.setText("Mouse Released");
    }
    public static void main(String[] args)
    {
        new MouseListenerExample();
    }
}
```

Output:

Event Handling for Keyboard

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 interface

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

Example

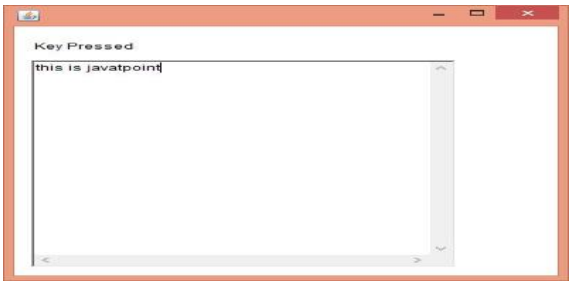
```
import java.awt.*;
import java.awt.event.*;
public class KeyListenerExample extends Frame implements KeyListener
{
    Label l;
    TextArea area;
    KeyListenerExample()
    {
```

```
                    l=new Label();
                    l.setBounds(20,50,100,20);
                    area=new TextArea();
                    area.setBounds(20,80,300, 300);

                    area.addKeyListener(this);

                   add(l);
                   add(area);
                    setSize(400,400);
                    setLayout(null);
                    setVisible(true);
  }
  public void keyPressed(KeyEvent e)
 {
       l.setText("Key Pressed");
  }
  public void keyReleased(KeyEvent e)
  {
       l.setText("Key Released");
  }
  public void keyTyped(KeyEvent e)
 {
       l.setText("Key Typed");
  }
  public static void main(String[] args)
  {
       new KeyListenerExample();
  }
}
```

Output:



Adapter Classes

- In a program, when a listener has many abstract methods to override, it becomes complex for the programmer to override all of them.
- For example, for closing a frame, we must override seven abstract methods of WindowListener, but we need only one method of them.
- For reducing complexity, Java provides a class known as "adapters" or adapter class.
- Adapters are abstract classes, that are already being overriden.

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

Java WindowAdapter Example

```
import java.awt.*;
import java.awt.event.*;
public class AdapterExample
{
        Frame f;
        AdapterExample()
        {
            f=new Frame("Window Adapter");
            f.addWindowListener(new WindowAdapter()
            {
                public void windowClosing(WindowEvent e)
```

```
                    {
                        f.dispose();
                    }
                });
                f.setSize(400,400);
                f.setVisible(true);
            }
        public static void main(String[] args)
        {
            new AdapterExample();
        }
}
```

Inner classes

- Inner class means one class which is a member of another class.
- We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Syntax of Inner class

```
class Outer_class
{
    //code
    class Inner_class
    {
        //code
    }
}
```

Types of Inner classes

There are four types of inner classes.

1. Member Inner class
2. Local inner classes
3. Anonymous inner classes
4. Static nested classes

1.Member inner class

A non-static class that is created inside a class but outside a method is called member inner class.

Syntax:

```
class Outer
{
    //code
    class Inner
    {
        //code
    }
}
```

Example

```
class TestMemberOuter
{
        private int data=30;
        class Inner
        {
         void msg()
         {
              System.out.println("data is "+data);
        }
}
public static void main(String args[])
{
        TestMemberOuter obj=new TestMemberOuter();
        TestMemberOuter.Inner in=obj.new Inner();
        in.msg();
}
}
```

2.Anonymous inner class

- In Java, a class can contain another class known as nested class. It's possible to create a nested class without giving any name.
- A nested class that doesn't have any name is known as an anonymous class.
- An anonymous class must be defined inside another class. Hence, it is also known as an anonymous inner class.

Example

```
abstract class Person
{
      abstract void eat();
}
class TestAnonymousInner
{
          public static void main(String args[])
          {
                    Person p=new Person()
                    {
                        void eat()
                        {
                             System.out.println("nice fruits");
                        }
                    };
                    p.eat();
          }
 }
```

3.Local inner class

- A class i.e. created inside a method is called local inner class in java.
- If you want to invoke the methods of local inner class, you must instantiate this class inside the method.

Example

```
public class localInner
{
          private int data=30;
          void display()
          {
              class Local
              {
                    void msg()
                    {
                        System.out.println(data);
                    }
              }
```

```
        Local l=new Local();
        l.msg();
    }
    public static void main(String args[])
    {
            localInner obj=new localInner();
            obj.display();
    }
}
```

4.static nested class

- A static class i.e. created inside a class is called static nested class in java. It cannot access non-static data members and methods. It can be accessed by outer class name.
- It can access static data members of outer class including private.
- Static nested class cannot access non-static (instance) data member or method.

Example

```
class TestOuter
{
    static int data=30;
    static class Inner
    {
        void msg()
        {
         System.out.println("data is "+data);
         }
    }
    public static void main(String args[])
    {
        TestOuter.Inner obj=new TestOuter.Inner();
        obj.msg();
    }
}
```

In this example, you need to create the instance of static nested class because it has instance method msg(). But you don't need to create the object of Outer class because nested class is static and static properties, methods or classes can be accessed without object.

Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

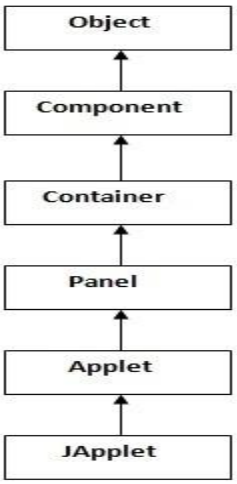
Advantages of Applet

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plugin is required at client browser to execute applet.

Hierarchy of Applet



As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

Lifecycle methods for Applet

The **java.applet.Applet** class provides 4 life cycle methods and java.awt.Component class provides 1 life cycle method for an applet.

java.applet.Applet class

For creating any applet **java.applet.Applet** class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

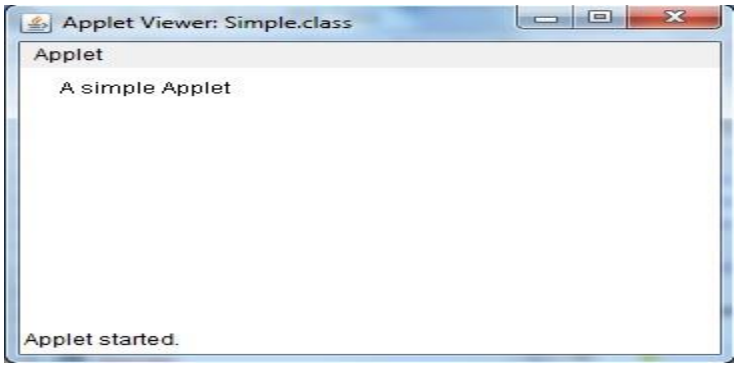
Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
  public void paint(Graphics g)
   {
     g.drawString("A simple Applet",20,20);
   }
 }
```

myapplet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```



Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
public void paint(Graphics g)
{
g.drawString("welcome to applet",150,150);
}
}
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

Parameter in Applet

We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named getParameter().

Syntax:

public String getParameter(String parameterName)

Example of using parameter in Applet:

```
import java.applet.Applet;
import java.awt.Graphics;
public class UseParam extends Applet
{
        public void paint(Graphics g)
        {
                String str=getParameter("msg");
                g.drawString(str,50, 50);
        }
}
```

myapplet.html

```
<html>
<body>
<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>
</body>
</html>
```

Applets and Applications

Java Application	Java Applet
Applications are just like a Java programs that can be execute independently without using the web browser.	Applets are small Java programs that are designed to be included with the HTML web document. They require a Java-enabled web browser for execution.

Java Application	Java Applet
Application program requires a main function for its execution.	Applet does not require a main function for its execution.
Java application programs have the full access to the local file system and network.	Applets don't have local disk and network access.
Applications can access all kinds of resources available on the system.	Applets can only access the browser specific services. They don't have access to the local system.
Applications can executes the programs from the local system.	Applets cannot execute programs from the local machine.
An application program is needed to perform some task directly for the user.	An applet program is needed to perform small tasks or the part of it.

Creating a Swing Applet

- So far, we have created the applets based on AWT(Abstract Window Toolkit) by extending the Applet class of the awt package.
- We can even create applets based on the Swing package.
- In order to create such applets, we must extend JApplet class of the swing package. JApplet extends Applet class, hence all the features of Applet class are available in JApplet as well, including JApplet's own Swing based features.
- Swing applets provides an easier to use user interface than AWT applets.

Example

```
import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
public class EventJApplet extends JApplet implements ActionListener
{
        JButton b;
        JTextField tf;
```

```
        public void init()
        {
                tf=new JTextField();
                tf.setBounds(30,40,150,20);
                b=new JButton("Click");
                b.setBounds(80,150,70,40);
                add(b);add(tf);
                b.addActionListener(this);
                setLayout(null);
        }
        public void actionPerformed(ActionEvent e)
        {
               tf.setText("Welcome");
        }
}
```

In the above example, we have created all the controls in init() method because it is invoked only once.

myapplet.html

```
<html>
<body>
<applet code="EventJApplet.class" width="300" height="300">
</applet>
</body>
</html>
```

Painting in Swing

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.

5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example

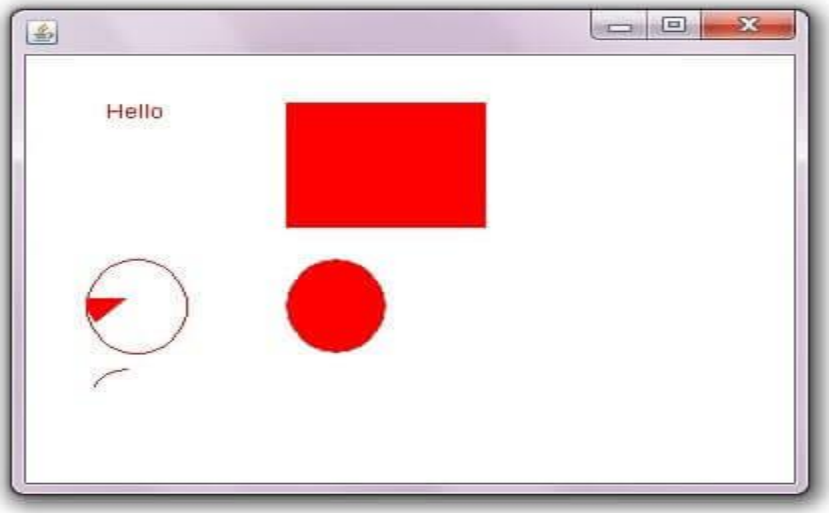
```
import java.awt.*;
import javax.swing.JFrame;
public class DisplayGraphics extends Canvas
{
  public void paint(Graphics g)
  {
        g.drawString("Hello",40,40);
        setBackground(Color.WHITE);
        g.fillRect(130, 30,100, 80);
        g.drawOval(30,130,50, 60);
        setForeground(Color.RED);
        g.fillOval(130,130,50, 60);
        g.drawArc(30, 200, 40,50,90,60);
        g.fillArc(30, 130, 40,50,180,40);

  }
  public static void main(String[] args)
  {
          DisplayGraphics m=new DisplayGraphics();
          JFrame f=new JFrame();
          f.add(m);
```

```
        f.setSize(400,400);
        //f.setLayout(null);
        f.setVisible(true);
    }

}
```

Output:



Exploring Swing Controls

Java 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.

Commonly used Methods

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	It sets the alignment of the label's contents along the X

alignment)	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.

Example

```
import javax.swing.*;
class LabelExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("Label Example");
        JLabel l1,l2;
        l1=new JLabel("First Label.");
        l1.setBounds(50,50, 100,30);
        l2=new JLabel("Second Label.");
        l2.setBounds(50,100, 100,30);
        f.add(l1); f.add(l2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output:

JTextField

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

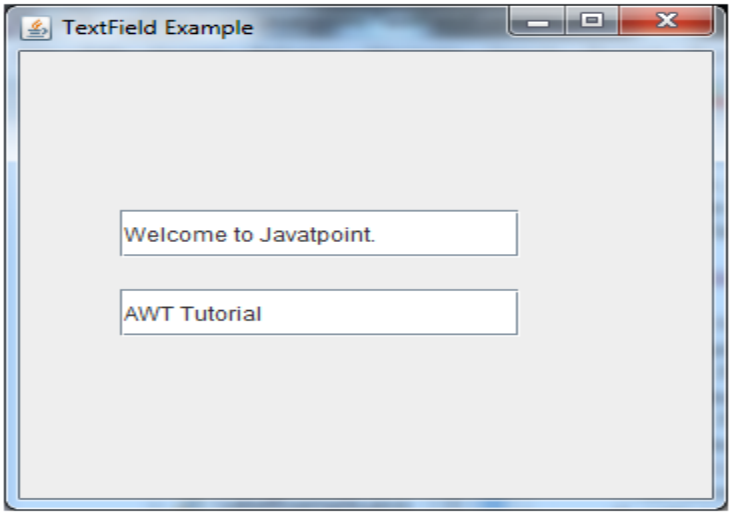
Commonly used Methods:

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.

Example

```
import javax.swing.*;
class TextFieldExample
{
public static void main(String args[])
  {
  JFrame f= new JFrame("TextField Example");
  JTextField t1,t2;
  t1=new JTextField("Welcome to Javatpoint.");
  t1.setBounds(50,100, 200,30);
  t2=new JTextField("AWT Tutorial");
  t2.setBounds(50,150, 200,30);
  f.add(t1);
  f.add(t2);
  f.setSize(400,400);
  f.setLayout(null);
  f.setVisible(true);
  }
  }
```

Output:



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 Methods of AbstractButton class:

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.

Example

```
import javax.swing.*;
public class ButtonExample
{
    public static void main(String[] args)
    {
        JFrame f=new JFrame("Button Example");
        JButton b=new JButton("Click Here");
        b.setBounds(50,100,95,30);
        f.add(b);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output:

Java JToggleButton

JToggleButton is used to create toggle button, it is two-states button to switch on or off.

Methods

Modifier and Type	Method	Description
AccessibleContext	getAccessibleContext()	It gets the AccessibleContext associated with this JToggleButton.

String	getUIClassID()	It returns a string that specifies the name of the l&f class that renders this component.
protected String	paramString()	It returns a string representation of this JToggleButton.
void	updateUI()	It resets the UI property to a value from the current look and feel.

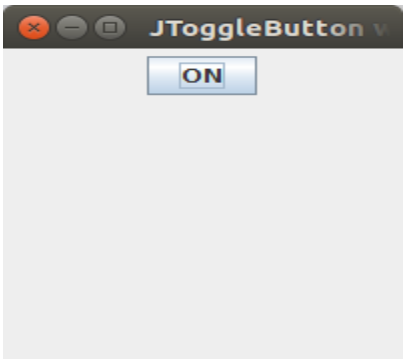
Example

```
import java.awt.FlowLayout;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import javax.swing.JFrame;
import javax.swing.JToggleButton;

public class JToggleButtonExample extends JFrame implements ItemListener
{
   public static void main(String[] args)
  {
      new JToggleButtonExample();
   }
   private JToggleButton button;
   JToggleButtonExample()
   {
      setTitle("JToggleButton with ItemListener Example");
      setLayout(new FlowLayout());
      setJToggleButton();
      setAction();
      setSize(200, 200);
      setVisible(true);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
   private void setJToggleButton()
  {
      button = new JToggleButton("ON");
      add(button);
   }
```

```
    private void setAction()
   {
        button.addItemListener(this);
    }
    public void itemStateChanged(ItemEvent eve)
    {
       if (button.isSelected())
          button.setText("OFF");
       else
          button.setText("ON");
    }
}
```

Output



JCheckBox

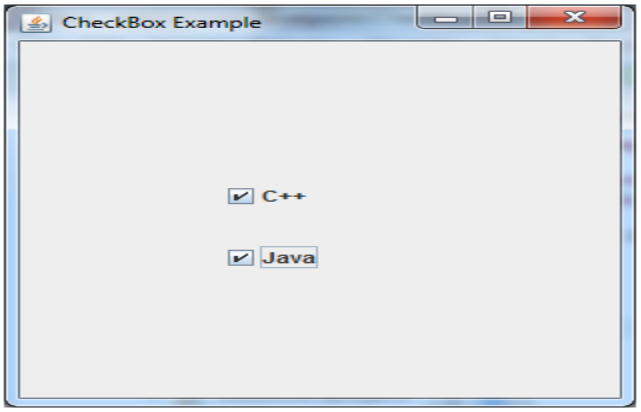
The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits JToggleButton class.

Example

```
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();
        }
}
```

Output:



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.
- It should be added in ButtonGroup to select one radio button only.

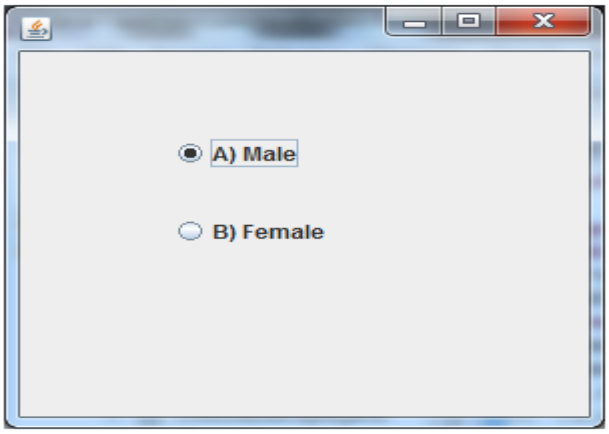
Example

```
import javax.swing.*;
public class RadioButtonExample
{

    RadioButtonExample()
    {
        JFrame f=new JFrame();
```

```
        JRadioButton r1=new JRadioButton("A) Male");
        JRadioButton r2=new JRadioButton("B) Female");
        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();
    }
}
```

Output:



JTabbedPane

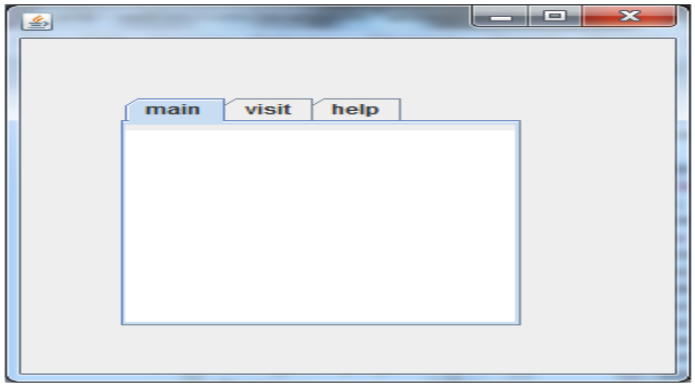
The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.

Example

```
import javax.swing.*;
public class TabbedPaneExample
{
```

```
    TabbedPaneExample()
    {
        JFrame f=new JFrame();
        JTextArea ta=new JTextArea(200,200);
        JPanel p1=new JPanel();
        p1.add(ta);
        JPanel p2=new JPanel();
        JPanel p3=new JPanel();
        JTabbedPane tp=new JTabbedPane();
        tp.setBounds(50,50,200,200);
        tp.add("main",p1);
        tp.add("visit",p2);
        tp.add("help",p3);
        f.add(tp);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String[] args)
    {
       new TabbedPaneExample();
    }
}
```

Output:



JScrollPane

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.

Useful Methods

Modifier	Method	Description
void	setColumnHeaderView(Component)	It sets the column header for the scroll pane.
void	setRowHeaderView(Component)	It sets the row header for the scroll pane.
void	setCorner(String, Component)	It sets or gets the specified corner. The int parameter specifies which corner and must be one of the following constants defined in ScrollPaneConstants: UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER, LOWER_LEFT_CORNER, LOWER_RIGHT_CORNER, LOWER_LEADING_CORNER, LOWER_TRAILING_CORNER, UPPER_LEADING_CORNER, UPPER_TRAILING_CORNER.
Component	getCorner(String)	
void	setViewportView(Component)	Set the scroll pane's client.

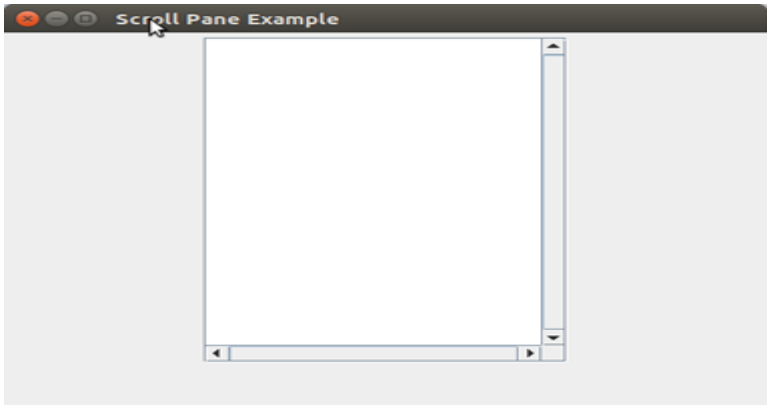
Example

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JtextArea;
public class JScrollPaneExample
{
   private static final long serialVersionUID = 1L;
   private static void createAndShowGUI() {
   // Create and set up the window.
   final JFrame frame = new JFrame("Scroll Pane Example");

      // Display the window.
      frame.setSize(500, 500);
      frame.setVisible(true);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      // set flow layout for the frame
```

```
        frame.getContentPane().setLayout(new FlowLayout());
        JTextArea textArea = new JTextArea(20, 20);
        JScrollPane scrollableTextArea = new JScrollPane(textArea);
        scrollableTextArea.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_AL
WAYS);
        scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
        frame.getContentPane().add(scrollableTextArea);
    }
    public static void main(String[] args)
    {
        javax.swing.SwingUtilities.invokeLater(new Runnable()
        {
            public void run()
            {
                createAndShowGUI();
            }
        });
    }
}
```

Output:



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.

Commonly used Methods:

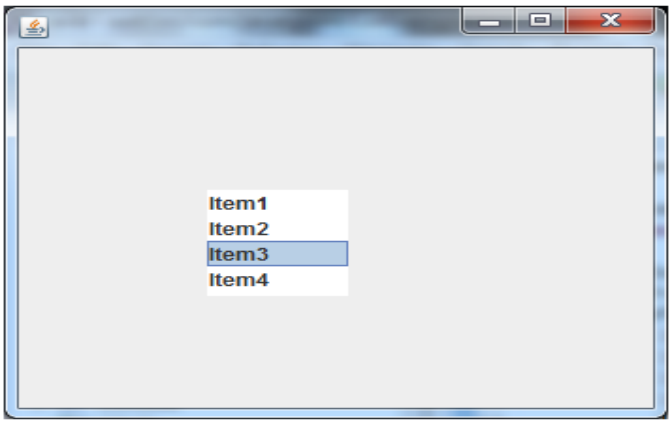
Methods	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return 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.

Example

```
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();
 }
}
```

Output:



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.

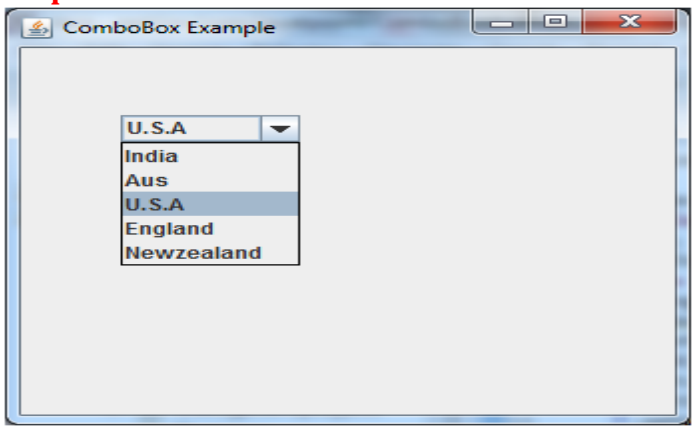
Commonly used Methods:

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.

Example

```
import javax.swing.*;
public class ComboBoxExample
{
        ComboBoxExample()
        {
             JFrame 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();
        }
}
```

Output:



Swing Menus

JMenuBar, JMenu and JMenuItem

The JMenuBar class is used to display menubar on the window or frame. It may have several menus.

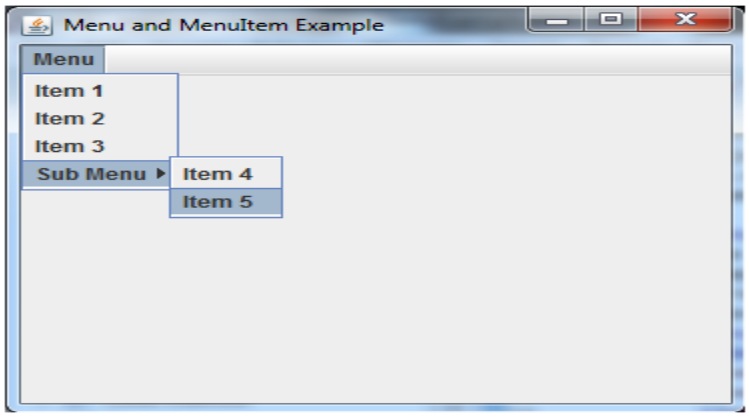
The object of JMenu class is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class.

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.

JMenuItem and JMenu Example

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

Output:



JDialog

- The JDialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Dialog class.
- Unlike JFrame, it doesn't have maximize and minimize buttons.

Example

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DialogExample
{
   private static JDialog d;
   DialogExample()
   {
            JFrame f= new JFrame();
            d = new JDialog(f , "Dialog Example", true);
            d.setLayout( new FlowLayout() );
            JButton b = new JButton ("OK");
            b.addActionListener ( new ActionListener()
            {
               public void actionPerformed( ActionEvent e )
               {
                  DialogExample.d.setVisible(false);
               }
            });
            d.add( new JLabel ("Click button to continue."));
            d.add(b);
```

```
            d.setSize(300,300);
            d.setVisible(true);
        }
    public static void main(String args[])
    {
        new DialogExample();
    }
}
```

Output: