UNIT V

EVENT DRIVEN PROGRAMMING

Syllabus

Graphics programming - Frame – Components - working with 2D shapes - Using color, fonts, and images - Basics of event handling - event handlers - adapter classes - actions - mouse events - AWT event hierarchy - Introduction to Swing – layout management - Swing Components – Text Fields , Text Areas – Buttons- Check Boxes – Radio Buttons – Lists- choices- Scrollbars – Windows –Menus – Dialog Boxes.

5.1 Introduction to Graphics Programming

There are current three sets of Java APIs for graphics programming

- AWT (Abstract Windowing Toolkit)
- Swing
- JavaFX

AWT API was introduced in JDK 1.0. Most of the AWT components have become obsolete and should be replaced by newer Swing components. Swing API, a much more comprehensive set of graphics libraries that enhances the AWT, was introduced as part of Java Foundation Classes (JFC) after the release of JDK 1.1. The latest JavaFX, which was integrated into JDK 8, is meant to replace Swing.

5.1.1 AWT Packages

AWT is huge. It consists of 12 packages of 370 classes (Swing is even bigger, with 18 packages of 737 classes as of JDK 8). 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.

Commonly used packages are: **java.awt and java.awt.event.**

The ***java.awt package*** contains the core AWT graphics classes:

- GUI Component classes, such as Button, TextField, and Label.
- GUI Container classes, such as Frame and Panel.
- Layout managers, such as FlowLayout, BorderLayout and GridLayout.
- Custom graphics classes, such as Graphics, Color and Font.

The *java.awt.event package* supports event handling:

- Event classes, such as ActionEvent, MouseEvent, KeyEvent and WindowEvent,
- Event Listener Interfaces, such as ActionListener, MouseListener, MouseMotionListener, KeyListener and WindowListener,
- Event Listener Adapter classes, such as MouseAdapter, KeyAdapter, and WindowAdapter.

5.1.2 Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.

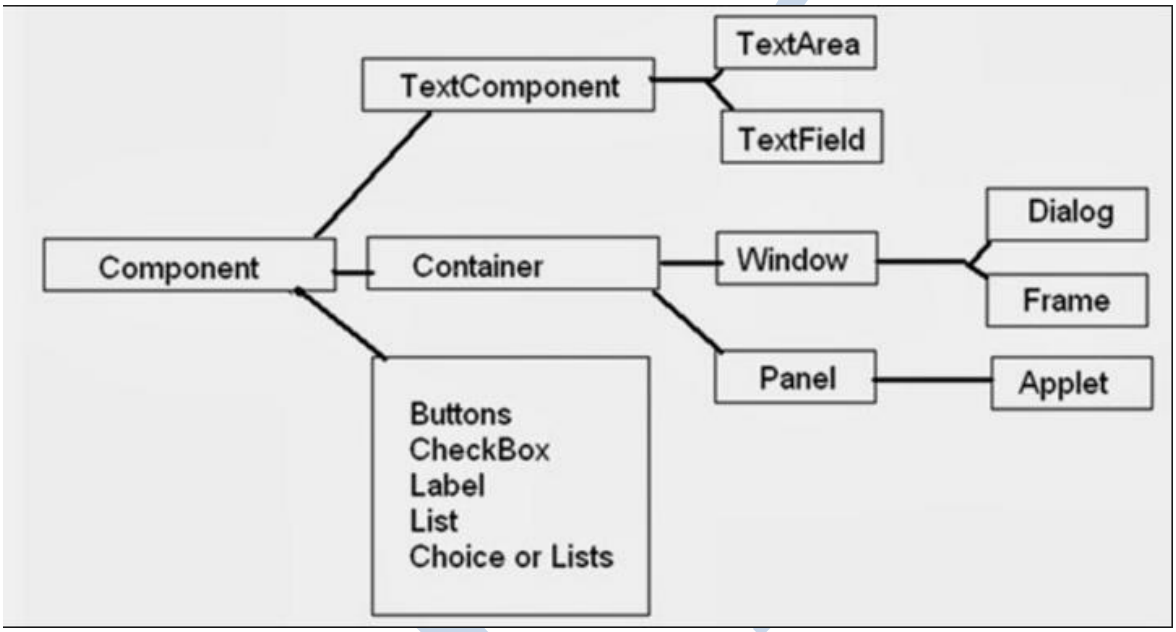
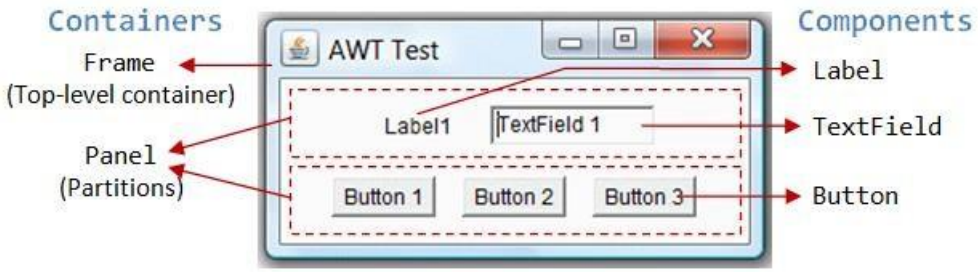


Figure5.1 AWT Hierarchy

5.2 Containers and Components

There are two types of GUI elements:

- Component: Components are elementary GUI entities, such as Button, Label, and TextField.
- Container: Containers, such as Frame and Panel, are used to hold components in a specific layout (such as FlowLayout or GridLayout). A container can also hold sub-containers.



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

In a GUI program, a component must be kept in a container. You need to identify a container to hold the components. Every container has a method called add(Component c). A container (say c) can invoke c.add(aComponent) to add aComponent into itself.

Example:

Panel pnl = new Panel(); → Panel is a container

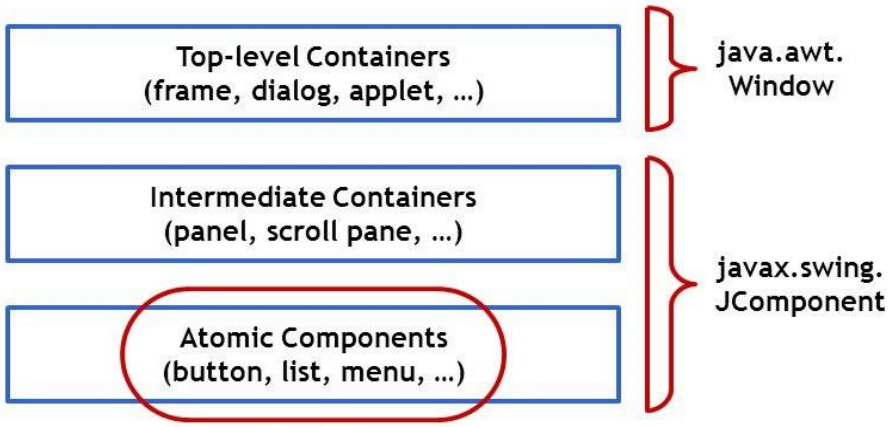
Button btn = new Button("Press"); → Button is a component

pnl.add(btn); → The Panel container adds a Button component

5.2.1 Useful Methods of Component class

Method	Description
add(Component c)	Inserts a component on this component.
setSize(int width,int height)	Sets the size (width and height) of the component.
setLayout(LayoutManager m)	Defines the layout manager for the component.
setVisible(boolean status)	Changes the visibility of the component, default is false.

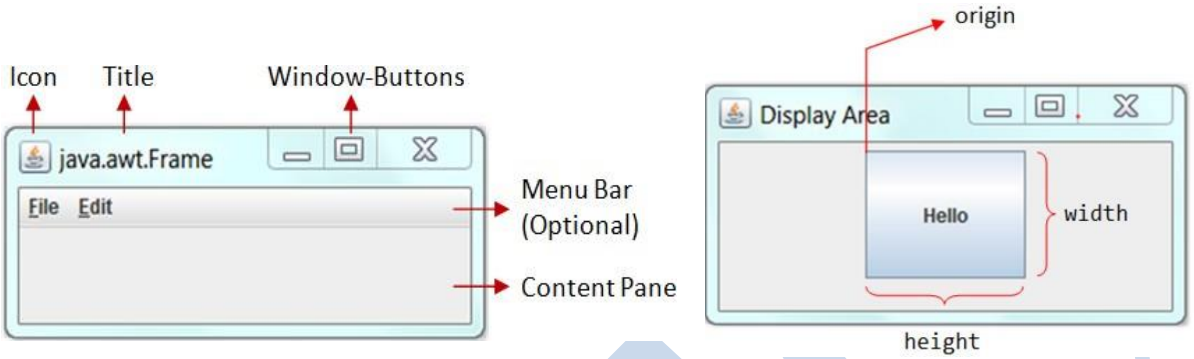
5.2.2 AWT Container Classes



Top – Level Containers: Frame, Dialog and Applet

Each GUI program has a *top-level container*. The commonly-used top-level containers in AWT are Frame, Dialog and Applet.

AWT Frame provides the "main window" for your GUI application. It has a title bar (containing an icon, a title, minimize/maximize/close buttons), an optional menu bar, and the content display area. It uses BorderLayout as default layout manager.



Java GUI program which extends java.awt.Frame

A GUI program is written as a subclass of Frame (the top-level container). This subclass inherits all properties from Frame, e.g., title, icon, buttons, content-pane

```
import java.awt.Frame;   → Using Frame class in package java.awt

public class MyGUIProgram extends Frame
{
   ......

  public MyGUIProgram()   → Constructor to setup the GUI components
   { ..... }
  .......
  public static void main(String[] args)
  {
     MyGUIProgram m = new MyGUIProgram();   → creating an instance
}
}
```

5.2.3 Java AWT Example

To create simple AWT example, you need a frame. There are two ways to create a frame in AWT.

- By extending Frame (Inheritance)
- By creating the object of Frame class (association)

Java AWT program which extends java.awt.Frame

```
import java.awt.*;
class Test extends Frame
{
   Test()
   {
   Button b = new Button("click me");   → Create a Button named "click me"
   b.setBounds(30,100,80,30); → setting button position
   add(b);    → Adding button into frame
   setSize(300,300);  → Frame size 300 width and 300 height
   setLayout(null);  → No layout manager
   setVisible(true);  → Now frame will be visible, by default not visible
   }
   public static void main(String args[])
   {
      First f = new First(); → An instance is created. Default Constructor is invoked
   }
}
```

Java AWT program by using Frame class

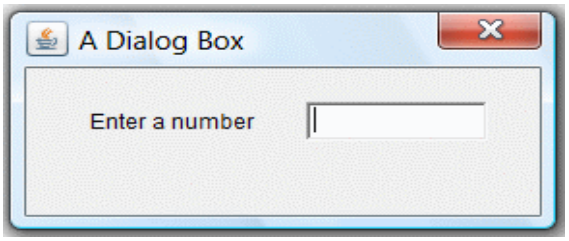
```
import java.awt.*;
class Test
{
   Test()
   {
      Frame f = new Frame();
      Button b = new Button("click me");
      b.setBounds(30,50,80,30);
      f.add(b);
      f.setSize(300,300);
      f.setLayout(null);
      f.setVisible(true);
   }

   public static void main(String args[])
   {
      First f = new First();
   }
}
```

Output:



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



AWT Applet (***in package java.applet***) is the top-level container for an applet, which is a Java program running inside a browser.

Intermediate Containers: Panel and ScrollPane

Secondary containers are placed inside a top-level container or another secondary container.

- ***Panel***: A rectangular box used to adda set of related GUI components.
- ***ScrollPane***: Provides automatic horizontal/vertical scrolling for a single child component.

5.2.4 AWT Component Classes

AWT provides many ready-made and reusable GUI components in package java.awt. They are: Button, TextField, Label, Checkbox, CheckboxGroup (radio buttons) , List, and Choice.



1. AWT GUI Component: java.awt.Label

A ***java.awt.Label*** provides a descriptive text string. Label is used to describe a text field.



java.awt.Label	
AWT Label Class Declaration	***public class Label extends Component implements Accessible***

java.awt.Label	
Constructors	**Label(String str, int alignment);** Construct a Label with the given text String, of the text alignment **Label(String strLabel);** Construct a Label with the given text String **Label();**
Methods	**getText() and setText()** →Used to read and modify the Label's text **addNotify()** →Creates the peer for this label. **getAlignment()** → Gets the current alignment of this label. **setAlignment(int alignment)** → Sets the alignment for this label.
Example	**Label lb = new Label("Enter ID");** → Construct a new Label **add(lb);** → this.add(lblInput) - "this" is a subclass of Frame **lb.setText("Enter password");** → Modify the Label's text string **lb.getText();**
Anonymous Instance	add(new Label("Enter Name: ", Label.RIGHT));

Java Program to implement Label

```
import java.awt.*;
class LabelExample
{
public static void main(String args[])
{
    Frame f = new Frame("Label Example");
    Label l1,l2;
    l1=new Label("First Label.");
    l1.setBounds(50,100, 100,30);
    l2=new Label("Second Label.");
    l2.setBounds(50,150, 100,30);
    f.add(l1);
    f.add(l2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
    }
}
```

Output



2. *AWT GUI Component: java.awt.Button*

A ***java.awt.Button*** is a GUI component that triggers some action on clicking.

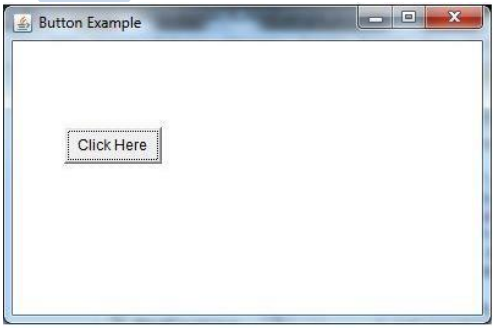


java.awt.Button	
AWT Button Class Declaration	***public class Button extends Component implements Accessible***
Constructors	**Button(String btnLabel);** Construct a Button with the given label **Button();** Construct a Button with empty label
Methods	**getLabel() and setLabel()** → Used to read the current label and modify the label of a button. **addActionListener()** →Adds the specified action listener. **removeActionListener()** → Removes the specified action listener **addNotify()** → Creates the peer of the button. **getActionCommand()** → Returns the command name. **setActionCommand()** → Sets the command name. **getActionListeners()** → Returns an array of all the action listeners.
Example	**Button bt = new Button("Red");** → Declare a Button **add(bt);** → Add the Button **bt.setLabel("Green");** →Change the button's label **bt.getLabel();** → Read the button's label
Anonymous Instance	add(Button("Blue"));

Java Program to implement Button

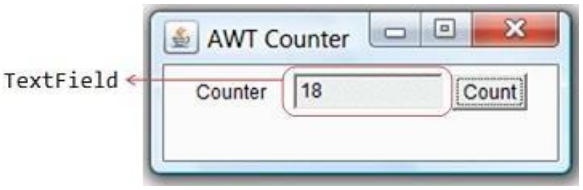
```
import java.awt.*;
public class ButtonExample
{
public static void main(String[] args)
{
    Frame f = new Frame("Button Example");
    Button b = new Button("Click Here");
    b.setBounds(50,100,80,30);
    f.add(b);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
    }
}
```

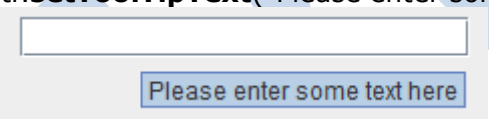
Output



3. *AWT GUI Component: java.awt.TextField*

A ***java.awt.TextField*** is single-line text box for users to enter texts.

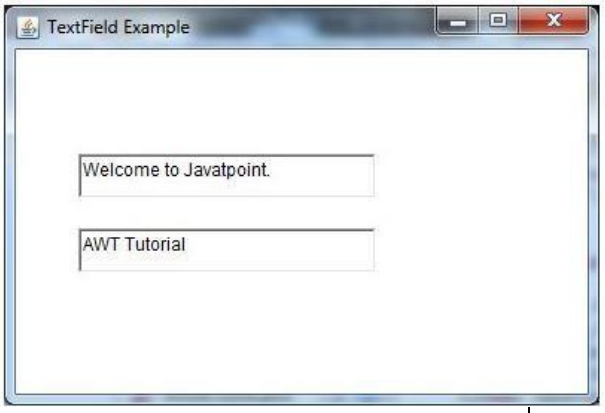


java.awt.TextField	
AWT TextField Class Declaration	***public class TextField extends TextComponent***
Constructors	**TextField(String initialText, int columns);** Construct a TextField with the given string and number of columns. **TextField(String initialText);** Construct a TextField with the given string. **TextField(int columns);** Construct a TextField with the number of columns.
Methods	**getText()** and **setText(String strText);** **setEditable(boolean);** → Set the TextField to editable/non-editable **addActionListener()** → Adds the specified action listener. **removeActionListener()** → Removes the specified action listener **addNotify()** → Creates the TextField's peer. **getActionListeners()** → Returns an array of all the action listeners. **getColumns()** →Gets the number of columns in this text field. **setColumns()** → Sets the number of columns in this text field. **getEchoChar()** → Gets the character that is to be used for echoing. **setEchoChar()** → Sets the echo character for this text field. **getMinimumSize()** → Gets the minumum dimensions for text field. **getPreferredSize()** → Gets the preferred size of this text field.
Example	**TextField tf = new TextField("This is a text", 20);** **tf.setEditable(false) ;** →Set to read-only **frame. add(tf);** → add textfield to a frame
Setting tooltip text	tf.**setToolTipText**("Please enter some text here"); 

Java Program to implement Textfield

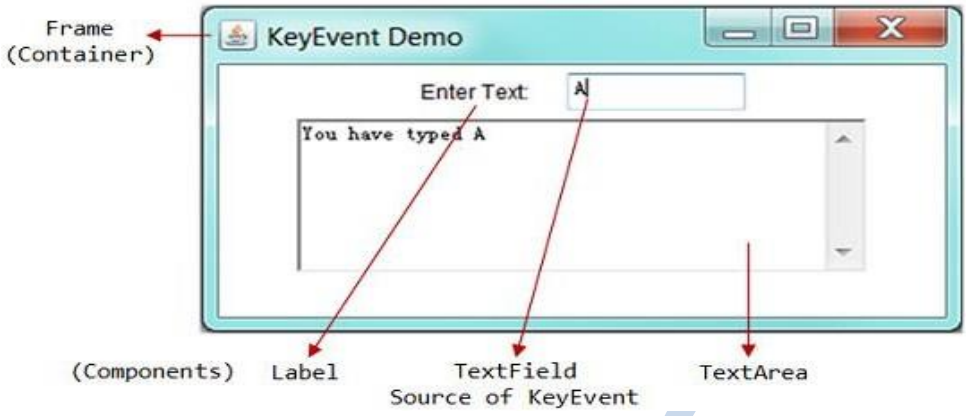
```
import java.awt.*;
class TextFieldExample{
public static void main(String args[])
{
   Frame f = new Frame("TextField Example");
   TextField t1,t2;
   t1 = new TextField("Welcome to Javatpoint.");
   t1.setBounds(50,100, 200,30);
   t2 = new TextField("AWT Tutorial");
   t2.setBounds(50,150, 200,30);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

Output



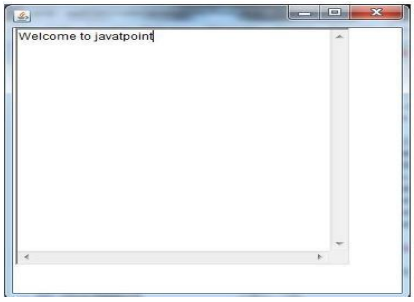
4. AWT GUI Component: java.awt.TextArea

A ***java.awt.TextArea*** is multi-line text box for users to enter texts.



java.awt.TextField	
AWT TextArea Class Declaration	***public class TextArea extends TextComponent***
Constructors	**TextArea()** Constructs a text area. **TextArea(int rows, int columns)** Constructs a text area with the specified number of rows and columns. **TextArea(String text)** Constructs a text area with the specified text. **TextArea(String text, int rows, int columns)** Constructs a text area with the specified text, and with the specified number of rows and columns. **TextArea(String text, int rows, int columns, int scrollbars)** Constructs a text area with the specified text, and with the rows, columns, and with the scroll bars.
Methods	**addNotify()** → Creates the TextArea's peer. **append(str)** → Appends the given text to the text area. **getColumns()** → Returns the number of columns. **getMinimumSize()** → Determines the minimum size. **getPreferredSize()** → Determines the preferred size. **getRows()** → Returns the number of rows. **getScrollbarVisibility()** →Returns an enumerated value of which scroll bars the text area uses. **insert(str, pos)** → Inserts the specified text at the specified position. **replaceRange(String str, int start, int end)** Replaces text between the indicated start and end positions with the specified string. **setColumns()** → Sets the number of columns. **setRows()** → Sets the number of rows.
Example	**TextArea area=new TextArea("Welcome to javatpoint");** **frame. add(area);** → add textarea to a frame

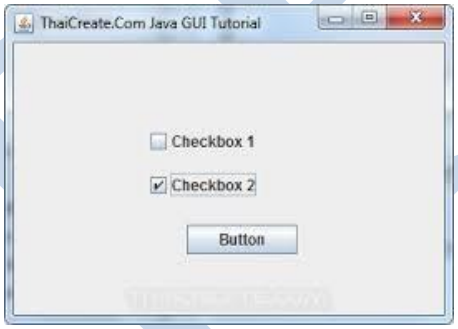
Java Program to implement TextArea




```
import java.awt.*;
public class TextAreaExample
{
  public static void main(String args[])
  {
    Frame f = new Frame();
    TextArea area = new TextArea("Welcome to javatpoint");
    area.setBounds(10,30, 300,300);
    f.add(area);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
  }
}
```

5. *AWT GUI Component: java.Checkbox.*

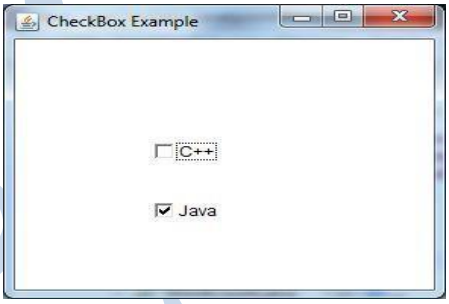
A ***java.awt.Checkbox*** is Checkbox control is used to turn an option on(true) or off (false).



java.awt.Checkbox	
AWT Checkbox Class Declaration	***public class Checkbox extends Component implements ItemSelectable, Accessible***
Constructors	**Checkbox()** Creates a empty check box. **Checkbox(String label)** Creates a check box with the specified label. **Checkbox(String label, boolean state)** Creates a check box with the specified label and set the state. **Checkbox(String label, boolean state, CheckboxGroup group)** Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.

java.awt.Checkbox	
Methods	**addNotify()** → Creates the peer of the Checkbox. **addItemListener()** → Adds the specified item listener. **removeItemListener()** → Removes the specified item listener **getItemListeners()** → Returns an array of item listeners. **getLabel()** → Get the label of this check box. **setLabel(String label)** → Sets check box's label with the string**.** **getCheckboxGroup()** → Determines this check box's group. **setCheckboxGroup()** → Sets this check box's group to the specified check box group. **getState()** → Determine whether check box is in the on or off state. **setState(boolean state)** → Sets the check box to the specified state.
Example	**Checkbox c2 = new Checkbox("Java", true);** **frame. add(c2);** → add checkbox to a frame

Java Program to implement Checkbox




```
import java.awt.*;
public class CheckboxExample
{
  public static void main(String args[])
  {
        Frame f = new Frame("Checkbox Example");
        Checkbox c1 = new Checkbox("C++");
        checkbox1.setBounds(100,100, 50,50);
        Checkbox c2 = new Checkbox("Java", true);
        checkbox2.setBounds(100,150, 50,50);
        f.add(c1);
        f.add(c2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

6. AWT GUI Component: java.CheckboxGroup.

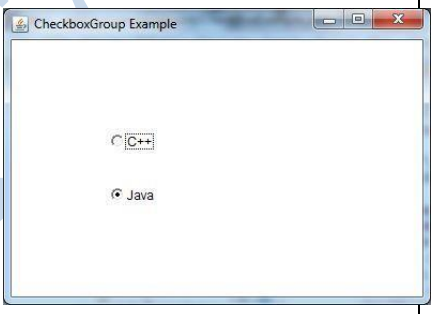
A ***java. awt. CheckboxGroup*** is used to group together a set of Checkbox. At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state.



java.awt.CheckboxGroup	
AWT CheckboxGroup Class Declaration	***public class CheckboxGroup extends Object implements Serializable***
Constructors	**CheckboxGroup()** → Creates a new CheckboxGroup.
Methods	**getSelectedCheckbox()** (or) **getCurrent()** Gets the current choice from the check box group. **setSelectedCheckbox()** (or) **setCurrent()** Sets the currently selected check box to the specified check box. **String toString()** Returns a string representation of this check box group
Example	**CheckboxGroup cbg = new CheckboxGroup();** Checkbox c1 = new Checkbox("C++", **cbg**, false); Checkbox c2 = new Checkbox("Java", **cbg**, true);

Java Program to implement CheckboxGroup

```
import java.awt.*;
public class CheckboxGroupExample
{
   public static void main(String args[])
   {
      Frame f = new Frame("CheckboxGroup Example");
      CheckboxGroup cbg = new CheckboxGroup();
      Checkbox c1 = new Checkbox("C++", cbg, false);
      c1.setBounds(100,100, 50,50);
      Checkbox c2 = new Checkbox("Java", cbg, true);
      c2.setBounds(100,150, 50,50);
      f.add(c1);
      f.add(c2);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
   }
}
```



7. AWT GUI Component: java.Choice.

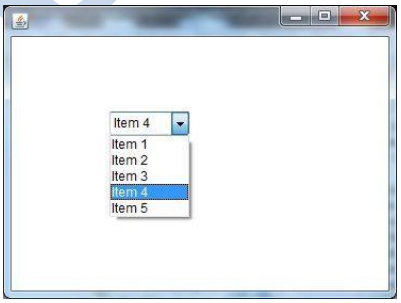
A ***java.awt.Choice*** is used to show popup menu of choices. Choice selected by user is shown on the top of a menu.



java.awt.Choice	
AWT Choice Class Declaration	***public class Choice extends Component implements ItemSelectable, Accessible***
Constructors	**Choice()** → Creates a new choice menu.
Methods	**add(item)** or **addItem(item)**→ Adds an item to this Choice menu. **addNotify()** → Creates the Choice's peer. **countItems()** or **getItemCount()** → Returns the number of items. **getItem(index)** → Gets the string at the specified index. **addItemListener()** → Adds the specified item listener **getItemListeners()** → Returns an array of item listeners. **getSelectedIndex()** → Returns the index of currently selected item. **insert(item, index)** → Inserts the item at the specified position. **remove(position)** → Removes an item at the specified position. **remove(item)** → Removes the first occurrence of item. **removeAll()** → Removes all items from the choice menu. **select(position)** → Sets the selected item at the specified position.
Example	**Choice c = new Choice();** **c**.add("Item 1"); **c**.add("Item 2"); **c**.add("Item 3");

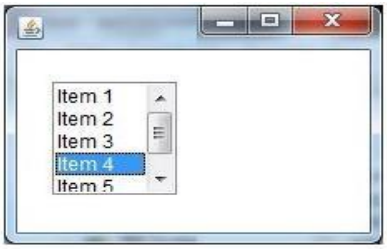
Java Program to implement Choice

```
import java.awt.*;
public class ChoiceExample
{
   public static void main(String args[])
   {
      Frame f = new Frame();
      Choice c = new Choice();
      c.setBounds(100,100, 75,75);
      c.add("Item 1");
      c.add("Item 2");
      c.add("Item 3");
      c.add("Item 4");
      c.add("Item 5");
      f.add(c);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
   }
}
```



8. *AWT GUI Component: java.List.*

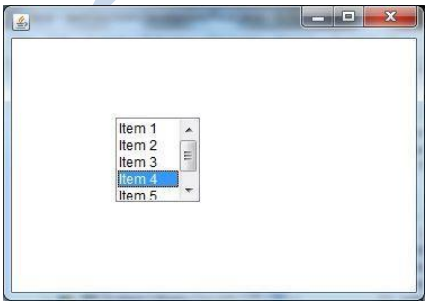
A ***java.awt.List*** represents a list of text items. Here, the user can choose either one item or multiple items.



java.awt.List	
AWT List Class Declaration	***public class List extends Component implements ItemSelectable, Accessible***
Constructors	**List()** → Creates a new scrolling list. **List(rows)** → Creates a list with the specified no. of rows. **List(rows, multipleMode)** → Whether multiple selections allowed.
Methods	**add(item)** or **addItem(String item)** → Adds the specified item to the end of list. **add(item,index)** or **addItem(item, index)** → Adds the specified item in the list at the given index. **addActionListener()** → Adds the specified action listener. **addNotify()** → Creates the peer for the list. **allowsMultipleSelections()** or **isMultipleMode()** → Determines whether this list allows multiple selections. **clear()** or **removeAll()** → Removes all items from this list. **countItems()** or **getItemCount()** → Gets the no. of items in the list. **getItems()** → Gets the items in the list. **getItem(index)** → Gets the item associated with the specified index. **void delItem()** or **remove()** → Removes the item. **replaceItem(String newValue, int index)** → Replaces the item at the specified index with the new string.
Example	**List l1 = new List(3);** → 3 denotes the no. of items in the list **l1.**add("Item 1"); **l1.**add("Item 2"); **l1.**add("Item 3");

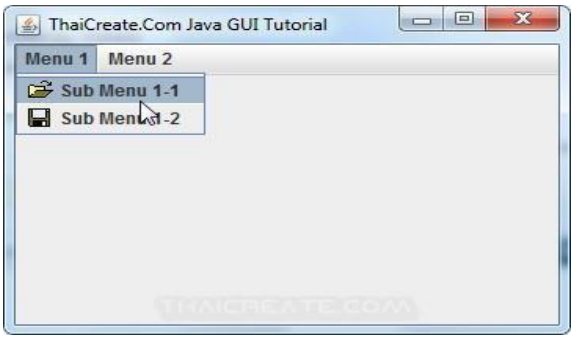
Java Program to implement List

```
import java.awt.*;
public class ListExample
{
   public static void main(String args[])
   {
      Frame f = new Frame();
      List l1 = new List(5);
      l1.setBounds(100,100, 75,75);
      l1.add("Item 1");
      l1.add("Item 2");
      l1.add("Item 3");
      l1.add("Item 4");
      l1.add("Item 5");
      f.add(l1);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
   }
}
```



9. *AWT GUI Component: java.MenuItem and Menu.*

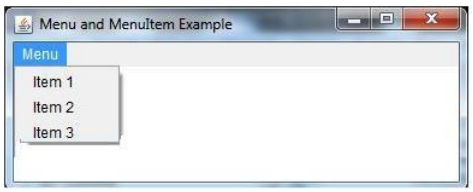
The MenuBar class represents the actual item in a menu. All items in a menu should derive from . ***awt.MenuItem*** .



java.awt.MenuItem	
AWT MenuItem class Declaration	***public class MenuItem extends MenuComponent implements Accessible***
Constructors	**MenuItem()** → Constructs a new MenuItem with an empty label **MenuItem(label)** → Constructs a MenuItem with the specified label
Methods	**addActionListener()** → Adds the specified action listener. **removeActionListener()** → Removes the specified action listener **addNotify()** → Creates the menu item's peer. **getActionListeners()** → Returns an array of action listeners. **getLabel()** → Gets the label for this menu item. **setLabel(label)** → Sets the label with the specified label. **isEnabled()** → Checks whether this menu item is enabled. **setEnabled(boolean b)** or **disable(),enable()** → Check whether this menu item is chosen or not.
Example	**MenuBar mbar = new MenuBar();** **Menu menu1 = new Menu("Menu 1");** **MenuItem** m1 = new **MenuItem**("Item 1"); MenuItem m2 = new MenuItem("Item 2");

Java Program to implement MenuItem

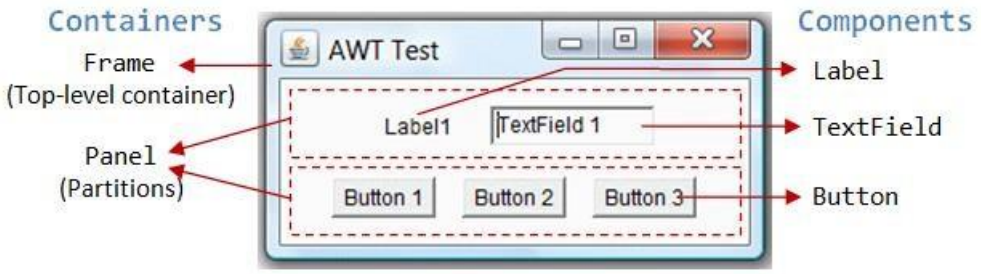
```
import java.awt.*;
class MenuExample
{
   public static void main(String args[])
   {
      Frame f = new Frame("Menu and MenuItem Example");
      MenuBar mb = new MenuBar();
      Menu menu1 = new Menu("Menu");
      MenuItem i1 = new MenuItem("Item 1");
      MenuItem i2 = new MenuItem("Item 2");
      MenuItem i3 = new MenuItem("Item 3");
      menu1.add(i1);
      menu1.add(i2);
      menu1.add(i3);
      mb.add(menu);
      f.setMenuBar(mb);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
   }
}
```



5.2.5 AWT Container Classes

AWT GUI Container: java.Panel

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

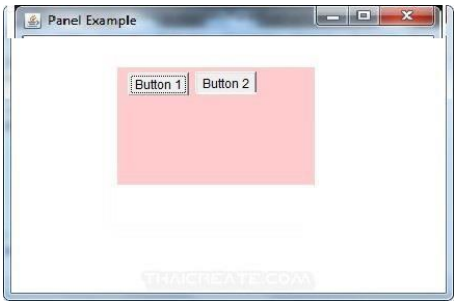


java.awt.Panel	
AWT Panel class Declaration	***public class Panel extends Container implements Accessible***
Constructors	**Panel()** → Creates a new panel. **Panel(layout)** → Creates a new panel with the specified layout.
Methods	**addNotify()** → Creates the Panel's peer. **getAccessibleContext()** → Gets the AccessibleContext associated with this Panel.
Example	**Panel p1 = new Panel();** Button b1= new Button("Button 1"); **p1.add(b1);** → add button to the panel **frame.add(p1);** → add panel to the frame

Java Program to implement Panel

```
import java.awt.*;
public class PanelExample
{
   public static void main(String args[])
   {
      Frame f = new Frame("Panel Example");
      Panel p1 = new Panel();
      p1.setBounds(40,80,200,200);
      p1.setBackground(Color.pink);
      Button b1 = new Button("Button 1");
      b1.setBounds(50,100,80,30);
      Button b2 =new Button("Button 2");
      b2.setBounds(100,100,80,30);
      p1.add(b1);   →  add buttons to the panel
      p1.add(b2);

      f.add(p1);   →  add panel to the frame
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
   }
}
```



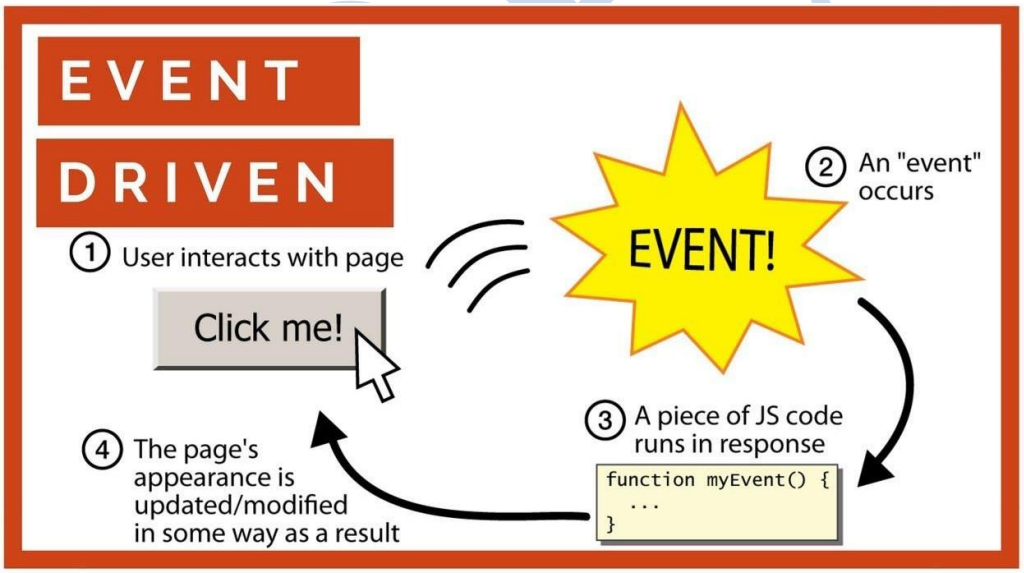
AWT GUI Container: java.Window

The class Window is a top level window with no border and no menubar. It uses BorderLayout as default layout manager.

java.awt.Window	
AWT Window class Declaration	***public class Window extends Container implements Accessible***
Constructors	**Window()** → Constructs a new, initially invisible window.
Methods	**addNotify()** → Creates the Panel's peer. **addWindowListener()** → Adds the specified window listener **getAccessibleContext()** → Gets the AccessibleContext associated with this Window. **dispose()** → Releases all of the resources used by this Window, its subcomponents, and all of its owned children. **isActive()** → Returns whether this Window is active.

5.4 Event Handling

Every time the user types a character or pushes a mouse button, an event occurs. Any object can be notified of the event.



Define Event:

- Changing the state of an object is known as an event. For example, click on button, dragging mouse etc.
- The ***java.awt.event*** package provides many event classes and Listener interfaces for event handling.

Define Event Handling:

- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs.

Components of Event Handling:

Event handling has three main components:

Event : Changing the state of an object is called an event.

Event Source : Event source is an object that generates an event.

Listener : A listener gets notified when an event occurs.

How Events are handled ?

A source generates an event and sends it to one or more listeners registered with the source. Once event is received by the listener, they process the event and return. Events are supported by a number of Java packages, like $java.util, java.awt$ and $java.awt.event$.

Important Event Classes and Interface

S. No.	Event Classes	Description	Listener Interface
1	**ActionEvent**	User clicks a button, or chooses a menu item or list-item is double clicked	**ActionListener**
2	**MouseEvent**	Generated when mouse is dragged, moved, clicked, pressed or released	**MouseListener**
3	**KeyEvent**	Generated when input is received from keyboard	**KeyListener**
4	**ItemEvent**	Generated when check-box or list item is clicked	**ItemListener**
5	**TextEvent**	Generated when textarea or textfield is changed	**TextListener**
6	**WindowEvent**	Generated when window is activated, deactivated, opened or closed.	**WindowListener**
7	**ComponentEvent**	Generated when component is moved or resized.	**ComponentEventListener**
8	**ContainerEvent**	Generated when component is added or removed from container	**ContainerListener**
9	**AdjustmentEvent**	Generated when scroll bar is manipulated	**AdjustmentListener**

Steps to handle events:

Step 1:

- Class should implements a listener interface or extends a class that implements a listener interface.
- Example: $\boldsymbol{public\ class\ MyClass\ implements\ ActionListener}\{\}$

Step 2:

- Register the listener to one or more components.
- Example: $\boldsymbol{component.addActionListener(instanceOfMyClass)}$;

Step 3:

- Implement the methods in the listener interface.

- Example:

  public void actionPerformed(ActionEvent e)

  { ...//code that reacts to the action... }

Registration Methods

Component	Event Listeners
Button, 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){}

Example 1: Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
class Test extends Frame implements ActionListener
{
   TextField tf;
   Test()
   {
      tf = new TextField();
      tf.setBounds(60,50,170,20);
      Button b=new Button("click me");
      b.setBounds(100,120,80,30);

      b.addActionListener(this);

      add(b);
      add(tf);
      setSize(300,300);
      setLayout(null);
      setVisible(true);
   }

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

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



→ Create components

→ Add register listener and pass current instance

→ Add components and set size, layout and visibility

→ Implementing the Action

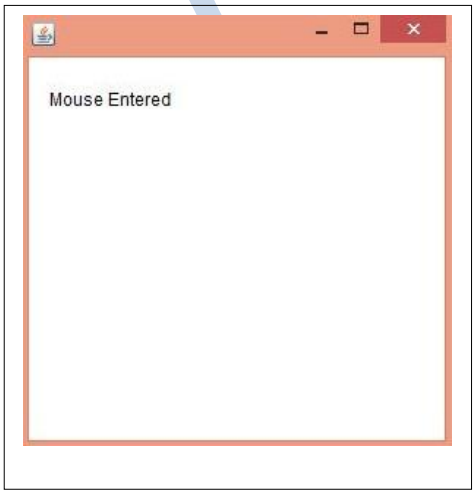
Example 2: Java event handling by implementing MouseListener

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener
{
   Label l;
   MouseListenerExample()   → Default Constructor
   {
      addMouseListener(this);   → Add a new Mouse Listener

      l = new Label();
      l.setBounds(20,50,100,20);
      add(l);
      setSize(300,300);
      setLayout(null);
      setVisible(true);
   }
   public void mouseClicked(MouseEvent e)
{
      l.setText("Mouse Clicked");                → Mouse clicked event
   }
   public void mouseEntered(MouseEvent e)
{
      l.setText("Mouse Entered");                → Mouse entered event
   }
   public void mouseExited(MouseEvent e)
{
      l.setText("Mouse Exited");                 → Mouse exited event
   }
   public void mousePressed(MouseEvent e)
{
      l.setText("Mouse Pressed");                → Mouse pressed event
   }
   public void mouseReleased(MouseEvent e)
{
      l.setText("Mouse Released");               → Mouse released event
   }

public static void main(String[] args)
{
   new MouseListenerExample();   → Invoking the default constructor
}
```

Output:



Example 3: Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
public class ActionListenerExample
{
public static void main(String[] args)
{
    Frame f = new Frame("Button Example");
    TextField tf = new TextField();
    tf.setBounds(50,50, 150,20);
    Button b=new Button("Click Here");
    b.setBounds(50,100,60,30);
    b.addActionListener(new ActionListener()
   {
    public void actionPerformed(ActionEvent e)
    {
        tf.setText("Welcome to Javatpoint.");
      }
    });
    f.add(b);
    f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
   }
}
```



5.5 Java Applets

5.5.1 Introduction

Applications are stand-alone programs that are executed with Java interpreter, whereas Applet is a small program that can be placed on a web page.

Define Applet:

Applet is a special type of Java programs that can be embedded in HTML documents to generate the dynamic content. It runs inside the browser and works at client side.

Advantages:

- 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.

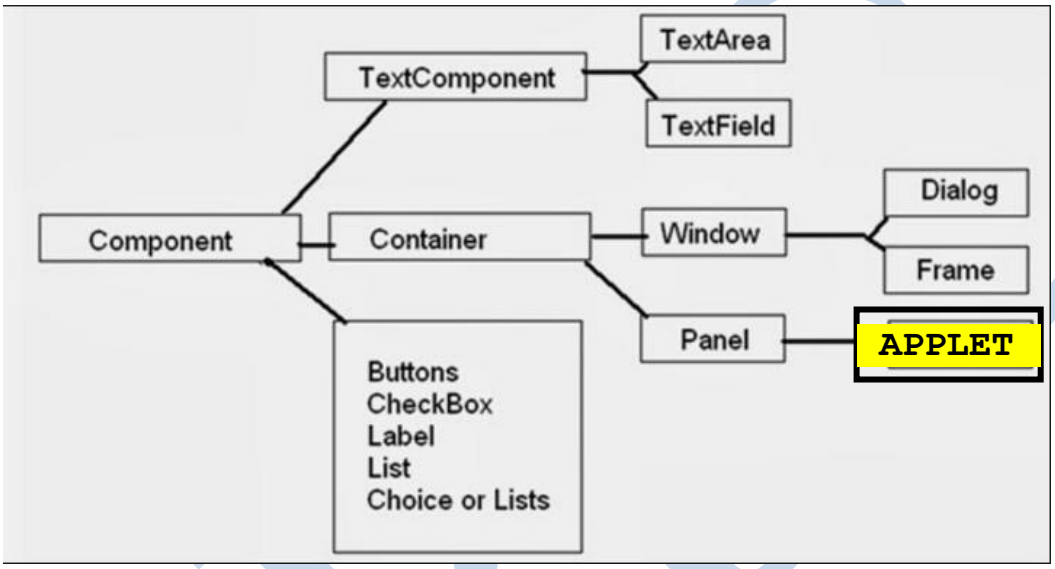
Disadvantages:

- Plugin is required at client browser to execute applet.

State the differences between Applications and Applets

S. No	Java Applications	Java Applets
1	It is a stand-alone program	It is an Embedded program Applet is embedded in HTML document
2	Application runs independently	Applet runs in a web browser
3	Application has a main() method	No main() method
4	No HTML file	Requires an HTML file
5	Run using JDK Java Intrepreter	Run using JDK appletviewer

Hierarchy of Applet



Hierarchy of an applet

5.5.2 Life cycle of an Applet

Java applet inherits features from the class Applet. Thus, whenever an applet is created, it undergoes a series of changes from initialization to destruction. Various stages of an applet life cycle are depicted in the figure below:

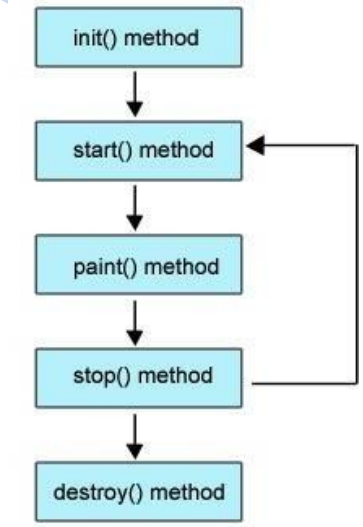


Figure: Life cycle of Applet

1. Initial State

When a new applet is born or created, it is activated by calling **init() method**. At this stage; initial values are set, images are loaded and the colors of the images are set. An applet is initialized *only once* in its lifetime.

Syntax:

```
public void init( )
{
        //Action to be performed
}
```

2. Running State

An applet enters to the running state when the system calls the start() method. This occurs as soon as the applet is initialized. An applet may also start when it is in idle state. An applet is initialized *only once* in its lifetime.

Syntax:

```
public void start( )
{
        //Action to be performed
}
```

3. Graphics state

An applet enters to the Graphics state when the system calls the paint() method. This method helps in drawing, writing and giving background colours of the applet. It takes an argument of the graphics class. To use graphics class, import the package ***java.awt.Graphics*.**

Syntax:

```
public void paint(Graphics g)
 {
        //Action to be performed
}
```

4. Idle state

An applet comes in idle state when it stopped its execution either implicitly or explicitly.

- Implicitly stopped → when we leave the currently running page.
- Explicitly stopped → when we call stop() method to stop its execution.

Syntax:

```
public void stop()
 {
        //Action to be performed
}
```

5. Dead State

An applet is in dead state when it's removed from the memory. This can be done by using destroy() method.

Syntax:

```
public void destroy()
 {
        //Action to be performed
}
```

5.5.3 Steps to create an Applet program

- Step 1: Create a Java Source File
- Step 2: Compile the Source File
- Step 3: Create an HTML File that Includes the Applet
- Step 4: Run the Applet

How to run an Applet?

There are two ways to run an applet.

- By HTML file.
- By appletViewer tool

Step 1: Create a Java Source File

Create a file named HelloWorld.java with the Java code shown here:

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet
{
   public void paint(Graphics g)
   {
      g.drawString("Hello world!", 150, 25);
   }
}
```

Create a class which extends Applet

Invoke the paint method of the Applet

Step 2: Compile the Source File

Compile the source file using the Java compiler. If the compilation succeeds, the compiler creates a file named HelloWorld.class. This class file contains Java bytecodes.

Step 3: Create an HTML File that Includes the Applet

Using a text editor, create a file named Hello.html in the same directory that contains HelloWorld.class. This HTML file should contain the following:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<APPLET CODE = . . . >
</APPLET>
</BODY>
</HTML>
```

For the given program

```
<HTML>
  <BODY>
     <APPLET CODE="HelloWorld.class" WIDTH=150 HEIGHT=25>
     </APPLET>
  </BODY>
</HTML>
```

Step 4: Run the Applet

To run the applet, run the Applet in your browser window. Type the path for your HTML file, c:\java\HelloWorld.html

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

c:\>appletviewer HelloWorld.java

```
//HelloWorld.java
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet
{
   public void paint(Graphics g)
   {
      g.drawString("Hello world!", 150, 25);
   }
}
```

Create the applet tag in Comments

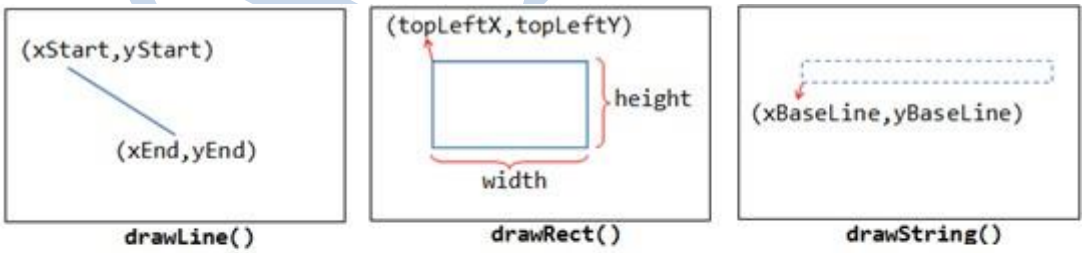
```
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

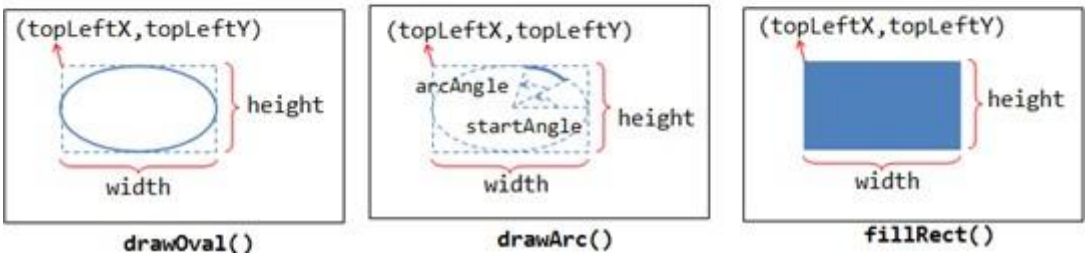
5.5.4 Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming. Most drawing methods include:

- drawString – For drawing text
- drawImage – For drawing images
- drawLine, drawArc, drawRect, drawOval, drawPolygon – For drawing geometric shapes

Methods of Graphics Class	
drawString(String str, int x, int y)	Used to draw the specified string
drawRect(int x, int y, int width, int height)	Used to draw a rectangle with the specified width and height.
fillRect(int x, int y, int width, int height)	Used to fill rectangle with the default color and specified width, height.
drawOval(int x, int y, int width, int height)	Used to draw oval with the specified width and height.
fillOval(int x, int y, int width, int height)	Used to fill oval with the default color and specified width,height
drawLine(int x1, int y1, int x2, int y2):	Used to draw line between the points(x1, y1) and (x2, y2).
boolean drawImage(Image img, int x, int y, ImageObserver observer)	Used draw the specified image.
drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)	Used draw a circular or elliptical arc.
fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)	Used to fill a circular or elliptical arc.
setColor(Color c)	Used to set the specified color.
setFont(Font font)	Used to set the specified font.



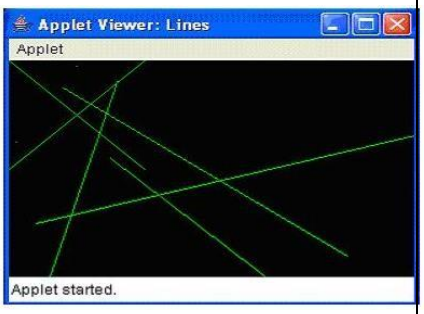


Java Applet to draw lines

```
import java.awt.*;
import java.applet.*;
/*
<applet code="Test. class" width=300 height=200>
</applet>
*/

public class Test extends Applet
{
        public void init()
        {
                setBackground (Color.black);
                setForeground(Color.green);
        }

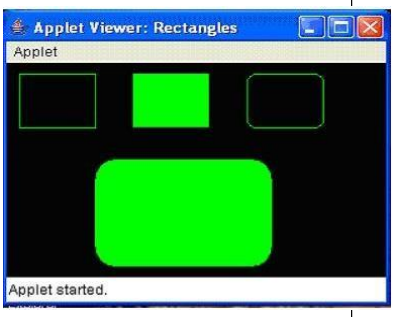
        public void paint(Graphics g)
        {
                g.drawLine(0, 0, 100, 100);
                g.drawLine(0, 100, 100, 0);
                g.drawLine(40, 25, 250, 180);
                g.drawLine(75, 90, 400, 400);
                g.drawLine(20, 150, 400, 40);    → Draw a line
                g.drawLine(5, 290, 80, 19);
                g.drawLine (5, 75, 5, 75);   → plot a point
                g.drawLine (50, 5, 50, 5);
        }
}
```



Java Applet to draw & fill Rectangles

```
import java.awt.*;
import java.applet.*;
/*
<applet code="Rectangles.class" width=300 height=200>
</applet>
*/
public class Rectangles extends Applet
{
        public void init()
        {
                setBackground(Color.black);
                setForeground(Color.green);
        }

        public void paint(Graphics g)
        {
                g.drawRect(10, 10, 60, 50);
                g.fillRect(100, 10, 60, 50);
                g.drawRoundRect(190, 10, 60, 50, 15, 15);
                g.fillRoundRect(70, 90, 140, 100, 30, 40);
        }
}
```



Java Applet to display an image in Applet

```
import java.awt.*;
import java.applet.*;
/*
<applet code=" DisplayImage.class" width=300 height=200>
</applet>
*/

public class DisplayImage extends Applet
{

  Image picture;

  public void init()
  {
    picture = getImage(getDocumentBase(),"sonoo.jpg");
  }

  public void paint(Graphics g)
 {
    g.drawImage(picture, 30,30, this);
  }

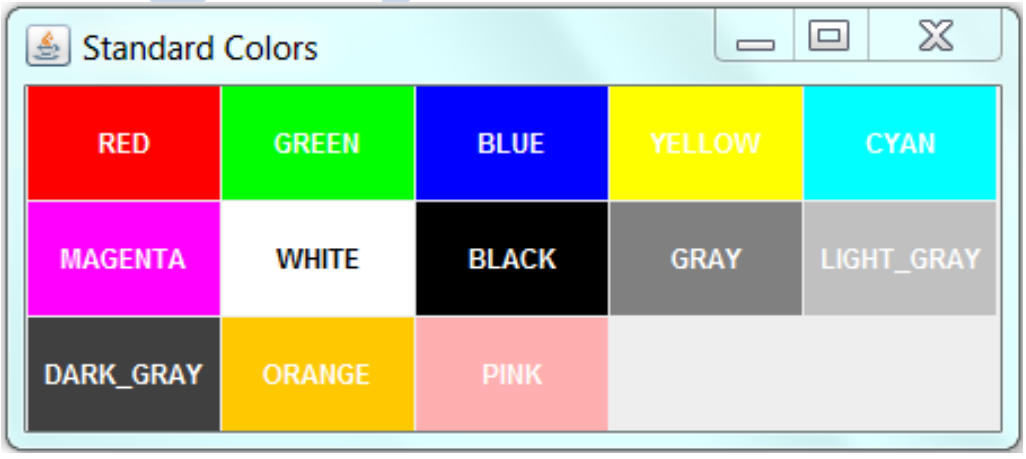
  }
```

5.5.5 Colors and Fonts

java.awt.Color

The class java.awt.Color provides 13 standard colors as named-constants. They are: Color.RED, GREEN, BLUE, MAGENTA, CYAN, YELLOW, BLACK, WHITE, GRAY, DARK_GRAY, LIGHT_GRAY, ORANGE, and PINK

we can use the toString() to print the RGB values of these color (e.g., System.out.println(Color.RED)):



RED	: java.awt.Color[r=255, g=0, b=0]
GREEN	: java.awt.Color[r=0, g=255, b=0]
BLUE	: java.awt.Color[r=0, g=0, b=255]
YELLOW	: java.awt.Color[r=255, g=255, b=0]
MAGENTA	: java.awt.Color[r=255, g=0, b=255]
CYAN	: java.awt.Color[r=0, g=255, b=255]
WHITE	: java.awt.Color[r=255, g=255, b=255]
BLACK	: java.awt.Color[r=0, g=0, b=0]
GRAY	: java.awt.Color[r=128, g=128, b=128]
LIGHT_GRAY	: java.awt.Color[r=192, g=192, b=192]
DARK_GRAY	: java.awt.Color[r=64, g=64, b=64]
PINK	: java.awt.Color[r=255, g=175, b=175]
ORANGE	: java.awt.Color[r=255, g=200, b=0]

Example:

g.setColor(Color.RED); → set the color as Red

g.drawLine(10, 20, 30, 40); → Now, line will be printed using red color

Color myColor = new Color(125, 155, 15); → create mycolor

g.setColor(myColor);

g.drawRect(10, 10, 40, 50); → draw the rectangle with myColor

java.awt.Font

The class java.awt.Font represents a specific font face, which can be used for representing texts.

Syntax:

public Font(String name, int style, int size);

Family name → "Dialog", "DialogInput", "Monospaced", "Serif", or "SansSerif

(or)

String name → Font.SERIF, Font.SANS_SERIF, Font.MONOSPACED

Style → Font.PLAIN, Font.BOLD, Font.ITALIC or Font.BOLD|Font.ITALIC

Size → font size (in pt) (1 inch has 72 pt).

Example:

```
Font myFont = new Font(Font.MONOSPACED, Font.PLAIN, 12);
Button b = new Button("RESET");
b.setFont(myFont);
```

5.6 Java Adapter Classes

Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code. The adapter classes are found in *java. awt. event*, *java. awt. dnd* and *javax. swing. event* packages. The adapter classes with their corresponding listener interfaces are given below.

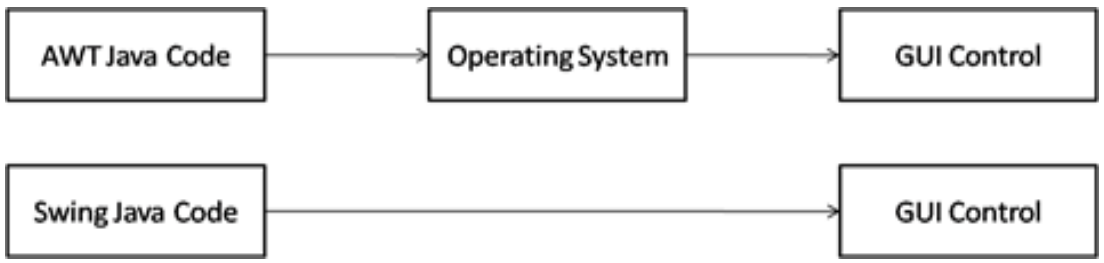
Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

5.7 Java Swing

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. Unlike AWT, Java Swing provides platform-independent and lightweight components.

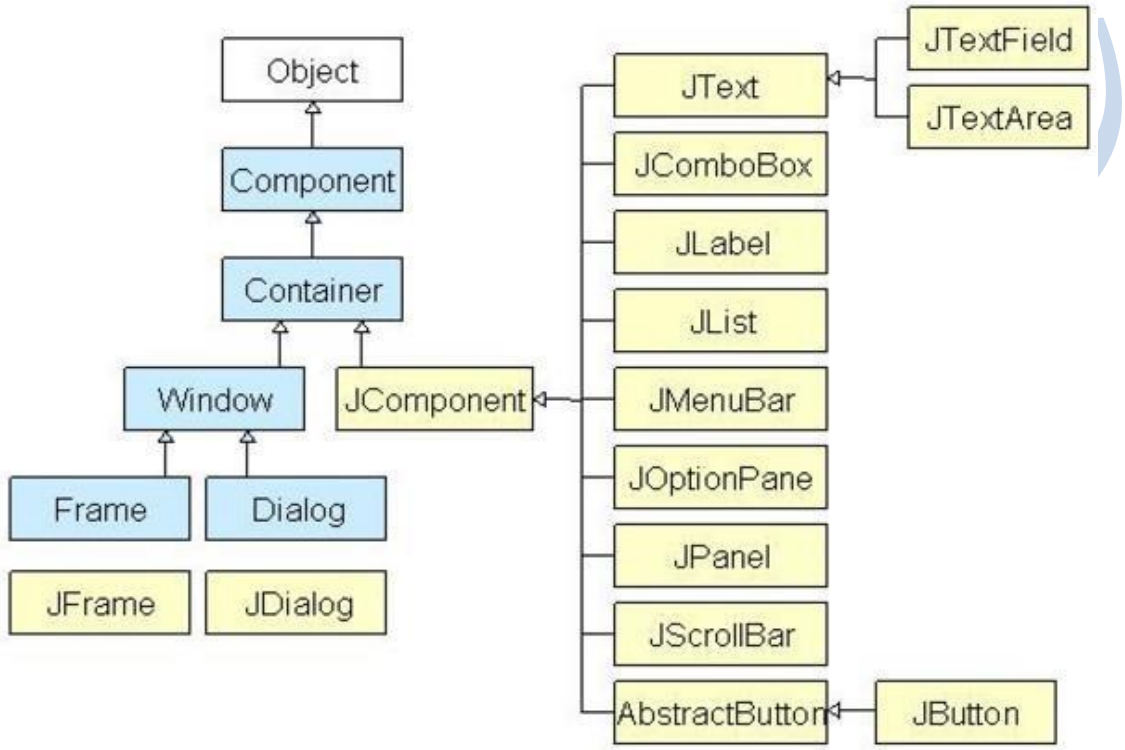
The ***javax. swing*** package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

State the differences between Java AWT and Java Swing



S. 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 fewer 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)	Swing follows MVC

5.7.1 Hierarchy of Java Swing classes



Commonly used Methods of Component class:

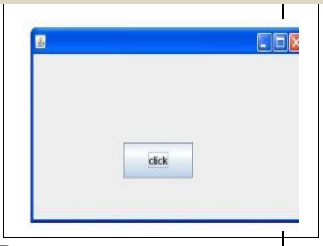
Method	Description
add(Component c)	Add a component on another component.
setSize(int width,int height)	Set the size of the component.
setLayout(LayoutManager m)	Set the layout manager for the component.
setVisible(boolean b)	Sets the visibility of the component. Default value is false.

Creating the frames:

There are two ways to create a frame:

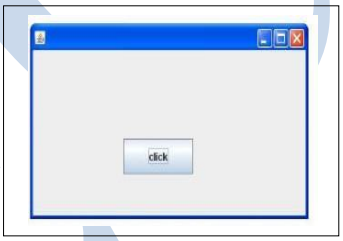
- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

Example 1: Simple Java Swing by creating object of the Frame class



```
import javax.swing.*;
public class FirstSwingExample
{
public static void main(String[] args)
{
   JFrame f = new JFrame();  → creating an instance of JFrame
   JButton b =new JButton("click"); → creating an 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); → set the size of the frame with 400 width and 500 height
    f.setLayout(null); → No layout manager
    f.setVisible(true);  → Make the frame visible
    }
}
```

Example 2: Simple Java Swing by extending Frame class



```
import javax.swing.*;
public class Test extends JFrame    → Inheriting JFrame
{
   JFrame f;

   Test()
   {
      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 Test();
 }
}
```

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

5.7.2 SWING UI Elements

Following is the list of commonly used controls while designing GUI using SWING.

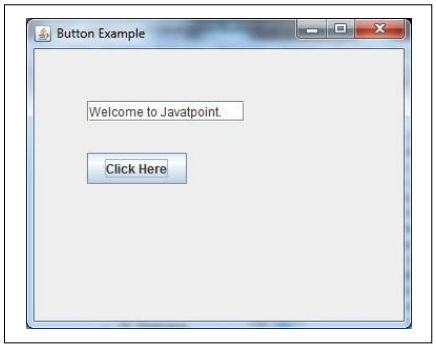
S.No.	Class & Description
1	**JLabel** → Used to describe a text field.
2	**JButton** → Create a labelled button.
3	**JColorChooser** → Allows a user to select a color.

S.No.	Class & Description
4	**JCheck Box** → Graphical component that can be in on (true) or off (false) state.
5	**JRadioButton** → graphical component that can be in on or off state in a group.
6	**JList** → Gives the scrolling list of text items.
7	**JComboBox** → Show the menu of choices.
8	**JTextField** → Gets the single line of text.
9	**JPasswordField** → Text component specialized for password entry.
10	**JTextArea** → Allows multiple lines of text.
11	**ImageIcon** → Paints Icons from Images
12	**JScrollbar** → Represents a scroll bar which enables the user to select from range of values.
14	**JFileChooser** → Represents a dialog window from which the user can select a file.
15	**JProgressBar** → Displays the task's percentage of completion.
16	**JSlider** → Allows the user to graphically select a value.
17	**JSpinner** → Allows the user select a number or an object value from an ordered sequence.

Simple Java Swing Example for JButton with ActionListener

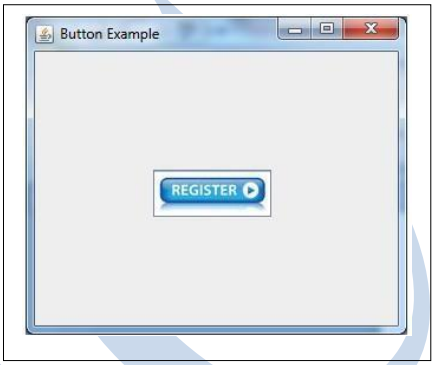
```
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample
{
public static void main(String[] args)
{
    JFrame f = new JFrame("Button Example");
    JTextField tf = new JTextField();
    tf.setBounds(50,50, 150,20);
    JButton b=new JButton("Click Here");
    b.setBounds(50,100,95,30);

    b.addActionListener(new ActionListener()
    {
    public void actionPerformed(ActionEvent e)
    {
       tf.setText("Welcome to Javatpoint.");
     }
    });
    f.add(b);
    f.add(tf);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
  }
}
```



Simple Java Swing Example for JButton by displaying image on the button

```
import javax.swing.*;
public class ButtonExample
{
   ButtonExample()
   {
      JFrame f = new JFrame("Button Example");
      JButton b = new JButton(new ImageIcon("D:\\icon.png"));
      b.setBounds(100,100,100, 40);
      f.add(b);
      f.setSize(300,400);
      f.setLayout(null);
      f.setVisible(true);
   }
public static void main(String[] args)
{
   new ButtonExample();
 }
}
```



Java JSlider

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

Simple Java Swing Example for JSlider

```
import javax.swing.*;
public class SliderExample1 extends JFrame
{
   public SliderExample1()
   {
      JSlider slider = new JSlider(JSlider.HORIZONTAL, 0, 50, 25);
      JPanel panel = new JPanel();
      panel.add(slider);
      add(panel);
   }

public static void main(String s[])
{
   SliderExample1 f = new SliderExample1();
   f.setVisible(true);
  }
}
```



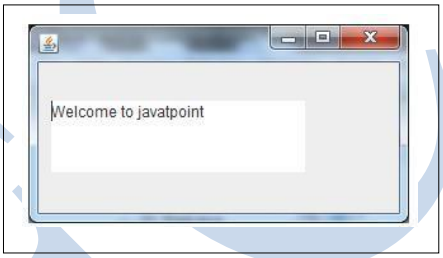
Note: JSlider(int orientation, int min, int max, int value)

Java JTextArea

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

Simple Java Swing Example for JTextArea

```
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();
 }
```



Java JTextField

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

Simple Java Swing Example for JTextField

```
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);
   }
}
```



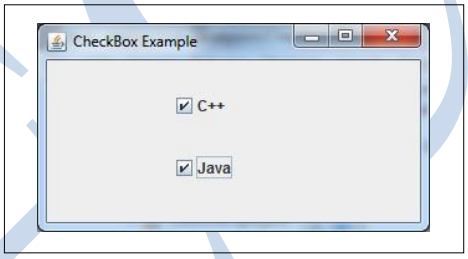
Java JCheckBox

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 ".

Simple Java Swing Example for JCheckBox

```
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();
   }
}
```



Java JRadioButton

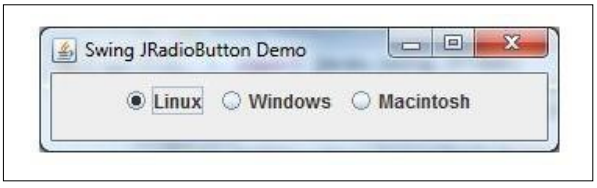
The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options.

Simple Java Swing Example for JRadioButton

```
import javax.swing.*;
public class RadioButtonExample
{
   RadioButtonExample ()
   {
       JFrame f = new JFrame("Swing JRadioButton Demo");
       JRadioButton b1 = new JRadioButton("Linux");
       JRadioButton b2 = new JRadioButton("Windows");
       JRadioButton b3 = new JRadioButton("Macintosh");

       ButtonGroup g = new ButtonGroup();
       g.add(b1);       g.add(b2);       g.add(b3);

       f.add(g);
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
    }
   public static void main(String args[])
   {
   new RadioButtonExample();
   }
}
```



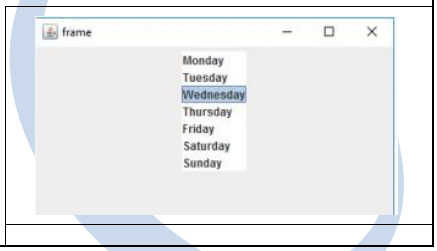
Java JList

Jlist helps the user can choose either one item or multiple items.

Simple Java Swing Example for JList

```
import javax.swing.*;
class Test extends JFrame
{
   public static void main(String[] args)
   {
      JFrame f = new JFrame("frame");   →        create a new frame
      JPanel p =new JPanel(); → create a panel
      JLabel l= new JLabel("select the day of the week"); → create a new label
      //String array to store weekdays
      String week[]= { "Monday","Tuesday","Wednesday",
                  "Thursday","Friday","Saturday","Sunday"};

      Jlist b = new JList(week);   → create list
      b.setSelectedIndex(2); → set a selected index
      p.add(b);   → add list to panel
      f.add(p);
      f.setSize(400,400);  → set the size of frame
   }
}
```



Java 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.

Simple Java Swing Example for JDialog

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DialogExample
{
   DialogExample()
  {
      JFrame f= new JFrame();
      JDialog 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();
   } }
```



5.8 Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner.

- java.awt.BorderLayout
- java.awt.FlowLayout
- java.awt.GridLayout
- java.awt.CardLayout
- java.awt.GridBagLayout
- javax.swing.BoxLayout
- javax.swing.GroupLayout
- javax.swing.ScrollPaneLayout
- 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.

Simple Example for BorderLayout

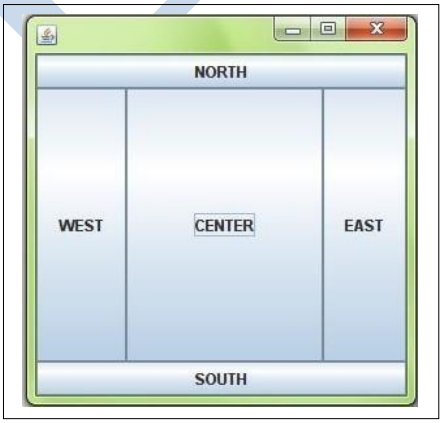
```
import java.awt.*;
import javax.swing.*;

public class BorderExample
{
   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 BorderExample();
   }
}
```



→ Adding border layout to the frame

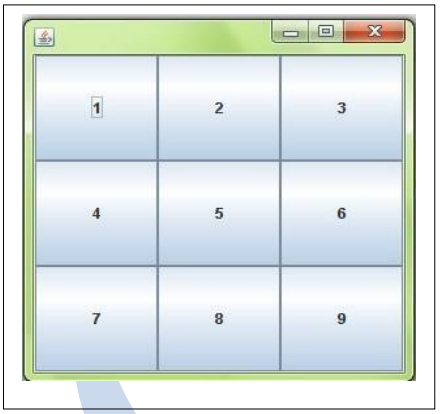
Java GridLayout

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

Simple Example for GridLayout

```
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)); → setting grid layout of 3 rows 3 columns
      f.setSize(300,300);
      f.setVisible(true);
    }

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



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.

Simple Example for FlowLayout

```
 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));
    //setting flow layout of right alignment
   f.setSize(300,300);
   f.setVisible(true);
}
public static void main(String[] args)
{     new MyFlowLayout();
 } }
```



Java BoxLayout

The BoxLayout is used to arrange the components either vertically or horizontally.

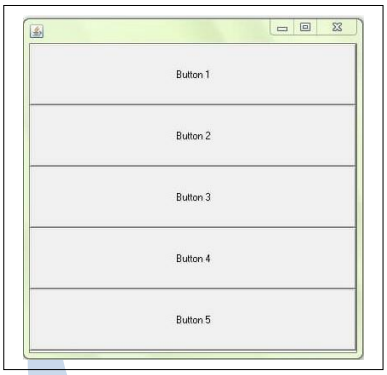
Simple Example for BoxLayout

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

public class BoxLayoutExample1 extends Frame
{
 BoxLayoutExample1 ()
 {
   Button b1 = new Button("Button 1");
   Button b2 = new Button("Button 2");
   Button b3 = new Button("Button 3");
   Button b4 = new Button("Button 4");
   Button b5 = new Button("Button 5");
   add(b1);    add(b2);    add(b3);    add(b4);
   add(b5);
}

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

public static void main(String args[])
{
new BoxLayoutExample1();
} }
```



Java 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.



Java 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.

