# Java AWT (Abstract Window Toolkit)

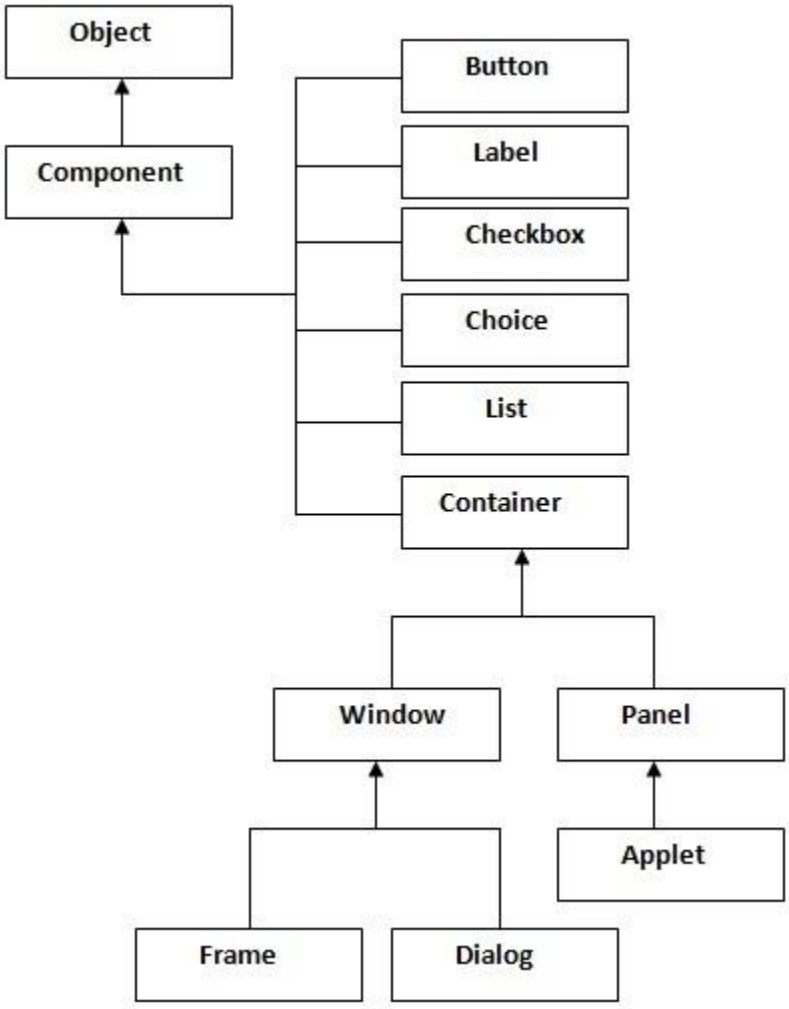
**Java AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

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.

# Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.




## 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.

## Panel

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

## Frame

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

# Useful Methods of Component class

| Method | Description |
|---|---|
| public void add(Component c) | inserts a component on this component. |
| public void setSize(int width,int height) | sets the size (width and height) of the component. |
| public void setLayout(LayoutManager m) | defines the layout manager for the component. |
| public void setVisible(boolean status) | changes the visibility of the component, by default false. |

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

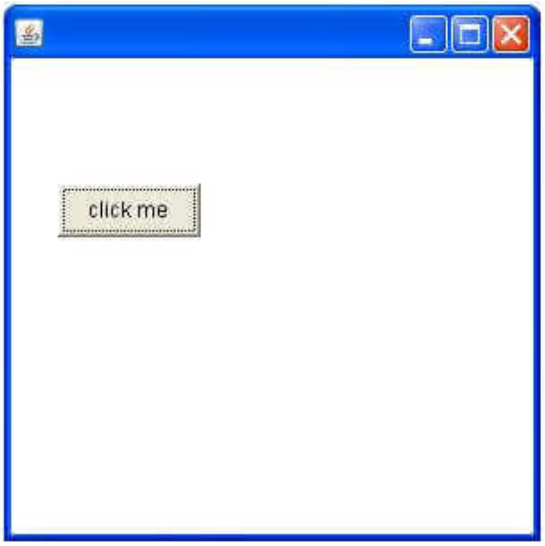
# AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First extends Frame{
First(){
Button b=new Button("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();
}}
```

The setBounds(int xaxis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.

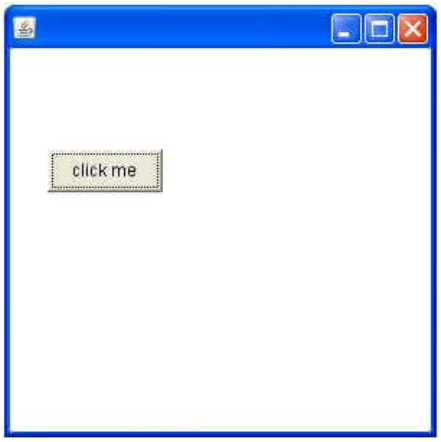



# AWT Example by Association

Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First2{
First2(){
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[]){
First2 f=new First2();
```

}}



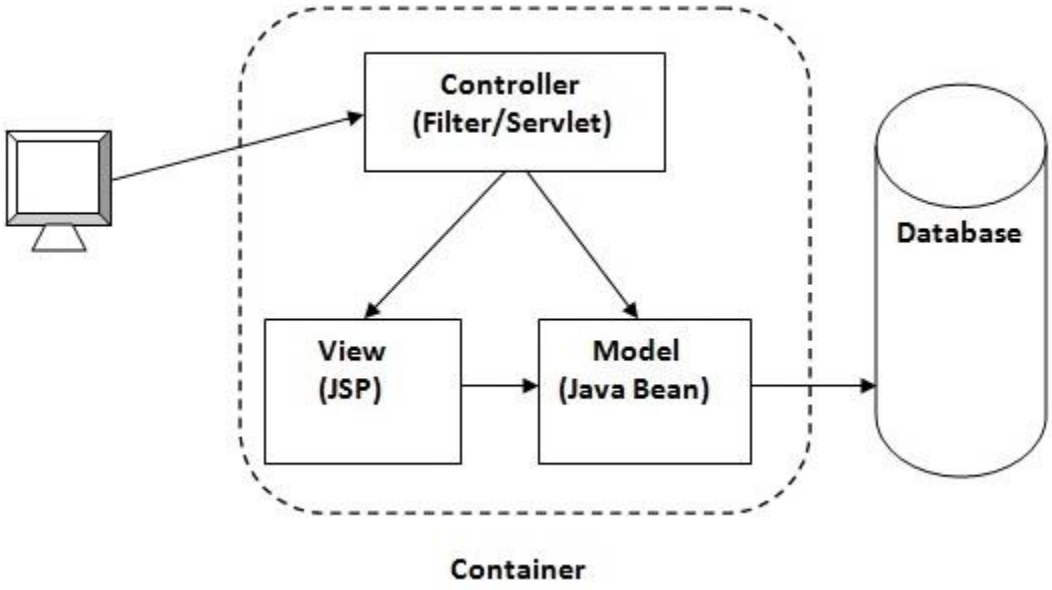

## (MVC) Architecture

Model 2 is based on the MVC (Model View Controller) design pattern. 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.

**View** The view module is responsible to display data i.e. it represents the presentation.

**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 Model 2 (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 Model 2 (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.

# 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.

# Difference between AWT and Swing

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

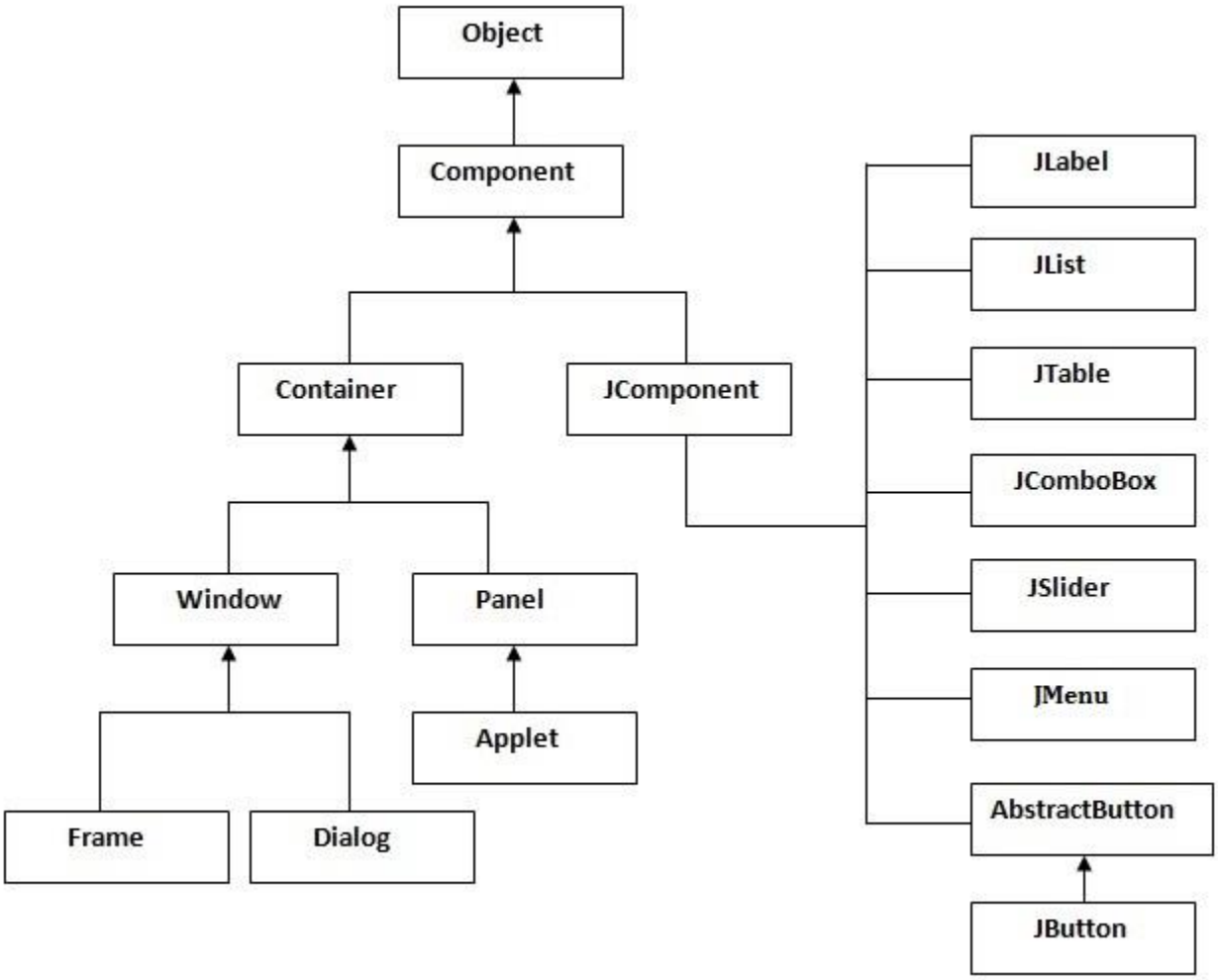
| No. | Java AWT | Java Swing |
|---|---|---|
| 1) | AWT components are **platform-dependent**. | Java swing components are **platform-independent**. |
| 2) | AWT components are **heavyweight**. | Swing components are **lightweight**. |
| 3) | AWT **doesn't support pluggable look and feel**. | Swing **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**. |

# What is JFC

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

# Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.




# Commonly used Methods of Component class

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. |

# Java Swing Examples

There are two ways to create a frame:

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

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

# Java 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.

## JButton class declaration

Let's see the declaration for javax.swing.JButton class.

1. **public class** JButton **extends** AbstractButton **implements** Accessible

## Commonly used Constructors:

| Constructor | Description |
|---|---|
| JButton() | It creates a button with no text and icon. |
| JButton(String s) | It creates a button with the specified text. |
| JButton(Icon i) | It creates a button with the specified icon object. |

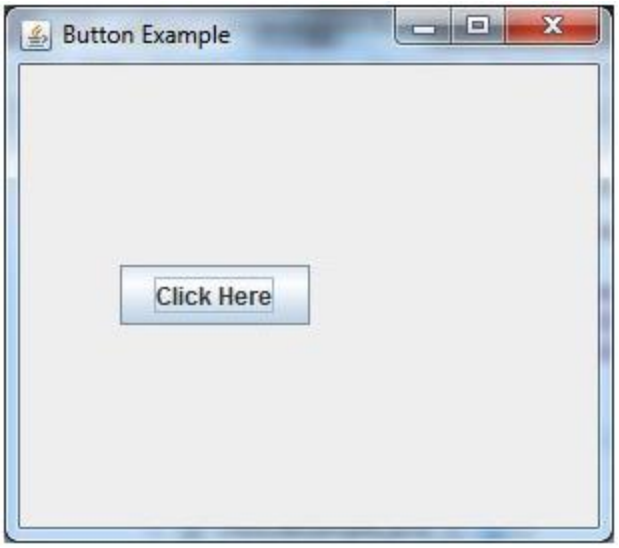
## Commonly used Methods of AbstractButton class:

| 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. |

# Java JButton 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 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.

# JLabel class declaration

Let's see the declaration for javax.swing.JLabel class.

1. **public class** JLabel **extends** JComponent **implements** SwingConstants, Accessible

## Commonly used Constructors:

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

## 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 alignment) | It sets the alignment of the label's contents along the X axis. |
| Icon getIcon() | It returns the graphic image that the label displays. |
| int getHorizontalAlignment() | It returns the alignment of the label's contents along the X axis. |

# Java JLabel 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:


# Java JTextField

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

## JTextField class declaration

Let's see the declaration for javax.swing.JTextField class.

1. **public class** JTextField **extends** JTextComponent **implements** SwingConstants

### Commonly used Constructors:

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

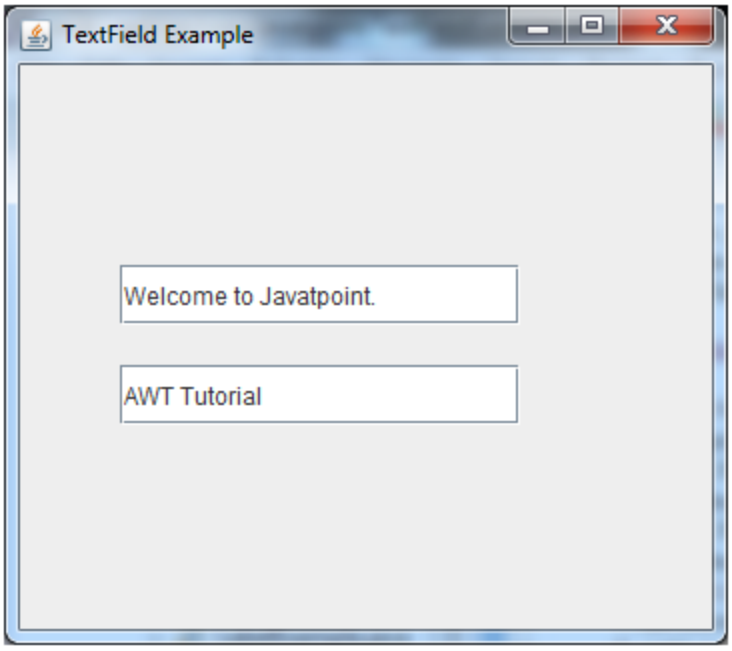
### Commonly used Methods:

| 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. |

# Java JTextField 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:




# Java JToggleButton

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

## Nested Classes

| Modifier and Type | Class | Description |
|---|---|---|
| protected class | JToggleButton.AccessibleJToggleButton | This class implements accessibility support for the JToggleButton class. |
| static class | JToggleButton.ToggleButtonModel | The ToggleButton model |

## Constructors

| Constructor | Description |
|---|---|
| JToggleButton() | It creates an initially unselected toggle button without setting the text or image. |

| | |
|---|---|
| JToggleButton(Action a) | It creates a toggle button where properties are taken from the Action supplied. |
| JToggleButton(Icon icon) | It creates an initially unselected toggle button with the specified image but no text. |
| JToggleButton(Icon icon, boolean selected) | It creates a toggle button with the specified image and selection state, but no text. |
| JToggleButton(String text) | It creates an unselected toggle button with the specified text. |
| JToggleButton(String text, boolean selected) | It creates a toggle button with the specified text and selection state. |
| JToggleButton(String text, Icon icon) | It creates a toggle button that has the specified text and image, and that is initially unselected. |
| JToggleButton(String text, Icon icon, boolean selected) | It creates a toggle button with the specified text, image, and selection state. |

## Methods

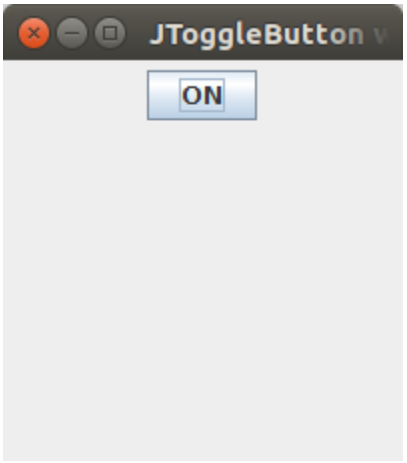
| Modifier and Type | Method | Description |
|---|---|---|
| AccessibleContext | getAccessibleContext() | It gets the AccessibleContext associated with this JToggleButton. |
| String | getUIClassID() | It returns a string that specifies the name of the l&f class that renders this component. |
| protected 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. |

# JToggleButton Example

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

class JToggleButtonExample extends JFrame
{
   public static void main(String[] args)
{
      new JToggleButtonExample();
   }

   JToggleButtonExample()
{
      setTitle("JToggleButton with ItemListener Example");
        JToggleButton button =new JToggleButton("on");
      setLayout(new FlowLayout());
      setSize(200, 200);
      setVisible(true);
        add(button);
   }
 }
```

Output




# 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 ".It inherits JToggleButton class.

# JCheckBox class declaration

Let's see the declaration for javax.swing.JCheckBox class.

1. **public class** JCheckBox **extends** JToggleButton **implements** Accessible

## Commonly used Constructors:

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

## Commonly used Methods:

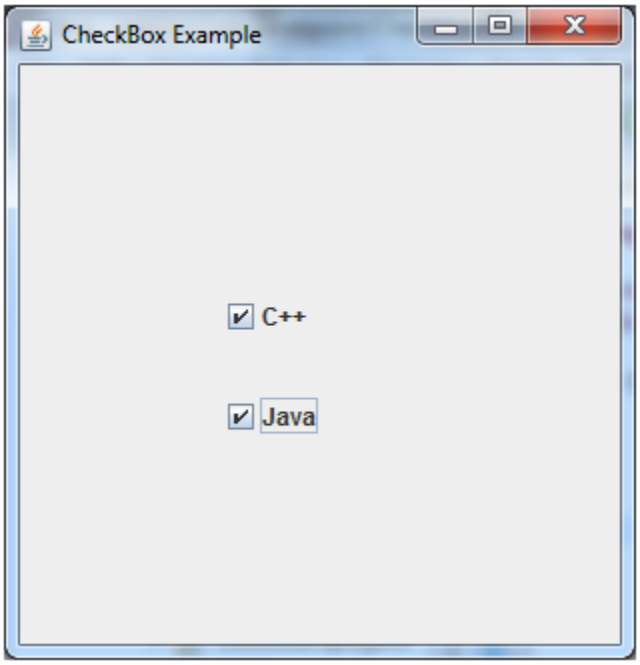
| Methods | Description |
|---|---|
| AccessibleContext getAccessibleContext() | It is used to get the AccessibleContext associated with this JCheckBox. |
| protected String paramString() | It returns a string representation of this JCheckBox. |

# Java JCheckBox 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:




# Java 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.

# JRadioButton class declaration

Let's see the declaration for javax.swing.JRadioButton class.

1. **public class** JRadioButton **extends** JToggleButton **implements** Accessible

## Commonly used Constructors:

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

## Commonly used Methods:

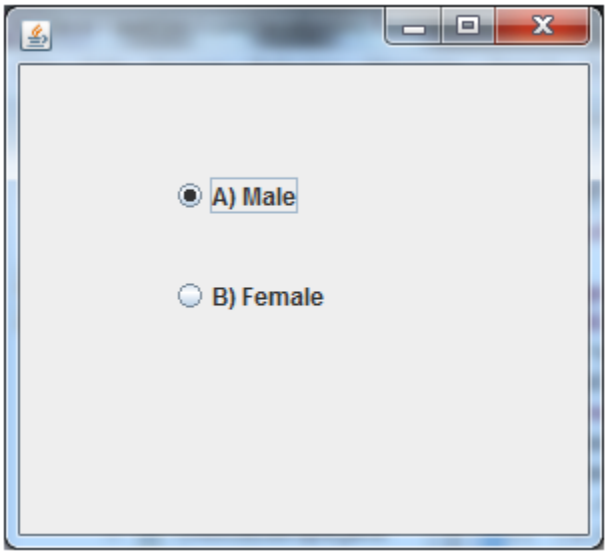
| Methods | Description |
|---|---|
| void setText(String s) | It is used to set specified text on button. |
| String getText() | It is used to return the text of the button. |
| void setEnabled(boolean b) | It is used to enable or disable the button. |
| void setIcon(Icon b) | It is used to set the 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. |

# Java JRadioButton Example

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

```
RadioButtonExample(){
f=new JFrame();
JRadioButton r1=new JRadioButton("A) Male");
JRadioButton r2=new JRadioButton("B) Female");
r1.setBounds(75,50,100,30);
r2.setBounds(75,100,100,30);
ButtonGroup bg=new ButtonGroup();
bg.add(r1);bg.add(r2);
f.add(r1);f.add(r2);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String[] args) {
   new RadioButtonExample();
}
}
```

Output:




# Java 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.

# JTabbedPane class declaration

Let's see the declaration for javax.swing.JTabbedPane class.

1. **public class** JTabbedPane **extends** JComponent **implements** Serializable, Accessible, SwingConstants

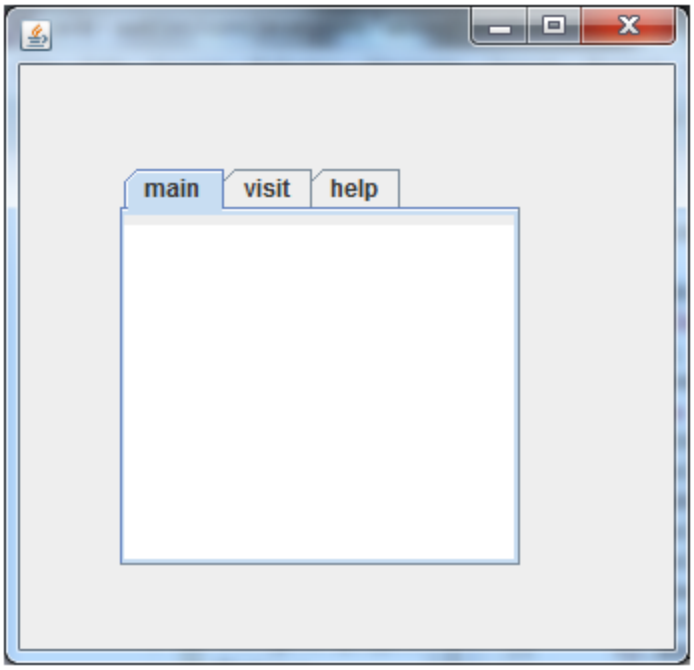
## Commonly used Constructors:

| Constructor | Description |
|---|---|
| JTabbedPane() | Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top. |
| JTabbedPane(int tabPlacement) | Creates an empty TabbedPane with a specified tab placement. |
| JTabbedPane(int tabPlacement, int tabLayoutPolicy) | Creates an empty TabbedPane with a specified tab placement and tab layout policy. |

# Java JTabbedPane Example

```
import javax.swing.*;
public class TabbedPaneExample {
JFrame f;
TabbedPaneExample(){
    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:




# Java 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.

## Constructors

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

## Useful Methods

| Modifier | Method | Description |
|---|---|---|

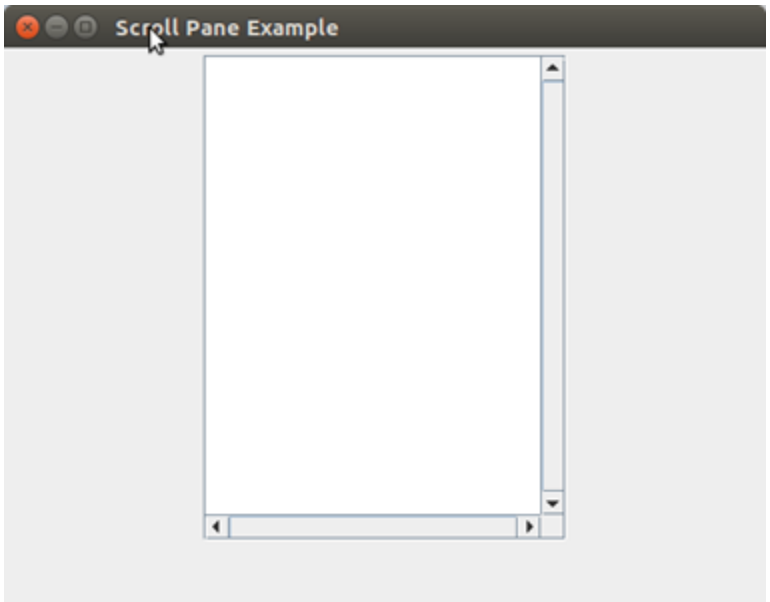
| | | |
|---|---|---|
| void | setColumnHeaderView(Component) | It sets the column header for the scroll pane. |
| void | setRowHeaderView(Component) | It sets the row header for the scroll pane. |
| void | setCorner(String, Component) | It sets or gets the specified corner. The int parameter specifies 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. |

# JScrollPane 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_ALWA
YS);
        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:




# Java 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.

# JList class declaration

Let's see the declaration for javax.swing.JList class.

1. **public class** JList **extends** JComponent **implements** Scrollable, Accessible

## Commonly used Constructors:

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

## Commonly used Methods:

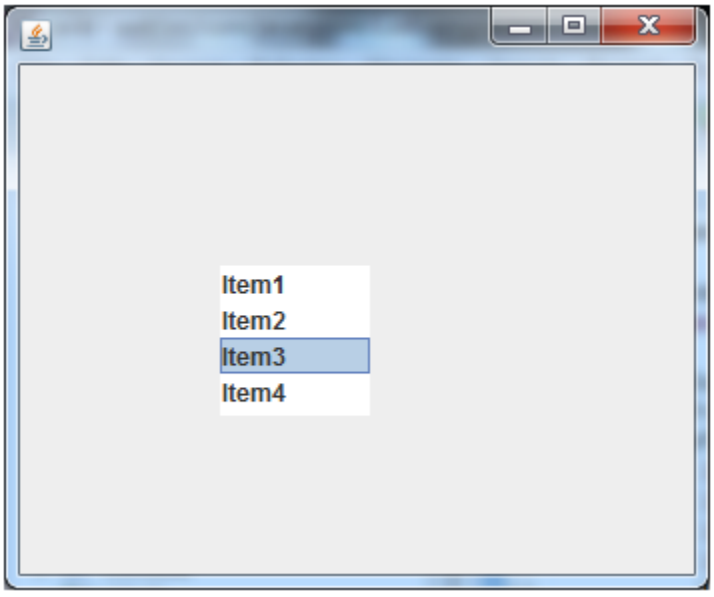
| Methods | Description |
|---|---|
| Void addListSelectionListener(ListSelectionListener listener) | It is used to add a listener to the list, to be notified each time a change to the selection occurs. |
| int getSelectedIndex() | It is used to return the smallest selected cell index. |
| ListModel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

# Java JList 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:




# Java 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.

# JComboBox class declaration

Let's see the declaration for javax.swing.JComboBox class.

1. **public class** JComboBox **extends** JComponent **implements** ItemSelectable, ListDataListener, ActionListener, Accessible

## Commonly used Constructors:

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

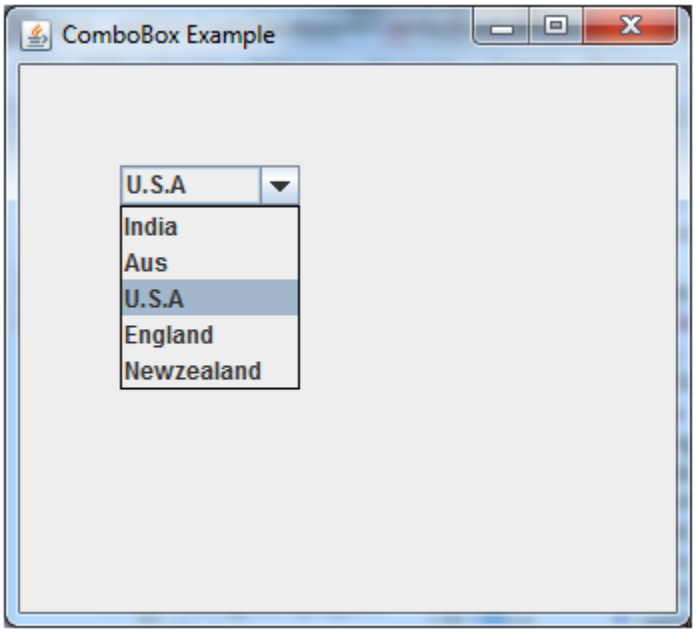
## Commonly used Methods:

| 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. |

# Java JComboBox Example

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

Output:




# Java JMenuBar, JMenu and JMenuItem

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

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

The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

## JMenuBar class declaration

1. **public class** JMenuBar **extends** JComponent **implements** MenuElement, Accessible

## JMenu class declaration

1. **public class** JMenu **extends** JMenuItem **implements** MenuElement, Accessible

## JMenuItem class declaration

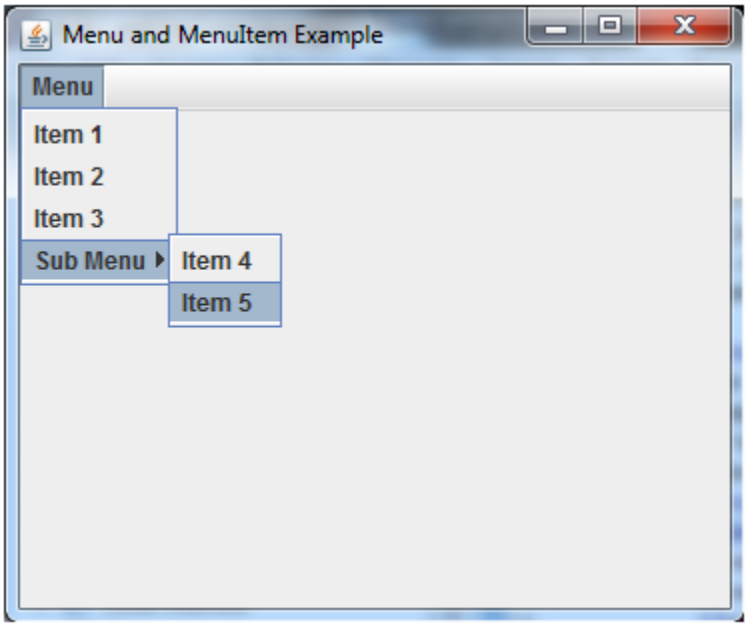
1. **public class** JMenuItem **extends** AbstractButton **implements** Accessible, MenuElement

# Java 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:




# 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.

## JDialog class declaration

Let's see the declaration for javax.swing.JDialog class.

1. **public class** JDialog **extends** Dialog **implements** WindowConstants, Accessible, RootPaneContainer

### Commonly used Constructors:

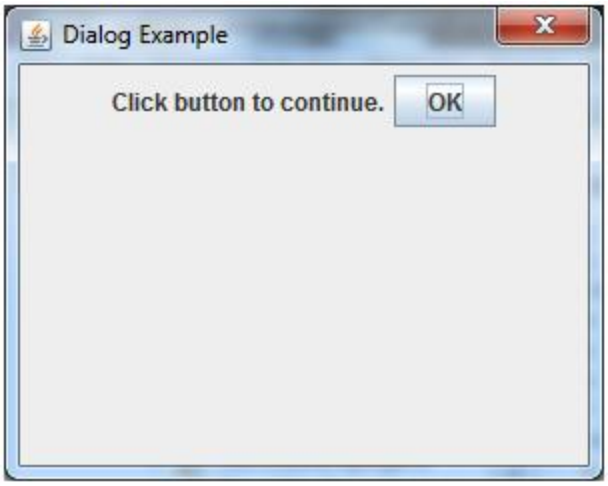
| Constructor | Description |
|---|---|
| JDialog() | It is used to create a modeless dialog without a title and without a specified Frame owner. |
| JDialog(Frame owner) | It is used to create a modeless dialog with specified Frame as its owner and an empty title. |
| JDialog(Frame owner, String title, | It is used to create a dialog with the specified title, owner |

| | |
|---|---|
| boolean modal) | Frame and modality. |

# Java JDialog 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:




## 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. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout

# Java FlowLayout

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

## Fields of FlowLayout class

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

## Constructors of FlowLayout class

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

2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

## Example of FlowLayout class


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

public class MyFlowLayout{
JFrame f;
MyFlowLayout(){
   f=new JFrame();

   JButton b1=new JButton("1");
   JButton b2=new JButton("2");
   JButton b3=new JButton("3");
   JButton b4=new JButton("4");
   JButton b5=new JButton("5");

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

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

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

## Java BorderLayout

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

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

## Constructors of BorderLayout class:

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

## Example of BorderLayout class:


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

public class Border {
JFrame f;
Border(){
   f=new JFrame();

   JButton b1=new JButton("NORTH");;
   JButton b2=new JButton("SOUTH");;
   JButton b3=new JButton("EAST");;
   JButton b4=new JButton("WEST");;
   JButton b5=new JButton("CENTER");;

   f.add(b1,BorderLayout.NORTH);
   f.add(b2,BorderLayout.SOUTH);
   f.add(b3,BorderLayout.EAST);
   f.add(b4,BorderLayout.WEST);
   f.add(b5,BorderLayout.CENTER);

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

```
}
}
```

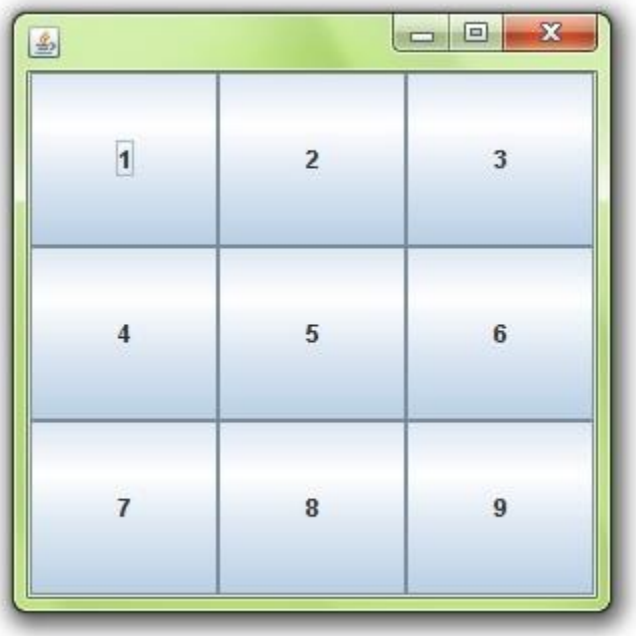
# Java GridLayout

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

## Constructors of GridLayout class

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

## Example of GridLayout class




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

public class MyGridLayout{
JFrame f;
MyGridLayout(){
   f=new JFrame();

   JButton b1=new JButton("1");
```

```
    JButton b2=new JButton("2");
    JButton b3=new JButton("3");
    JButton b4=new JButton("4");
    JButton b5=new JButton("5");
       JButton b6=new JButton("6");
       JButton b7=new JButton("7");
    JButton b8=new JButton("8");
       JButton b9=new JButton("9");

    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
    f.add(b6);f.add(b7);f.add(b8);f.add(b9);

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

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

# 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.

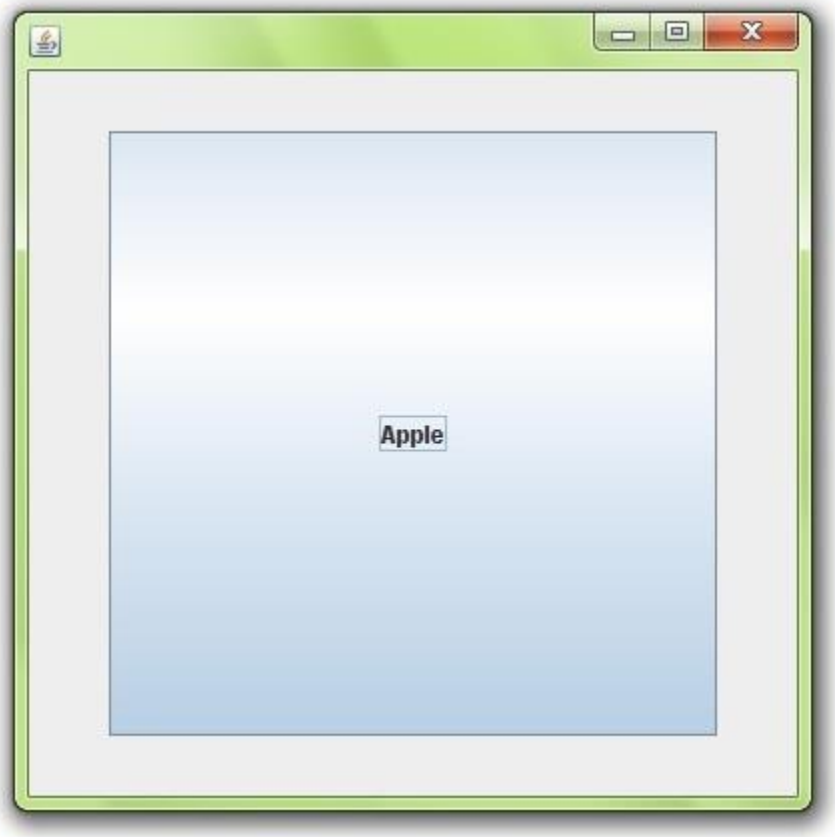
## Constructors of CardLayout class

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

## Commonly used methods of CardLayout class

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

# 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);
    }
}
```

# 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. 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 | AdjustForGravity(GridBagConstraints constraints, Rectangle r) | This method is for backwards compatibility only |
| 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() {
   GridBagLayout  grid = 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 and Listener (Java Event Handling)

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.

## Java Event classes and Listener interfaces

| Event Classes | Listener Interfaces |
|---|---|
| ActionEvent | ActionListener |
| MouseEvent | MouseListener and MouseMotionListener |
| MouseWheelEvent | MouseWheelListener |

| | |
|---|---|
| KeyEvent | KeyListener |
| ItemEvent | ItemListener |
| TextEvent | TextListener |
| AdjustmentEvent | AdjustmentListener |
| WindowEvent | WindowListener |
| ComponentEvent | ComponentListener |
| ContainerEvent | ContainerListener |
| FocusEvent | FocusListener |

# Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

# 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){}

## Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){

//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);

//register listener
b.addActionListener(this);//passing current instance

//add components and set size, layout and visibility
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 AEvent();
}
}
```

**public void setBounds(int xaxis, int yaxis, int width, int height);** have been used in the above example that sets the position of the component it may be button, textfield etc.


## 2) Java event handling by outer class

```
import java.awt.*;
import java.awt.event.*;
class AEvent2 extends Frame{
TextField tf;
AEvent2(){
//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
//register listener
Outer o=new Outer(this);
b.addActionListener(o);//passing outer class instance
//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent2();
}
```

```
}
import java.awt.event.*;
class Outer implements ActionListener{
AEvent2 obj;
Outer(AEvent2 obj){
this.obj=obj;
}
public void actionPerformed(ActionEvent e){
obj.tf.setText("welcome");
}
}
```

## 3) Java event handling by anonymous class

```
import java.awt.*;
import java.awt.event.*;
class AEvent3 extends Frame{
TextField tf;
AEvent3(){
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(50,120,80,30);

b.addActionListener(new ActionListener(){
public void actionPerformed(){
tf.setText("hello");
}
});
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main(String args[]){
new AEvent3();
}
}
```

# Java ActionListener Interface

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

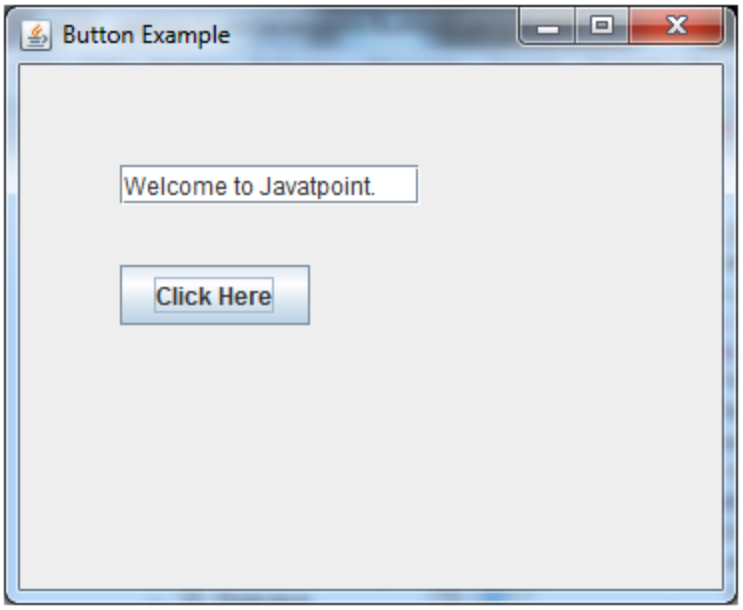
# actionPerformed() method

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

# Java JButton Example with ActionListener

```
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample {
public static void main(String[] args) {
    JFrame f=new JFrame("Button Example");
    final 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);
}
}
```

Output:




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

# Java MouseListener Interface

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

## Methods of MouseListener interface

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

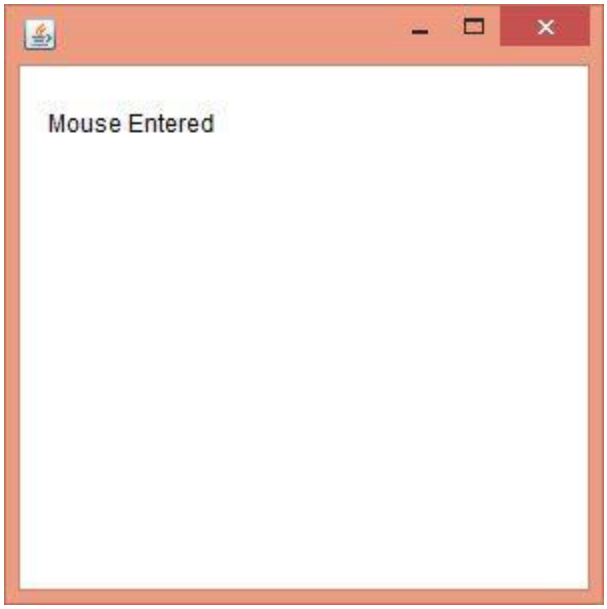
1. **public abstract void** mouseClicked(MouseEvent e);
2. **public abstract void** mouseEntered(MouseEvent e);
3. **public abstract void** mouseExited(MouseEvent e);
4. **public abstract void** mousePressed(MouseEvent e);
5. **public abstract void** mouseReleased(MouseEvent e);

## Java MouseListener Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener{
   Label l;
   MouseListenerExample(){
      addMouseListener(this);

      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");
    }
    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:




# Java MouseMotionListener Interface

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

# Methods of MouseMotionListener interface

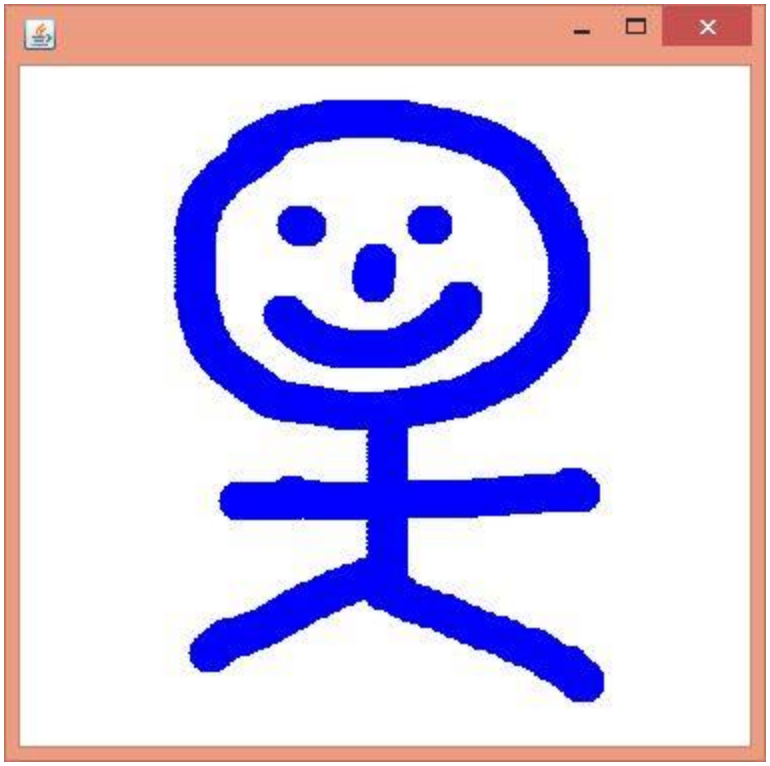
The signature of 2 methods found in MouseMotionListener interface are given below:

1. **public abstract void** mouseDragged(MouseEvent e);
2. **public abstract void** mouseMoved(MouseEvent e);

# Java MouseMotionListener Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseMotionListenerExample extends Frame implements MouseMotionListener{
   MouseMotionListenerExample(){
      addMouseMotionListener(this);

      setSize(300,300);
      setLayout(null);
      setVisible(true);
   }
public void mouseDragged(MouseEvent e) {
   Graphics g=getGraphics();
   g.setColor(Color.BLUE);
   g.fillOval(e.getX(),e.getY(),20,20);
}
public void mouseMoved(MouseEvent e) {}

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

Output:



# Java ItemListener Interface

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

# itemStateChanged() method

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

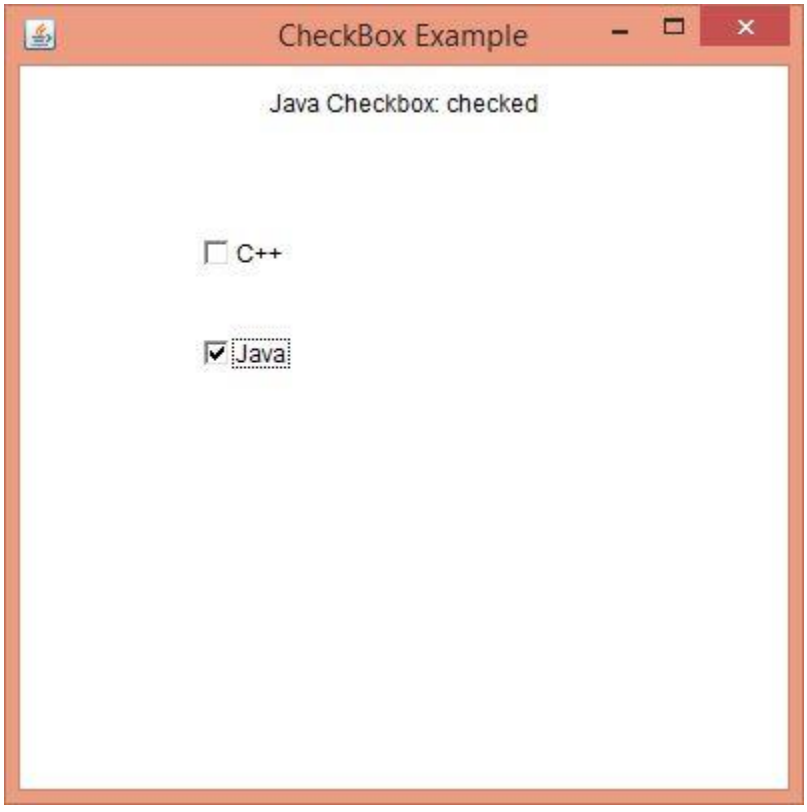
1. **public abstract void** itemStateChanged(ItemEvent e);

# Java ItemListener Example

```
import java.awt.*;
import java.awt.event.*;
public class ItemListenerExample implements ItemListener{
    Checkbox checkBox1,checkBox2;
    Label label;
    ItemListenerExample(){
        Frame f= new Frame("CheckBox Example");
        label = new Label();
        label.setAlignment(Label.CENTER);
        label.setSize(400,100);
        checkBox1 = new Checkbox("C++");
        checkBox1.setBounds(100,100, 50,50);
        checkBox2 = new Checkbox("Java");
```

```
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1); f.add(checkBox2); f.add(label);
        checkBox1.addItemListener(this);
        checkBox2.addItemListener(this);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
     }
    public void itemStateChanged(ItemEvent e) {
        if(e.getSource()==checkBox1)
            label.setText("C++ Checkbox: "
            + (e.getStateChange()==1?"checked":"unchecked"));
        if(e.getSource()==checkBox2)
        label.setText("Java Checkbox: "
        + (e.getStateChange()==1?"checked":"unchecked"));
     }
public static void main(String args[])
{
    new ItemListenerExample();
}
}
```

Output:



# Java KeyListener Interface

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

The signature of 3 methods found in KeyListener interface are given below:

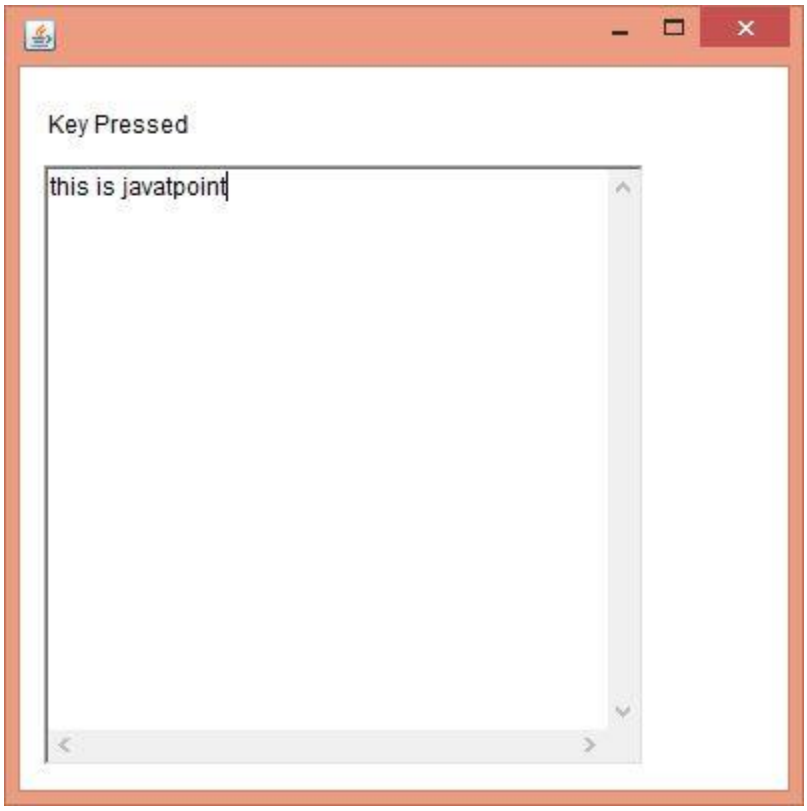
1. **public abstract void** keyPressed(KeyEvent e);
2. **public abstract void** keyReleased(KeyEvent e);
3. **public abstract void** keyTyped(KeyEvent e);

# Java KeyListener 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:




# Java WindowListener Interface

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

## Methods of WindowListener interface

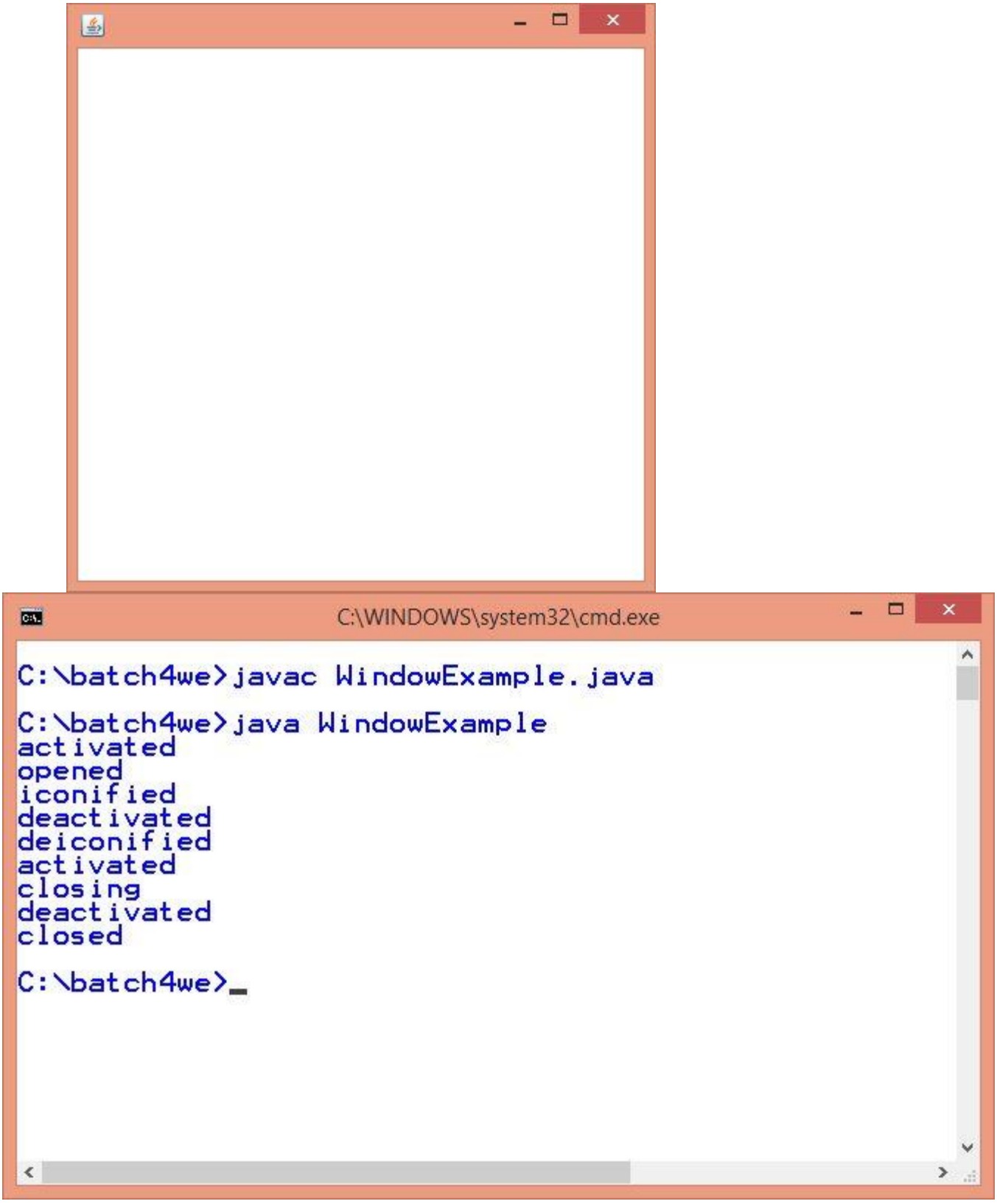
The signature of 7 methods found in WindowListener interface are given below:

1. **public abstract void** windowActivated(WindowEvent e);
2. **public abstract void** windowClosed(WindowEvent e);
3. **public abstract void** windowClosing(WindowEvent e);
4. **public abstract void** windowDeactivated(WindowEvent e);
5. **public abstract void** windowDeiconified(WindowEvent e);
6. **public abstract void** windowIconified(WindowEvent e);
7. **public abstract void** windowOpened(WindowEvent e);

# Java WindowListener Example

```
import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
public class WindowExample extends Frame implements WindowListener{
    WindowExample(){
        addWindowListener(this);

        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }

public static void main(String[] args) {
    new WindowExample();
}
public void windowActivated(WindowEvent arg0) {
    System.out.println("activated");
}
public void windowClosed(WindowEvent arg0) {
    System.out.println("closed");
}
public void windowClosing(WindowEvent arg0) {
    System.out.println("closing");
    dispose();
}
public void windowDeactivated(WindowEvent arg0) {
    System.out.println("deactivated");
}
public void windowDeiconified(WindowEvent arg0) {
    System.out.println("deiconified");
}
public void windowIconified(WindowEvent arg0) {
    System.out.println("iconified");
}
public void windowOpened(WindowEvent arg0) {
    System.out.println("opened");
}
}
```

Output:




# 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.

# java.awt.event Adapter classes

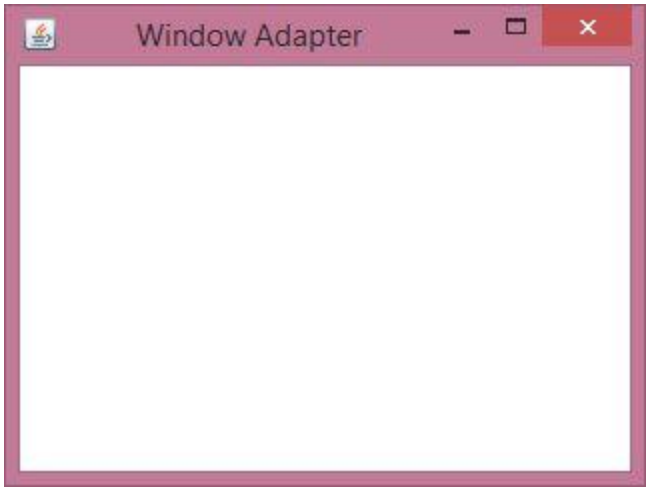
| Adapter class | Listener interface |
|---|---|
| WindowAdapter | WindowListener |
| KeyAdapter | KeyListener |
| MouseAdapter | MouseListener |
| MouseMotionAdapter | MouseMotionListener |
| FocusAdapter | FocusListener |
| ComponentAdapter | ComponentListener |
| ContainerAdapter | ContainerListener |
| 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.setLayout(null);
```

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

Output:




**19) Write a Java program that works as a simple calculator. Use a grid layout to arrange buttons for the digits and for the +, -,*, % operations. Add a text field to display the result.**

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

class calculator extends JFrame implements ActionListener
{
JLabel l1,l2,l3;
JTextField tf1,tf2,tf3;
JButton b1,b2,b3,b4;
calculator()
{
l1=new JLabel("INPUT1");
l2=new JLabel("INPUT2");
```

```
l3=new JLabel("RESULT");
tf1=new JTextField(10);
tf2=new JTextField(10);
tf3=new JTextField(10);
b1=new JButton("+");
b2=new JButton("-");
b3=new JButton("*");
b4=new JButton("%");
add(l1);
add(tf1);
add(l2);
add(tf2);
add(l3);
add(tf3);
add(b1);
add(b2);
add(b3);
add(b4);
setSize(300,300);
setTitle("CALCULATOR");
setLayout(new GridLayout(5,2,3,3));
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
setVisible(true);
}
public void actionPerformed(ActionEvent e)
```

```
{
int result=0;
int n1=Integer.parseInt(tf1.getText());
int n2=Integer.parseInt(tf2.getText());
if(e.getSource()==b1)
result=n1+n2;
else if(e.getSource()==b2)
result=n1-n2;
else if(e.getSource()==b3)
result=n1*n2;
else
result=n1%n2;
tf3.setText(result+"");
}
public static void main(String args[])
{
new calculator();
}
}
```

# 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.

## Advantage of Applet

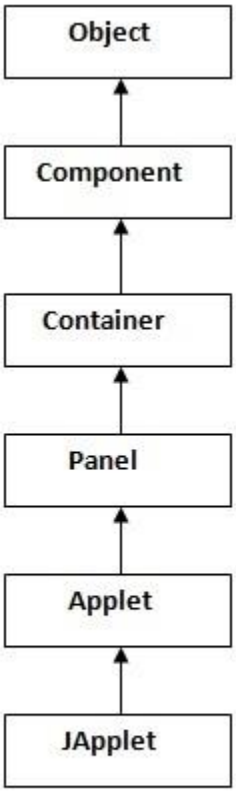
There are many advantages of applet. They are as follows:

- 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.

## Q2. What are the differences between applications and applets?

*Answer:*

Though the applications and applets are written in Java, there is a lot of difference between their initialization, execution and utilization of system resources.

Basic differences between an Application and Applet are:

| Property | Application | Applet |
|---|---|---|
| 1. **main( ) method** | exists | does not exist |
| 2. **Nature** | stand-alone programs | can't be stand-alone |
| 3. **Execution** | needs a Java interpreter | needs a browser like Netscape |
| 4. **Security** | does not need any security | needs top-most security for hard disk files |
| 5. **Restrictions** | no hard disk accessing restrictions | can't access hard disk files, by default |
| 6. **Extra Software** | can share any software available | can't share eg. ActiveX controls in the system |
| 7. **Plug-ins** | latest additions of software can be embedded through plug-ins | can't use plug-ins directly including browser plug-ins that are incorporated on the user's system. |

## Q3. What are applet security restrictions?

*Answer :*

Because Java applets are run on a Web user's system, there are serious restrictions on the execution of an applet. An applet can be downloaded from Internet and can be executed on our system. We do not know who developed the applet and the applet may have some virus or malicious code that may corrupt our system files or may collect our personal information available on our system. For this reason, we require security for our system resources when an applet executes on our system.

Java designers restrict the applet's functionality. As a general rule, an applet cannot do any of the following:

- They cannot read or write files on the user file system.
- They cannot communicate with an Internet site other than the one that served the Web page that included the applet.
- They cannot run any programs on the user's system.
- They cannot load programs stored on the user's system, such as executable programs and shared libraries.

## 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 4 life cycle methods and java.awt.Component class provides 1 life cycle methods 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.

## Who is responsible to manage the life cycle of an applet?

Java Plug-in software.

## 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.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{

public void paint(Graphics g){
g.drawString("welcome",150,150);
}

}
```

***Note: class must be public because its object is created by Java Plugin software that resides on the browser.***

## myapplet.html

```
1. <html>
2. <body>
3. <applet code="First.class" width="300" height="300">
4. </applet>
5. </body>
6. </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.

```
//First.java
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
```

# Displaying Graphics in Applet

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 of Graphics in applet:

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

public class GraphicsDemo extends Applet{
```

```
public void paint(Graphics g){
g.setColor(Color.red);
g.drawString("Welcome",50, 50);
g.drawLine(20,30,20,300);
g.drawRect(70,100,30,30);
g.fillRect(170,100,30,30);
g.drawOval(70,200,30,30);

g.setColor(Color.pink);
g.fillOval(170,200,30,30);
g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);

}
}
```

## myapplet.html

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

# EventHandling in Applet

As we perform event handling in AWT or Swing, we can perform it in applet also. Let's see the simple example of event handling in applet that prints a message by click on the button.

# Example of EventHandling in applet:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class EventApplet extends Applet implements ActionListener{
Button b;
TextField tf;

public void init(){
tf=new TextField();
```

```
tf.setBounds(30,40,150,20);

b=new Button("Click");
b.setBounds(80,150,60,50);

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

# JApplet class in Applet

As we prefer Swing to AWT. Now we can use JApplet that can have all the controls of swing. The JApplet class extends the Applet class.

# Example of EventHandling in JApplet:

```
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 Applet

We can perform painting operation in applet by the mouseDragged() method of MouseMotionListener.

# Example of Painting in Applet:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class MouseDrag extends Applet implements MouseMotionListener{

public void init(){
addMouseMotionListener(this);
setBackground(Color.red);
```

```
}

public void mouseDragged(MouseEvent me){
Graphics g=getGraphics();
g.setColor(Color.white);
g.fillOval(me.getX(),me.getY(),5,5);
}
public void mouseMoved(MouseEvent me){}

}
```

In the above example, getX() and getY() method of MouseEvent is used to get the current x-axis and y-axis. The getGraphics() method of Component class returns the object of Graphics.

## myapplet.html

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

# 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:

1. **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>
```