

# UNIT - V GUI Programming with Swing

## Introduction

**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.

## limitations of AWT, MVC architecture

**Limitations of AWT:** The AWT defines a basic set of controls, windows, and dialog boxes that support a usable, but limited graphical interface. One reason for the limited nature of the AWT is that it translates its various visual components into their corresponding, platform-specific equivalents or peers. This means that the look and feel of a component is defined by the platform, not by java. Because the AWT components use native code resources, they are referred to as heavy weight. The use of native peers led to several problems. First, because of variations between operating systems, a component might look, or even act, differently on different platforms. This variability threatened java's philosophy: write once, run anywhere. Second, the look and feel of each component was fixed and could not be changed. Third, the use of heavyweight components caused some frustrating restrictions. Due to these limitations Swing came and was integrated to java. Swing is built on the AWT. Two key Swing features are: Swing components are light weight, Swing supports a pluggable look and feel.

## The MVC Connection:

In general, a visual component is a composite of three distinct aspects:

- The way that the component looks when rendered on the screen
- The way that the component reacts to the user
- The state information associated with the component.

The Model-View-Controller architecture is successful for all these.

## Components and Containers:

A component is an independent visual control, such as a push button. A container holds a group of components. Furthermore in order for a component to be displayed it must be held within a container. Swing components are derived from JComponent class. Note that all component classes begin with the letter J. For example a label is JLabel, a button is JButton

etc. Swing defines two types of containers. The first are top level containers: JFrame, JApplet, JWindow, and JDialog. These containers do not inherit JComponent. They do, however inherit the AWT classes Container and Component. Unlike Swing's other components which are heavy weight, the top level containers are heavy weight. The second type of containers are light weight inherit from JComponent. Example- Jpanel.

### Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. The **Java LayoutManagers** facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers. There are the following classes that represent the layout managers:

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

### Java BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west, and center. Each region (area) may contain one component only. It is the default layout of a 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.
- **BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example of BorderLayout class: Using BorderLayout() constructor

**FileName:** Border.java

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

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

        // creating buttons
        JButton b1 = new JButton("NORTH"); // the button will be labeled as NORTH
        JButton b2 = new JButton("SOUTH"); // the button will be labeled as SOUTH
        JButton b3 = new JButton("EAST"); // the button will be labeled as EAST
        JButton b4 = new JButton("WEST"); // the button will be labeled as WEST
        JButton b5 = new JButton("CENTER"); // the button will be labeled as CENTER

        f.add(b1, BorderLayout.NORTH); // b1 will be placed in the North Direction
        f.add(b2, BorderLayout.SOUTH); // b2 will be placed in the South Direction
        f.add(b3, BorderLayout.EAST); // b2 will be placed in the East Direction
        f.add(b4, BorderLayout.WEST); // b2 will be placed in the West Direction
        f.add(b5, BorderLayout.CENTER); // b2 will be placed in the Center

        f.setSize(300, 300);
        f.setVisible(true);
    }
}
```

```

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

```

**Output:**



Example of BorderLayout class: Using BorderLayout(int hgap, int vgap) constructor

The following example inserts horizontal and vertical gaps between buttons using the parameterized constructor BorderLayout(int hgap, int gap)

**FileName:** BorderLayoutExample.java

```

// import statement
import java.awt.*;
import javax.swing.*;
public class BorderLayoutExample
{
    JFrame jframe;
    // constructor

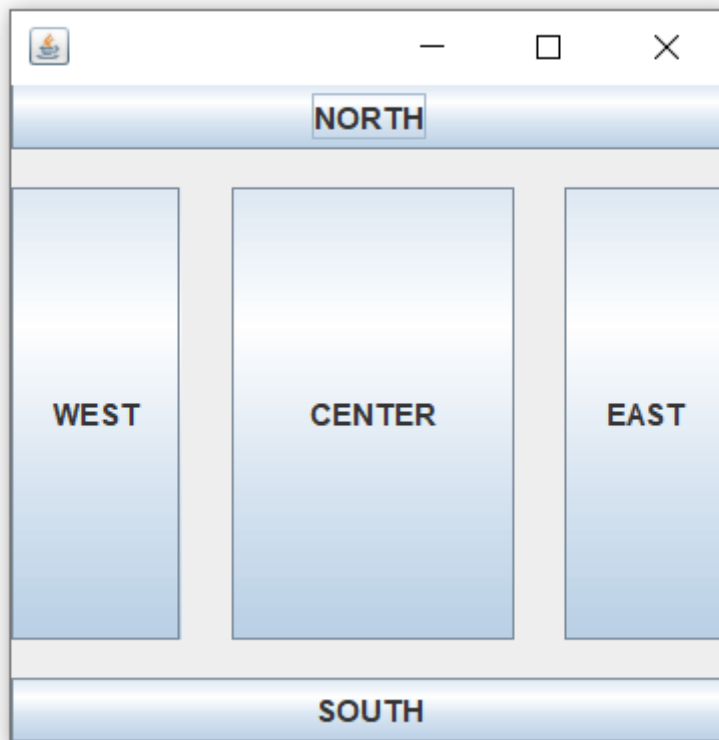
```

```

BorderLayoutExample()
{
    // creating a Frame
    JFrame jframe = new JFrame();
    // create buttons
    JButton btn1 = new JButton("NORTH");
    JButton btn2 = new JButton("SOUTH");
    JButton btn3 = new JButton("EAST");
    JButton btn4 = new JButton("WEST");
    JButton btn5 = new JButton("CENTER");
    // creating an object of the BorderLayout class using
    // the parameterized constructor where the horizontal gap is 20
    // and vertical gap is 15. The gap will be evident when buttons are placed
    // in the frame
    jframe.setLayout(new BorderLayout(20, 15));
    jframe.add(btn1, BorderLayout.NORTH);
    jframe.add(btn2, BorderLayout.SOUTH);
    jframe.add(btn3, BorderLayout.EAST);
    jframe.add(btn4, BorderLayout.WEST);
    jframe.add(btn5, BorderLayout.CENTER);
    jframe.setSize(300,300);
    jframe.setVisible(true);
}
// main method
public static void main(String args[]) {
    new BorderLayoutExample();
}
}

```

**Output:**



### Java BorderLayout: Without Specifying Region

The add() method of the JFrame class can work even when we do not specify the region. In such a case, only the latest component added is shown in the frame, and all the components added previously get discarded. The latest component covers the whole area. The following example shows the same.

**FileName:** BorderLayoutWithoutRegionExample.java

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

public class BorderLayoutWithoutRegionExample {
    JFrame jframe;

    // constructor
    BorderLayoutWithoutRegionExample()
    {
```

```

jframe = new JFrame();

JButton btn1 = new JButton("NORTH");
JButton btn2 = new JButton("SOUTH");
JButton btn3 = new JButton("EAST");
JButton btn4 = new JButton("WEST");
JButton btn5 = new JButton("CENTER");

// horizontal gap is 7, and the vertical gap is 7
// Since region is not specified, the gaps are of no use
jframe.setLayout(new BorderLayout(7, 7));

// each button covers the whole area
// however, the btn5 is the latest button
// that is added to the frame; therefore, btn5
// is shown
jframe.add(btn1);
jframe.add(btn2);
jframe.add(btn3);
jframe.add(btn4);
jframe.add(btn5);

jframe.setSize(300,300);
jframe.setVisible(true);
}

// main method
public static void main(String argsv[])
{
    new BorderLayoutWithoutRegionExample();
}
}

```

## Output:

### Java GridLayout

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

### Constructors of GridLayout class

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

### Example of GridLayout class: Using GridLayout() Constructor

The GridLayout() constructor creates only one row. The following example shows the usage of the parameterless constructor.

**FileName:** GridLayoutExample.java

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

public class GridLayoutExample
{
    JFrame frameObj;

    // constructor
    GridLayoutExample()
    {
```

```

frameObj = new JFrame();

// creating 9 buttons
JButton btn1 = new JButton("1");
JButton btn2 = new JButton("2");
JButton btn3 = new JButton("3");
JButton btn4 = new JButton("4");
JButton btn5 = new JButton("5");
JButton btn6 = new JButton("6");
JButton btn7 = new JButton("7");
JButton btn8 = new JButton("8");
JButton btn9 = new JButton("9");

// adding buttons to the frame
// since, we are using the parameterless constructor, therefore;
// the number of columns is equal to the number of buttons we
// are adding to the frame. The row count remains one.
frameObj.add(btn1); frameObj.add(btn2); frameObj.add(btn3);
frameObj.add(btn4); frameObj.add(btn5); frameObj.add(btn6);
frameObj.add(btn7); frameObj.add(btn8); frameObj.add(btn9);

// setting the grid layout using the parameterless constructor
frameObj.setLayout(new GridLayout());

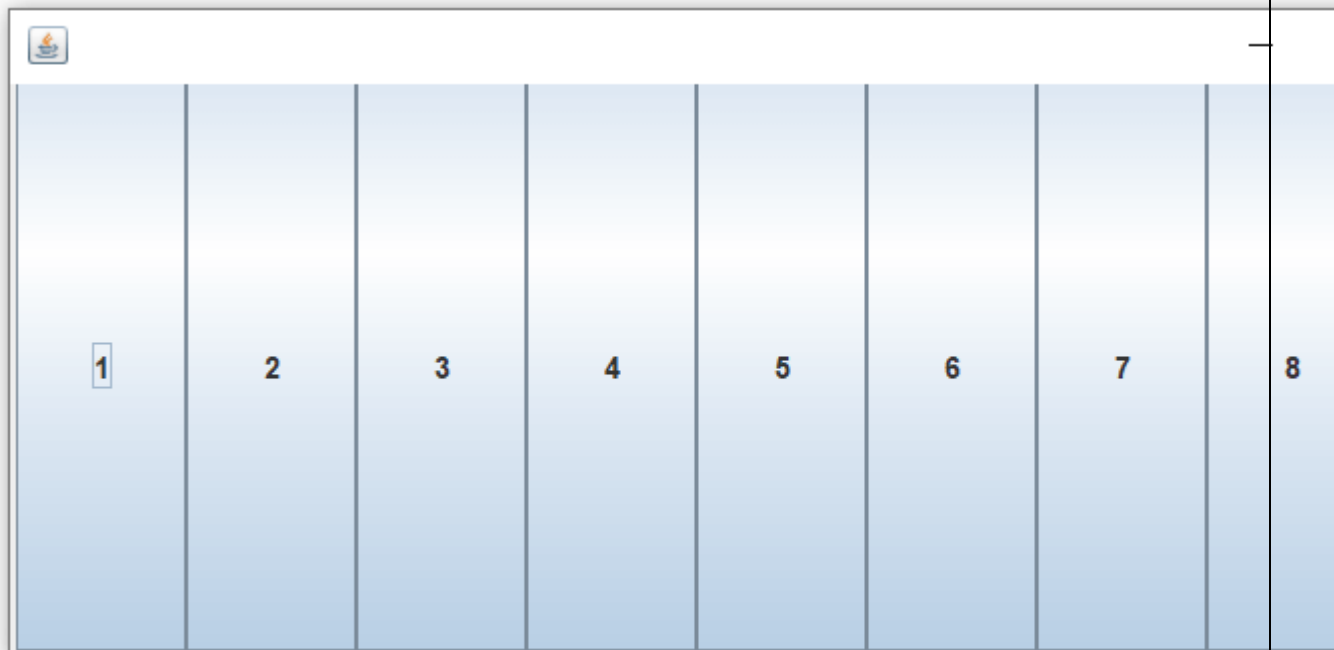
frameObj.setSize(300, 300);
frameObj.setVisible(true);
}

// main method
public static void main(String argsv[])
{
    new GridLayoutExample();
}

```

```
}  
}
```

**Output:**



## Java FlowLayout

The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the 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: Using FlowLayout() constructor

**FileName:** FlowLayoutExample.java

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

public class FlowLayoutExample
{

    JFrame frameObj;

    // constructor
    FlowLayoutExample()
    {
```

```

// creating a frame object
frameObj = new JFrame();

// creating the buttons
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");
JButton b10 = new JButton("10");

// adding the buttons to frame
frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);
frameObj.add(b5); frameObj.add(b6); frameObj.add(b7); frameObj.add(b8);
frameObj.add(b9); frameObj.add(b10);

// parameter less constructor is used
// therefore, alignment is center
// horizontal as well as the vertical gap is 5 units.
frameObj.setLayout(new FlowLayout());

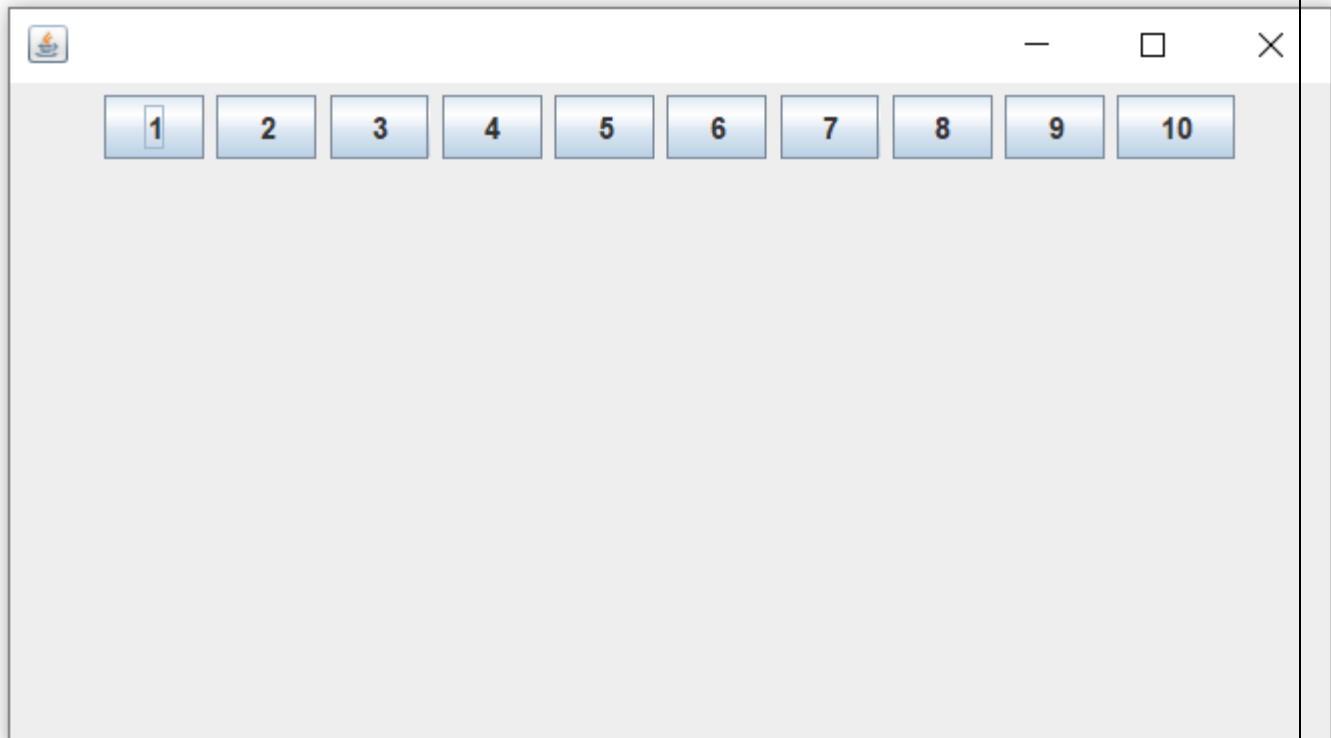
frameObj.setSize(300, 300);
frameObj.setVisible(true);
}

// main method
public static void main(String args[])
{

```

```
new FlowLayoutExample();  
}  
}
```

**Output:**



## Java CardLayout

The **Java 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: Using Default Constructor

The following program uses the next() method to move to the next card of the container.

**FileName:** CardLayoutExample1.java

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

public class CardLayoutExample1 extends JFrame implements ActionListener
{

    CardLayout crd;

    // button variables to hold the references of buttons
    JButton btn1, btn2, btn3;
    Container cPane;

    // constructor of the class
    CardLayoutExample1()
    {
```

```
cPane = getContentPane();
```

```
//default constructor used
```

```
// therefore, components will
```

```
// cover the whole area
```

```
crd = new CardLayout();
```

```
cPane.setLayout(crd);
```

```
// creating the buttons
```

```
btn1 = new JButton("Apple");
```

```
btn2 = new JButton("Boy");
```

```
btn3 = new JButton("Cat");
```

```
// adding listeners to it
```

```
btn1.addActionListener(this);
```

```
btn2.addActionListener(this);
```

```
btn3.addActionListener(this);
```

```
cPane.add("a", btn1); // first card is the button btn1
```

```
cPane.add("b", btn2); // first card is the button btn2
```

```
cPane.add("c", btn3); // first card is the button btn3
```

```
}
```

```
public void actionPerformed(ActionEvent e)
```

```
{
```

```
// Upon clicking the button, the next card of the container is shown
```

```
// after the last card, again, the first card of the container is shown upon clicking
```

```
crd.next(cPane);
```

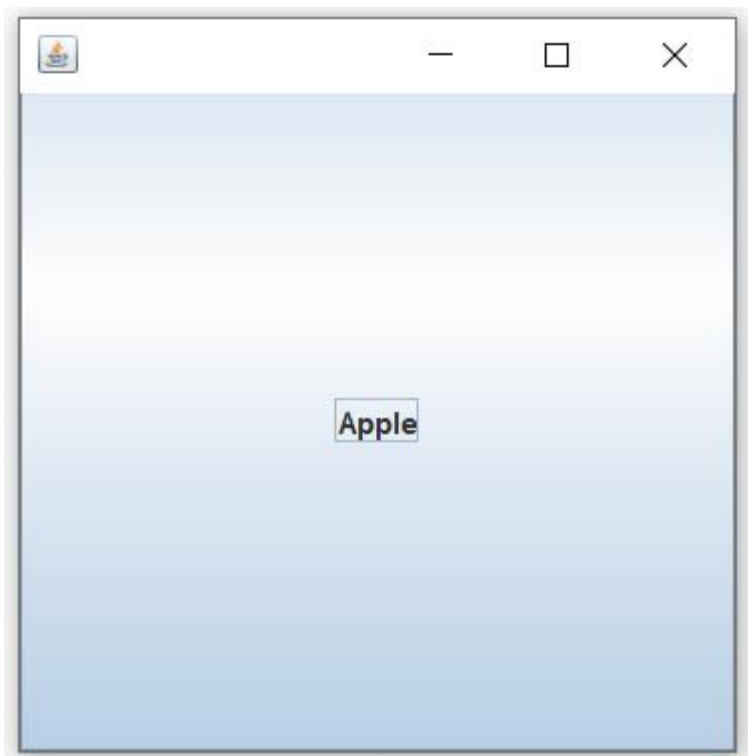
```
}
```

```
// main method
```

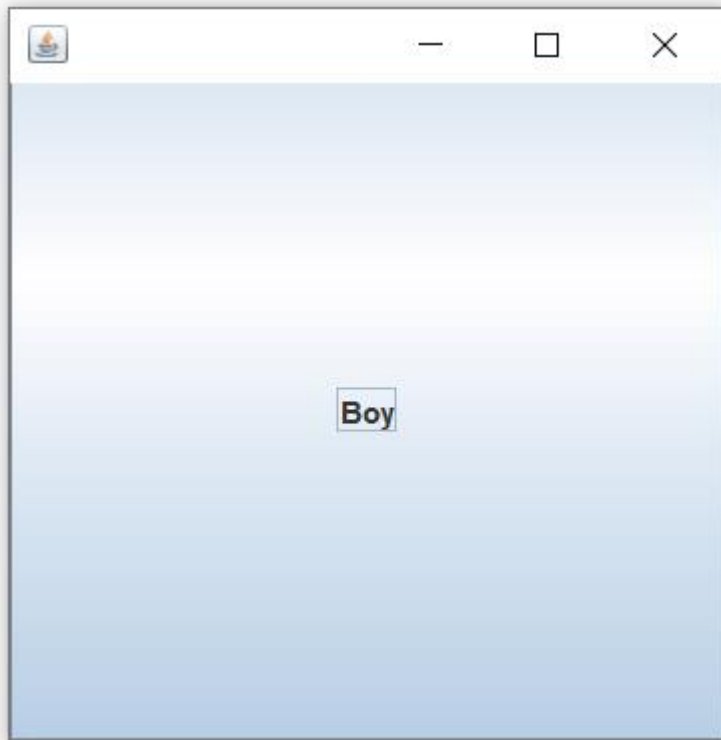
```
public static void main(String args[])
{
    // creating an object of the class CardLayoutExample1
    CardLayoutExample1 crdl = new CardLayoutExample1();

    // size is 300 * 300
    crdl.setSize(300, 300);
    crdl.setVisible(true);
    crdl.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}
```

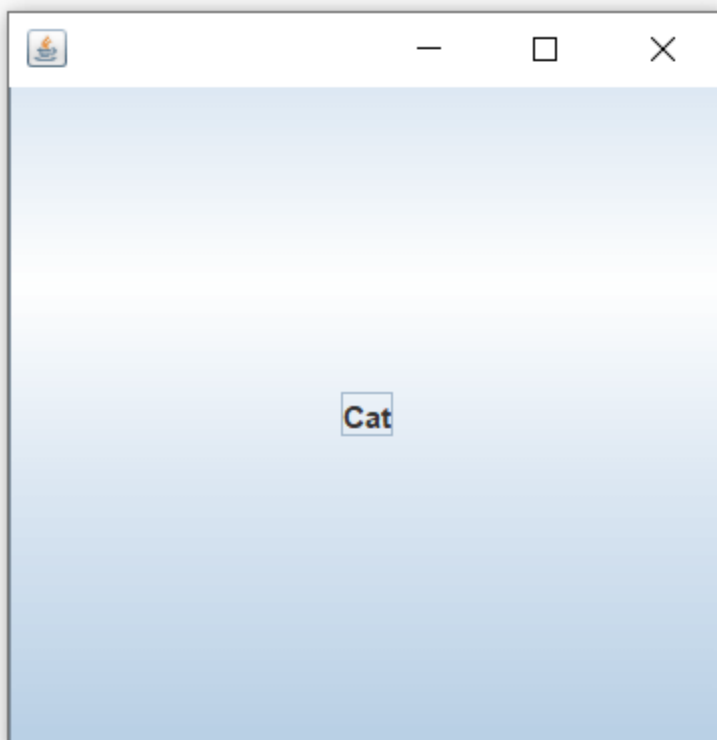
**Output:**



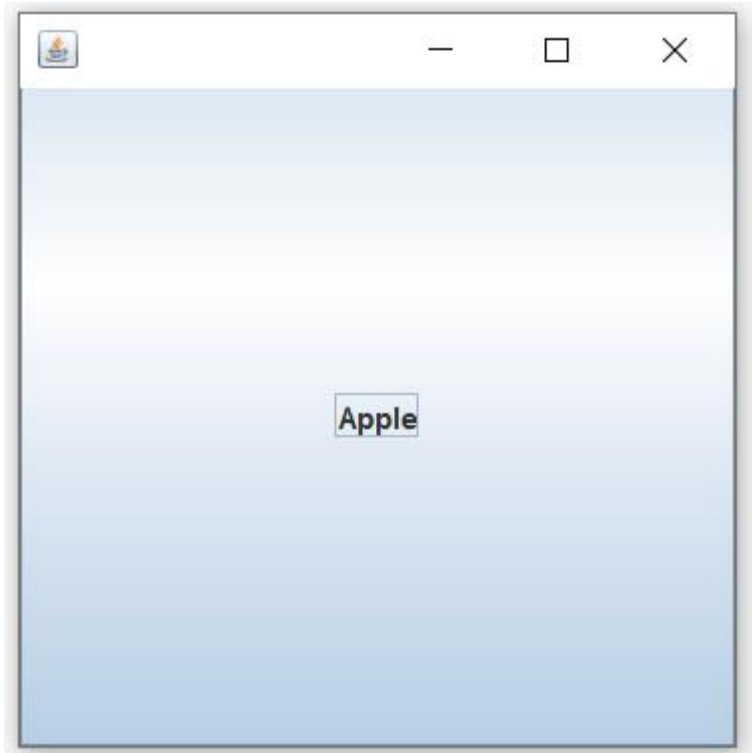
When the button named apple is clicked, we get



When the boy button is clicked, we get



Again, we reach the first card of the container if the cat button is clicked, and the cycle continues.



### Java GridBagLayout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.

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 GridBagConstraints	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	PREFERRED_SIZE	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.

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 GridBagConstraints		getLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.
protected GridBagConstraints		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.
Department of CSE			Page 20 of 63

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

The components may not be of the 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 the constraints object, we arrange the component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine the component's size. GridBagLayout components are also arranged in the rectangular grid but can have many different sizes and can occupy multiple rows or columns.

Constructor

**GridBagLayout():** The parameterless constructor is used to create a grid bag layout manager.

GridBagLayout Methods

Example 1

**FileName:** GridBagLayoutExample.java

```
import java.awt.Button;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
```

```

import javax.swing.*;

public class GridBagLayoutExample extends JFrame{

    public static void main(String[] args) {

        GridBagLayoutExample a = new GridBagLayoutExample();

    }

    public GridBagLayoutExample() {
        GridBagLayoutgrid = new GridBagLayout();
        GridBagConstraints gbc = new GridBagConstraints();
        setLayout(grid);
        setTitle("GridBag Layout Example");
        GridBagLayout layout = new GridBagLayout();
        this.setLayout(layout);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridx = 0;
        gbc.gridy = 0;
        this.add(new Button("Button One"), gbc);
        gbc.gridx = 1;
        gbc.gridy = 0;
        this.add(new Button("Button two"), gbc);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.ipady = 20;
        gbc.gridx = 0;
        gbc.gridy = 1;
        this.add(new Button("Button Three"), gbc);
        gbc.gridx = 1;
        gbc.gridy = 1;
        this.add(new Button("Button Four"), gbc);
        gbc.gridx = 0;
        gbc.gridy = 2;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridwidth = 2;
        this.add(new Button("Button Five"), gbc);
    }
}

```

```

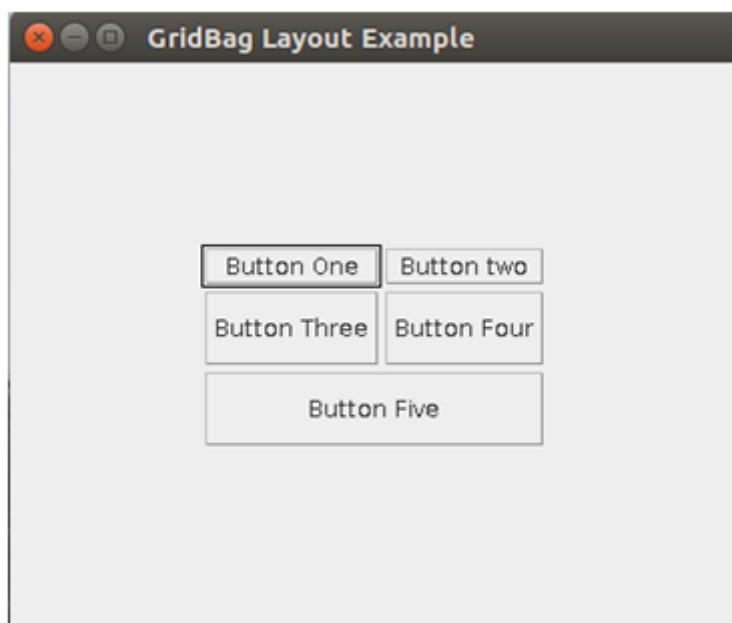
        setSize(300, 300);
        setPreferredSize(getSize());
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

    }

}

```

**Output:**



## Event Handling

### Delegation Event Model in Java

The Delegation Event model is defined to handle events in GUI [programming languages](#)

. The [GUI](#)

stands for Graphical User Interface, where a user graphically/visually interacts with the system.

The GUI programming is inherently event-driven; whenever a user initiates an activity such as a mouse activity, clicks, scrolling, etc., each is known as an event that is mapped to a code to respond to functionality to the user. This is known as event handling.

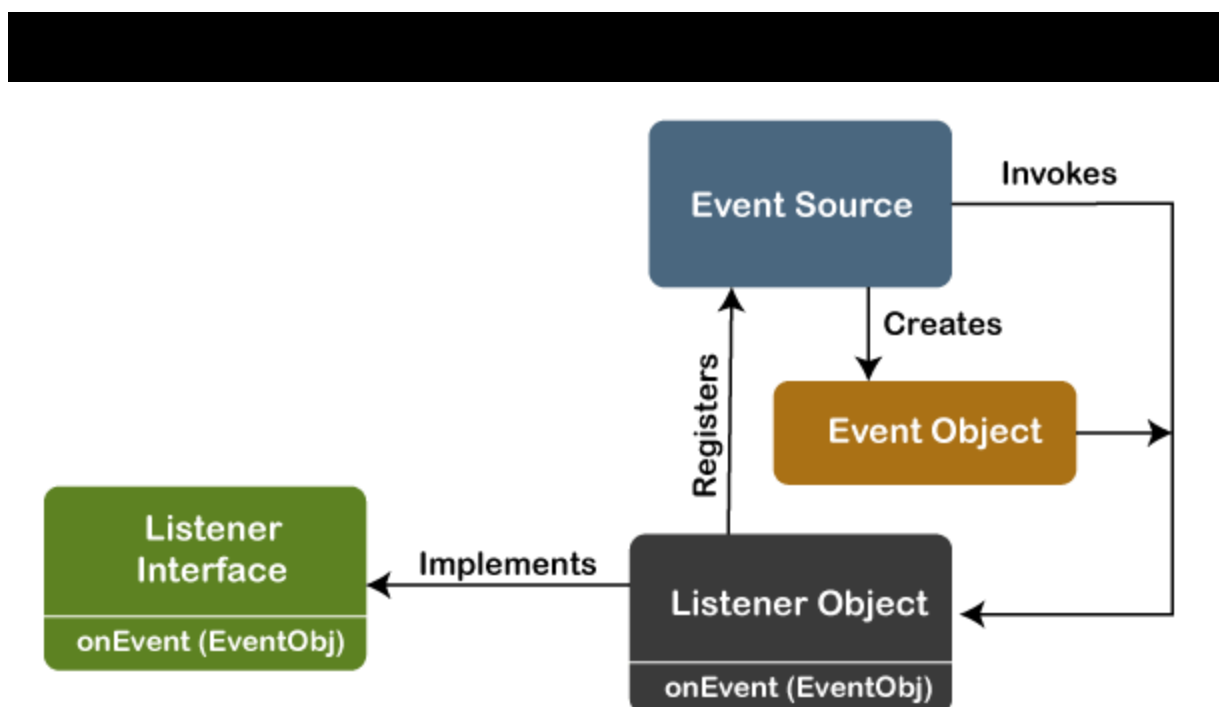
In this section, we will discuss event processing and how to implement the delegation event model in [Java](#)

. We will also discuss the different components of an Event Model.

### Event Processing in Java

Java support event processing since Java 1.0. It provides support for [AWT \( Abstract Window Toolkit\)](#)

, which is an API used to develop the Desktop application. In Java 1.0, the AWT was based on inheritance. To catch and process GUI events for a program, it should hold subclass GUI components and override action() or handleEvent() methods. The below image demonstrates the event processing.



But, the modern approach for event processing is based on the Delegation Model. It defines a standard and compatible mechanism to generate and process events. In this model, a source generates an event and forwards it to one or more listeners. The listener waits until it receives an event. Once it receives the event, it is processed by the listener and returns it. The UI elements are able to delegate the processing of an event to a separate function.

The key advantage of the Delegation Event Model is that the application logic is completely separated from the interface logic.

In this model, the listener must be connected with a source to receive the event notifications. Thus, the events will only be received by the listeners who wish to receive them. So, this approach is more convenient than the inheritance-based event model (in Java 1.0).

In the older model, an event was propagated up the containment until a component was handled. This needed components to receive events that were not processed, and it took lots of time. The Delegation Event model overcame this issue.

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

- Events
- Events Sources
- Events Listeners

## Events

The Events are the objects that define state change in a source. An event can be generated as a reaction of a user while interacting with GUI elements. Some of the event generation activities are moving the mouse pointer, clicking on a button, pressing the keyboard key, selecting an item from the list, and so on. We can also consider many other user operations as events.

The Events may also occur that may be not related to user interaction, such as a timer expires, counter exceeded, system failures, or a task is completed, etc. We can define events for any of the applied actions.

## Event Sources

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

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

### 1. **public void** addTypeListener (TypeListener e1)

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

Some listeners allow only one listener to register. Below is an example:

1. **public void** addTypeListener(TypeListener e2) **throws** java.util.TooManyListenersException

From the above syntax, the Type is the name of the event, and e2 is the event listener's reference. When the specified event occurs, it will be notified to the registered listener. This process is known as **unicasting** events.

A source should contain a method that unregisters a specific type of event from the listener if not needed. Below is an example of the method that will remove the event from the listener.

1. **public void** removeTypeListener(TypeListener e2?)

From the above syntax, the Type is an event name, and e2 is the reference of the listener. For example, to remove the keyboard listener, the **removeKeyListener()** method will be called.

The source provides the methods to add or remove listeners that generate the events. For example, the Component class contains the methods to operate on the different types of events, such as adding or removing them from the listener.

### Event Listeners

An event listener is an object that is invoked when an event triggers. The listeners require two things; first, it must be registered with a source; however, it can be registered with several resources to receive notification about the events. Second, it must implement the methods to receive and process the received notifications.

The methods that deal with the events are defined in a set of interfaces. These interfaces can be found in the java.awt.event package.

For example, the **MouseMotionListener** interface provides two methods when the mouse is dragged and moved. Any object can receive and process these events if it implements the MouseMotionListener interface.

### Types of Events

The events are categorized into the following two categories:

#### The Foreground Events:

The foreground events are those events that require direct interaction of the user. These types of events are generated as a result of user interaction with the GUI component. For example, clicking on a button, mouse movement, pressing a keyboard key, selecting an option from the list, etc.

### The Background Events :

The Background events are those events that result from the interaction of the end-user. For example, an Operating system interrupts system failure (Hardware or Software).

To handle these events, we need an event handling mechanism that provides control over the events and responses.

### Handling Mouse Events

Mouse events occur with mouse movements in forms and controls. Following are the various mouse events related with a Control class –

- **MouseDown** – it occurs when a mouse button is pressed
- **MouseEnter** – it occurs when the mouse pointer enters the control
- **MouseHover** – it occurs when the mouse pointer hovers over the control
- **MouseLeave** – it occurs when the mouse pointer leaves the control
- **MouseMove** – it occurs when the mouse pointer moves over the control
- **MouseUp** – it occurs when the mouse pointer is over the control and the mouse button is released
- **MouseWheel** – it occurs when the mouse wheel moves and the control has focus

The event handlers of the mouse events get an argument of type **MouseEventArgs**. The MouseEventArgs object is used for handling mouse events. It has the following properties –

- **Buttons** – indicates the mouse button pressed
- **Clicks** – indicates the number of clicks
- **Delta** – indicates the number of detents the mouse wheel rotated
- **X** – indicates the x-coordinate of mouse click
- **Y** – indicates the y-coordinate of mouse click

### Example

Following is an example, which shows how to handle mouse events. Take the following steps –

- Add three labels, three text boxes and a button control in the form.
- Change the text properties of the labels to - Customer ID, Name and Address, respectively.
- Change the name properties of the text boxes to txtID, txtName and txtAddress, respectively.
- Change the text property of the button to 'Submit'.
- Add the following code in the code editor window –

#### Public Class Form1

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialsponit.com"
End Sub
```

```

Private Sub txtID_MouseEnter(sender As Object, e As EventArgs) _
    Handles txtID.MouseEnter
    'code for handling mouse enter on ID textbox
    txtID.BackColor = Color.CornflowerBlue
    txtID.ForeColor = Color.White
End Sub

Private Sub txtID_MouseLeave(sender As Object, e As EventArgs) _
    Handles txtID.MouseLeave
    'code for handling mouse leave on ID textbox
    txtID.BackColor = Color.White
    txtID.ForeColor = Color.Blue
End Sub

Private Sub txtName_MouseEnter(sender As Object, e As EventArgs) _
    Handles txtName.MouseEnter
    'code for handling mouse enter on Name textbox
    txtName.BackColor = Color.CornflowerBlue
    txtName.ForeColor = Color.White
End Sub

Private Sub txtName_MouseLeave(sender As Object, e As EventArgs) _
    Handles txtName.MouseLeave
    'code for handling mouse leave on Name textbox
    txtName.BackColor = Color.White
    txtName.ForeColor = Color.Blue
End Sub

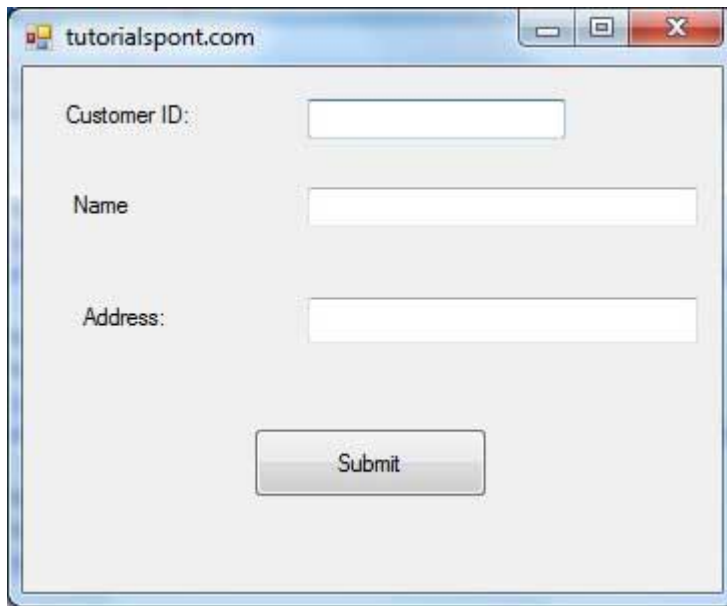
Private Sub txtAddress_MouseEnter(sender As Object, e As EventArgs) _
    Handles txtAddress.MouseEnter
    'code for handling mouse enter on Address textbox
    txtAddress.BackColor = Color.CornflowerBlue
    txtAddress.ForeColor = Color.White
End Sub

Private Sub txtAddress_MouseLeave(sender As Object, e As EventArgs) _
    Handles txtAddress.MouseLeave
    'code for handling mouse leave on Address textbox
    txtAddress.BackColor = Color.White
    txtAddress.ForeColor = Color.Blue
End Sub

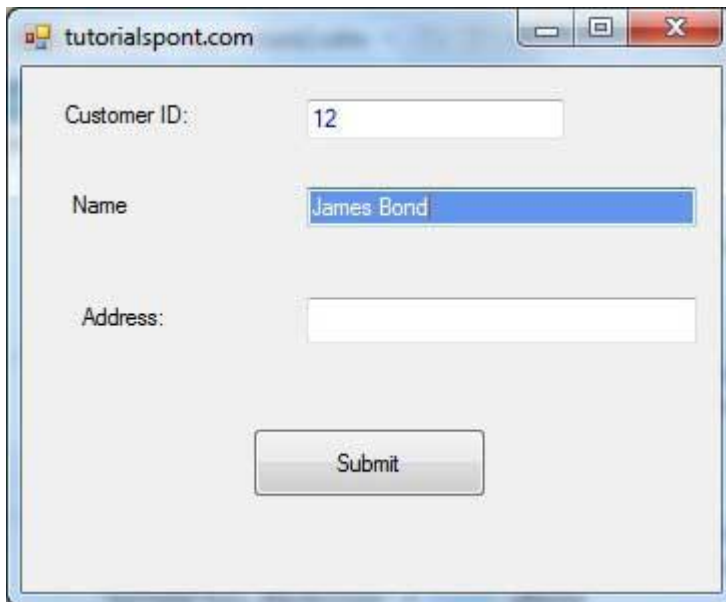
Private Sub Button1_Click(sender As Object, e As EventArgs) _
    Handles Button1.Click
    MsgBox("Thank you " & txtName.Text & ", for your kind cooperation")
End Sub
End Class

```

When the above code is executed and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window –



Try to enter text in the text boxes and check the mouse events –



### Handling Keyboard Events

Following are the various keyboard events related with a Control class –

- **KeyDown** – occurs when a key is pressed down and the control has focus
- **KeyPress** – occurs when a key is pressed and the control has focus
- **KeyUp** – occurs when a key is released while the control has focus

The event handlers of the KeyDown and KeyUp events get an argument of type **KeyEventArgs**. This object has the following properties –

- **Alt** – it indicates whether the ALT key is pressed
- **Control** – it indicates whether the CTRL key is pressed
- **Handled** – it indicates whether the event is handled
- **KeyCode** – stores the keyboard code for the event
- **KeyData** – stores the keyboard data for the event

- **KeyValue** – stores the keyboard value for the event
- **Modifiers** – it indicates which modifier keys (Ctrl, Shift, and/or Alt) are pressed
- **Shift** – it indicates if the Shift key is pressed

The event handlers of the KeyDown and KeyUp events get an argument of type **KeyEventArgs**. This object has the following properties –

- **Handled** – indicates if the KeyPress event is handled
- **KeyChar** – stores the character corresponding to the key pressed

Example

Let us continue with the previous example to show how to handle keyboard events. The code will verify that the user enters some numbers for his customer ID and age.

- Add a label with text Property as 'Age' and add a corresponding text box named txtAge.
- Add the following codes for handling the KeyUP events of the text box txtID.

```
Private Sub txtID_KeyUP(sender As Object, e As KeyEventArgs) _
    Handles txtID.KeyUp
```

```
    If (Not Char.IsNumber(ChrW(e.KeyCode))) Then
        MessageBox.Show("Enter numbers for your Customer ID")
        txtID.Text = " "
```

```
    End If
```

```
End Sub
```

- Add the following codes for handling the KeyUP events of the text box txtID.

```
Private Sub txtAge_KeyUP(sender As Object, e As KeyEventArgs) _
    Handles txtAge.KeyUp
```

```
    If (Not Char.IsNumber(ChrW(e.keyCode))) Then
        MessageBox.Show("Enter numbers for age")
        txtAge.Text = " "
```

```
    End If
```

```
End Sub
```

When the above code is executed and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window –

A screenshot of a web browser window displaying a form from 'tutorialspont.com'. The form contains four input fields: 'Customer ID:', 'Name', 'Address:', and 'Age'. Each field is currently empty. Below the fields is a 'Submit' button.

If you leave the text for age or ID as blank or enter some non-numeric data, it gives a warning message box and clears the respective text –

A screenshot of the same web browser window. The form now contains the following data: 'Customer ID:' is '12', 'Name' is 'James Bond', 'Address:' is 'California, US', and 'Age' is '1n'. A warning message box is displayed over the form, containing the text 'Enter numbers for age' and an 'OK' button. The 'Submit' button is still visible at the bottom of the form.

### 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*.

**Pros of using Adapter classes:**

- It assists the unrelated classes to work combinedly.
- It provides ways to use classes in different ways.

- It increases the transparency of classes.
- It provides a way to include related patterns in the class.
- It provides a pluggable kit for developing an application.
- It increases the reusability of the class.

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	<u>WindowListener</u>
KeyAdapter	<u>KeyListener</u>
MouseAdapter	<u>MouseListener</u>
MouseMotionAdapter	<u>MouseMotionListener</u>
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

#### java.awt.dnd Adapter classes

Adapter class	Listener interface
---------------	--------------------

DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

### javax.swing.event Adapter classes

Adapter class	Listener interface
MouseInputAdapter	MouseInputListener
InternalFrameAdapter	InternalFrameListener

### Java WindowAdapter Example

In the following example, we are implementing the WindowAdapter class of AWT and one its methods windowClosing() to close the frame window.

#### AdapterExample.java

// importing the necessary libraries

```
import java.awt.*;
```

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

```
public class AdapterExample {
```

// object of Frame

```
    Frame f;
```

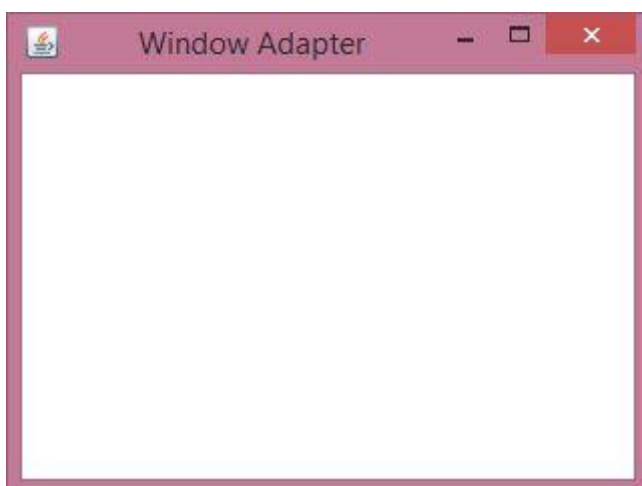
// class constructor

```
    AdapterExample() {
```

```
// creating a frame with the title
    f = new Frame ("Window Adapter");
// adding the WindowListener to the frame
// overriding the windowClosing() method
    f.addWindowListener (new WindowAdapter() {
        public void windowClosing (WindowEvent e) {
            f.dispose();
        }
    });
// setting the size, layout and
f.setSize (400, 400);      f.setLayout (null);
    f.setVisible (true);
}

// main method
public static void main(String[] args) {
    new AdapterExample();
}
}
```

**Output:**



## Java Inner Classes (Nested Classes)

**Java inner class** or nested class is a class that is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place to be more readable and maintainable.

Additionally, it can access all the members of the outer class, including private data members and methods.

### *Syntax of Inner class*

```
class Java_Outer_class{  
    //code  
    class Java_Inner_class{  
        //code  
    }  
}
```

### Advantage of Java inner classes

There are three advantages of inner classes in Java. They are as follows:

1. Nested classes represent a particular type of relationship that is **it can access all the members (data members and methods) of the outer class**, including private.
2. Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.
3. **Code Optimization:** It requires less code to write.

### Java Anonymous inner class

Java anonymous inner class is an inner class without a name and for which only a single object is created. An anonymous inner class can be useful when making an instance of an object with certain "extras" such as overloading methods of a class or interface, without having to actually subclass a class.

In simple words, a class that has no name is known as an anonymous inner class in Java. It should be used if you have to override a method of class or interface. Java Anonymous inner class can be created in two ways:

1. Class (may be abstract or concrete).
2. Interface

Java anonymous inner class example using class

**TestAnonymousInner.java**

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

**Output:**

```
nice fruits
```

A Simple Swing Application, Applets

## 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

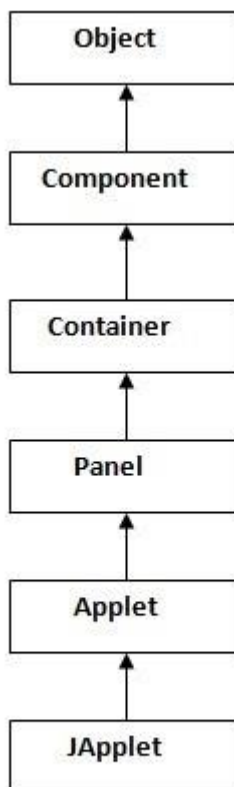
There are many advantages of applet. They are as follows:

- 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



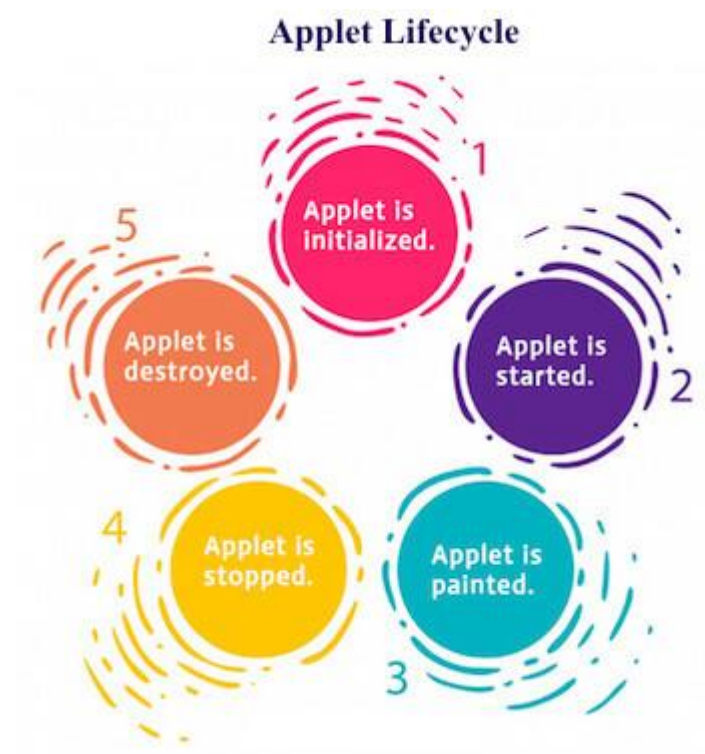
As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the s

Component.

---

### Lifecycle of Java Applet

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



---

### Lifecycle methods for Applet:

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

#### `java.applet.Applet` class

For creating any applet, the `java.applet.Applet` class must be inherited. It provides 4 life cycle methods of an applet.

1. **public void init():** is used to initialize the Applet. It is invoked only once.

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

#### java.awt.Component class

The Component class provides 1 life cycle method of applet.

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

#### How to run an Applet?

There are two ways to run an applet

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

---

#### Simple example of Applet by html file:

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

//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);

}

}

## Java applets and applications

Last Updated: 2021-03-08

An applet is a Java™ program designed to be included in an HTML Web document. You can write your Java applet and include it in an HTML page, much in the same way an image is included. When you use a Java-enabled browser to view an HTML page that contains an applet, the applet's code is transferred to your system and is run by the browser's Java virtual machine.

The HTML document contains tags, which specify the name of the Java applet and its Uniform Resource Locator (URL). The URL is the location at which the applet bytecodes reside on the Internet. When an HTML document containing a Java applet tag is displayed, a Java-enabled Web browser downloads the Java bytecodes from the Internet and uses the Java virtual machine to process the code from within the Web document. These Java applets are what enable Web pages to contain animated graphics or interactive content.

You can also write a Java application that does not require the use of a Web browser.

For more information, see [Writing Applets](#), Sun Microsystems' tutorial for Java applets. It includes an overview of applets, directions for writing applets, and some common applet problems.

**Applications** are stand-alone programs that do not require the use of a browser. Java applications run by starting the Java interpreter from the command line and by specifying the file that contains the compiled application. Applications usually reside on the system on which they are deployed. Applications access resources on the system, and are restricted by the [Java security model](#).

### Parameters passed to an applet

Steps to accomplish this task -:

- *To pass the parameters to the Applet* we need to use the **param** attribute of **<applet>** tag.
- *To retrieve a parameter's value*, we need to use the **getParameter()** method of **Applet** class.

### *Signature of the `getParameter()` method*

```
public String getParameter(String name)
```

- Method takes a String argument *name*, which represents the name of the parameter which was specified with the **param** attribute in the **<applet>** tag.
- Method returns the value of the *name* parameter (if it was defined) else **null** is returned.
- *Passing parameters to an applet.*

- In the upcoming code, we are going to pass a few parameters like Name, Age, Sport, Food, Fruit, Destination to the applet using **param** attribute in <applet>
- Next, we will retrieve the values of these parameters using **getParameter()** method of Applet class.

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

/*
<applet code="Applet8" width="400" height="200">
<param name="Name" value="Roger">
<param name="Age" value="26">
<param name="Sport" value="Tennis">
<param name="Food" value="Pasta">
<param name="Fruit" value="Apple">
<param name="Destination" value="California">
</applet>
*/

public class Applet8 extends Applet
{
    String name;
    String age;
    String sport;
    String food;
    String fruit;
    String destination;

    public void init()
    {
        name = getParameter("Name");
        age = getParameter("Age");
        food = getParameter("Food");
        fruit = getParameter("Fruit");
        destination = getParameter("Destination");
        sport = getParameter("Sport");
    }
}
```

```

public void paint(Graphics g)
{
    g.drawString("Reading parameters passed to this applet -", 20, 20);
    g.drawString("Name -" + name, 20, 40);
    g.drawString("Age -" + age, 20, 60);
    g.drawString("Favorite fruit -" + fruit, 20, 80);
    g.drawString("Favorite food -" + food, 20, 100);
    g.drawString("Favorite destination -" + name, 20, 120);
    g.drawString("Favorite sport -" + sport, 20, 140);
}
}

```

### Output

In order to run our applet using the **appletviewer**, type the following command at the command prompt-

```
appletviewer Applet8.java
```

### 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. 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>
A Paint example
```

```
// Paint lines to a panel.
import java.awt.Graphics;
import java.awt.Insets;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

// This class extends JPanel. It overrides
// the paintComponent() method so that random
// lines are plotted in the panel.
class PaintPanel extends JPanel {
    Insets ins; // holds the panel's insets
    // Construct a panel.
    PaintPanel() { // w w w . d e m o 2 s . c o m
    }
    // Override the paintComponent() method.
    protected void paintComponent(Graphics g) {
        // Always call the superclass method first.
        super.paintComponent(g);

        int x, y, x2, y2;

        // Get the height and width of the component.
        int height = getHeight();
        int width = getWidth();

        // Get the insets.
        ins = getInsets();

        x = width - ins.left;
        y = height - ins.bottom;
        x2 = width - ins.left;
        y2 = height - ins.bottom;
        // Draw the line.
        g.drawLine(x-10, y-10, 100, 100);
    }
}
```

```

// Demonstrate painting directly onto a panel.
public class Main {
    // Create the panel that will be painted.
    PaintPanel pp = new PaintPanel();
    Main() {
        JFrame jfrm = new JFrame("Paint Demo");
        jfrm.setSize(200, 150);
        // Terminate the program when the user closes the application.
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jfrm.add(pp);
        // Display the frame.
        jfrm.setVisible(true);
    }
    public static void main(String args[]) {
        // Create the frame on the event dispatching thread.
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new Main();
            }
        });
    }
}

```

## 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

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

## Java JLabel Example

```

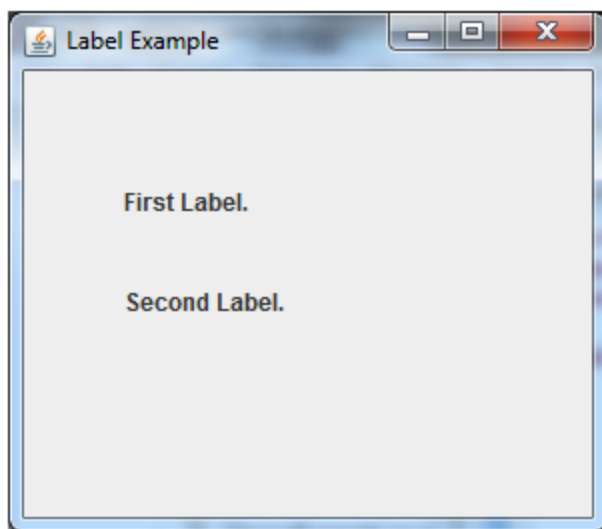
import javax.swing.*;

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.

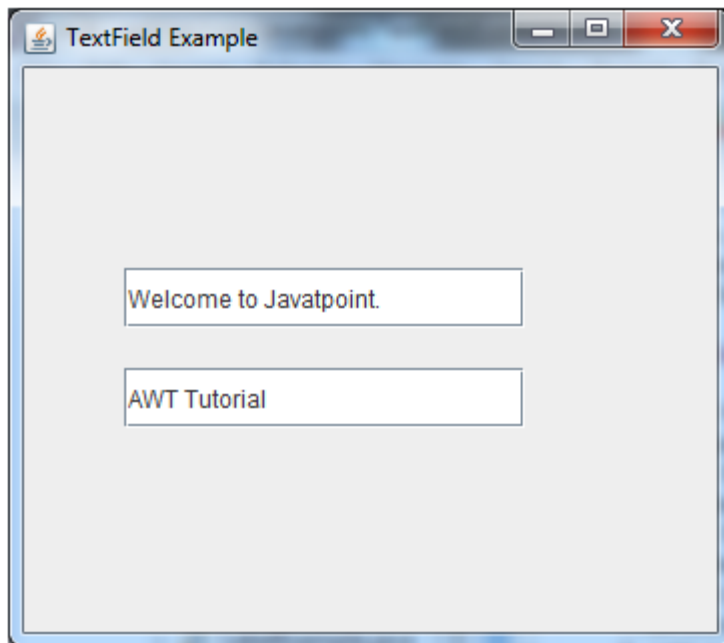
**public class** JTextField **extends** JTextComponent **implements** SwingConstants

### 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 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.

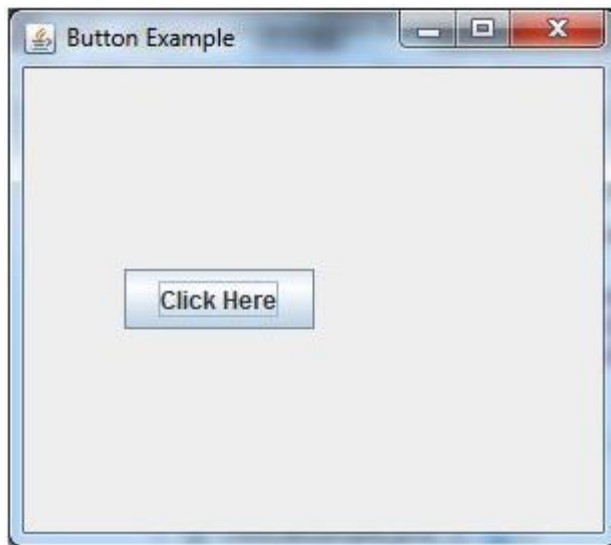
**public class** JButton **extends** AbstractButton **implements** Accessible

## 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:



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

Java JToggleButton

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

JToggleButton Example

```
import java.awt.FlowLayout;
```

```
import java.awt.event.ItemEvent;
```

```
import java.awt.event.ItemListener;
```

```
import javax.swing.JFrame;
```

```
import javax.swing.JToggleButton;
```

```
public class JToggleButtonExample extends JFrame implements ItemListener {
```

```
    public static void main(String[] args) {
```

```
        new JToggleButtonExample();
```

```
    }
```

```
    private JToggleButton button;
```

```
    JToggleButtonExample() {
```

```

setTitle("JToggleButton with ItemListener Example");
setLayout(new FlowLayout());
setJToggleButton();
setAction();
setSize(200, 200);
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

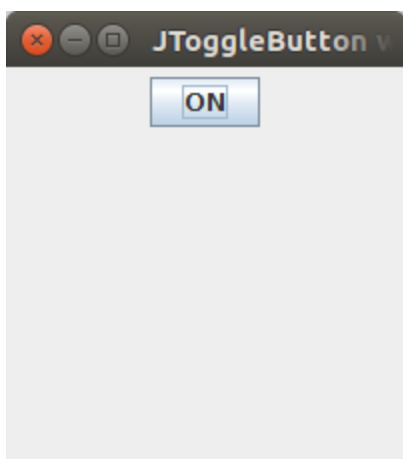
private void setJToggleButton() {
    button = new JToggleButton("ON");
    add(button);
}

private void setAction() {
    button.addItemListener(this);
}

public void itemStateChanged(ItemEvent eve) {
    if (button.isSelected())
        button.setText("OFF");
    else
        button.setText("ON");
}
}

```

Output



## 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.

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

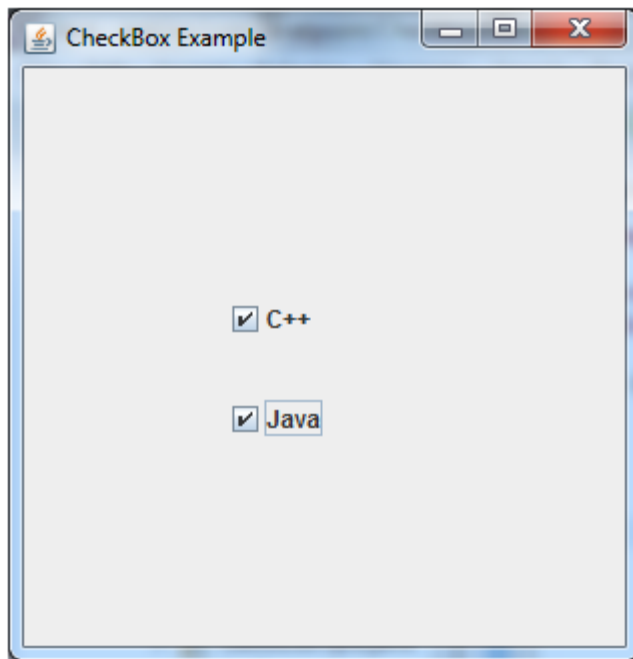
### Java JCheckBox Example

**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:



[Next](#) → ← [Prev](#)

### 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.

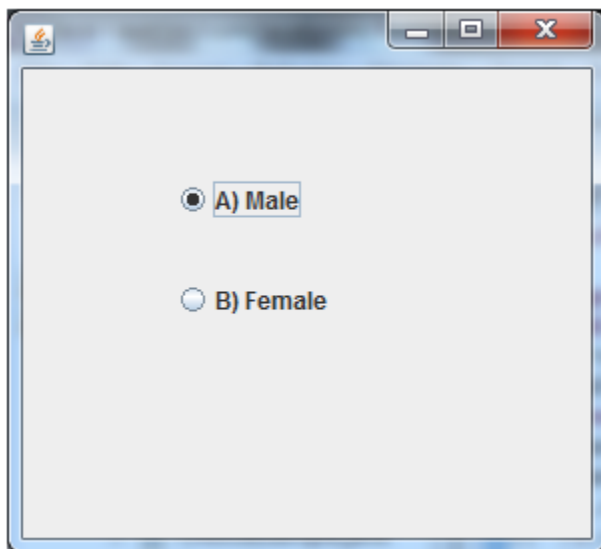
**public class** JRadioButton **extends** JToggleButton **implements** Accessible

### 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);
```

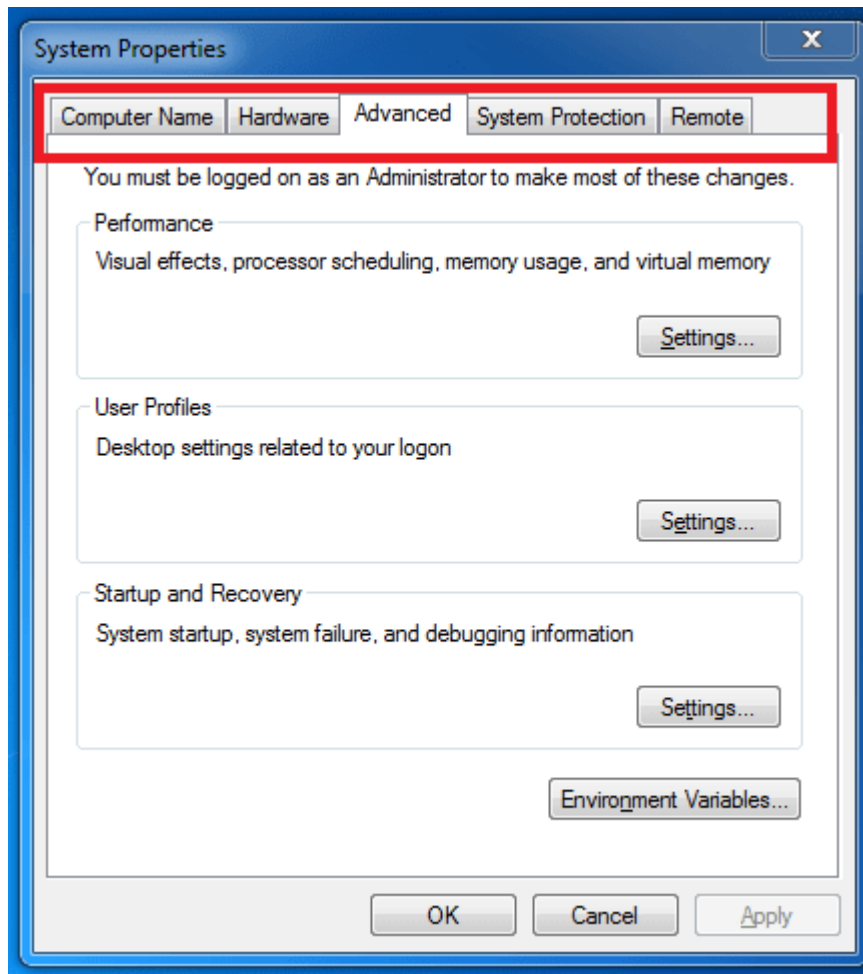
```
ButtonGroup bg=new ButtonGroup();
bg.add(r1);bg.add(r2);
f.add(r1);f.add(r2);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String[] args) {
    new RadioButtonExample();
}
}
```

Output:



### Swing JTabbedPane :

We can see the tabbed pane in windows operating by opening the system properties like below.



## 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.

### 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_SCROLL
LBAR_ALWAYS);
scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR
_ALWAYS);

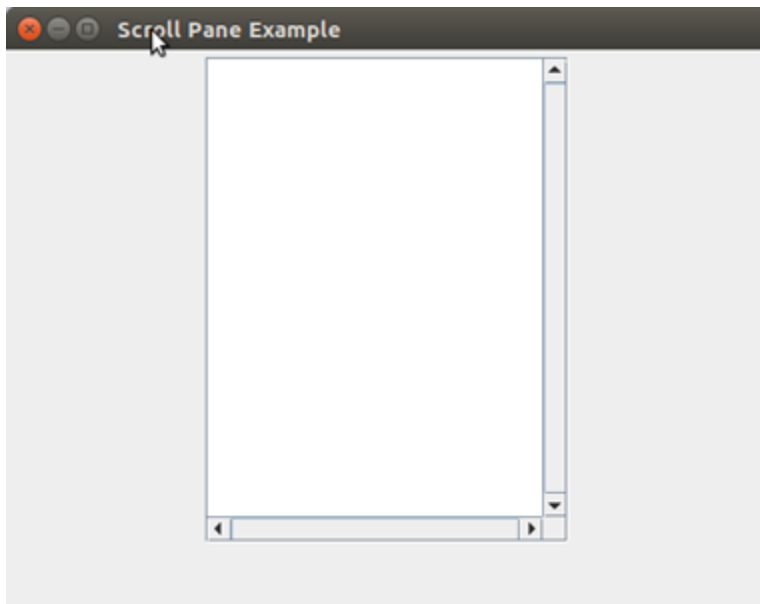
frame.getContentPane().add(scrollableTextArea);
}
public static void main(String[] args) {

    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.

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

### Java JList Example

```
import javax.swing.*;

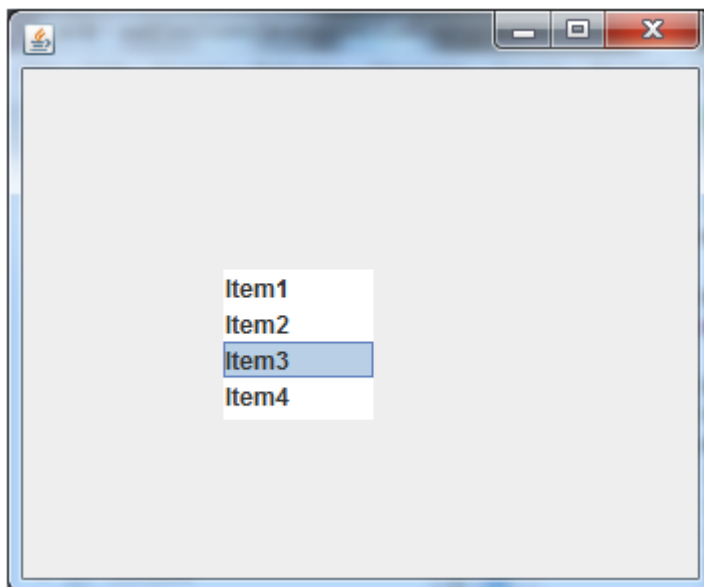
public class ListExample
{
    ListExample(){
        JFrame f= new JFrame();
        DefaultListModel<String> l1 = new DefaultListModel<>();
        l1.addElement("Item1");
        l1.addElement("Item2");
        l1.addElement("Item3");
        l1.addElement("Item4");
        JList<String> list = new 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.

```

public class JComboBox extends JComponent implements ItemSelectable, ListDataListener
, Action

```

### Java JComboBox Example

```

import javax.swing.*.*;

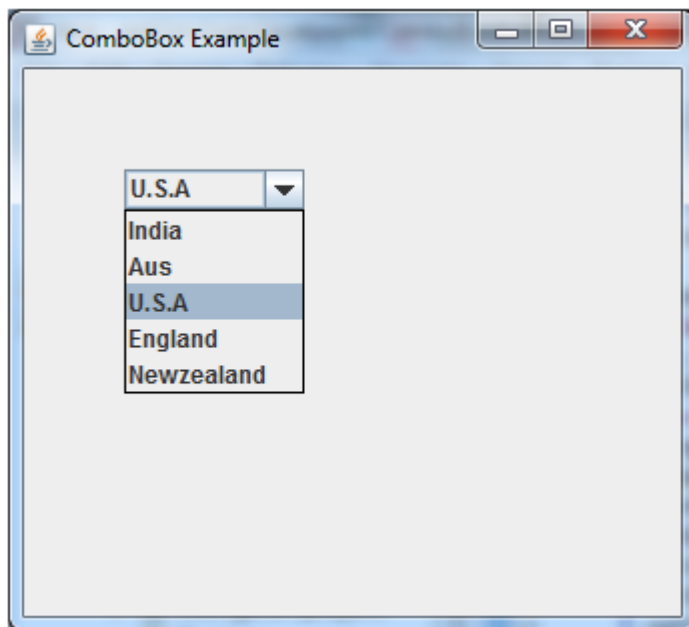
```

```

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:



### **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

```
public class JMenuBar extends JComponent implements MenuElement, Accessible
```

#### JMenu class declaration

```
public class JMenu extends JMenuItem implements MenuElement, Accessible
```

#### JMenuItem class declaration

```
public class JMenuItem extends AbstractButton implements Accessible, MenuElement
```

#### Java JMenuItem and JMenu Example

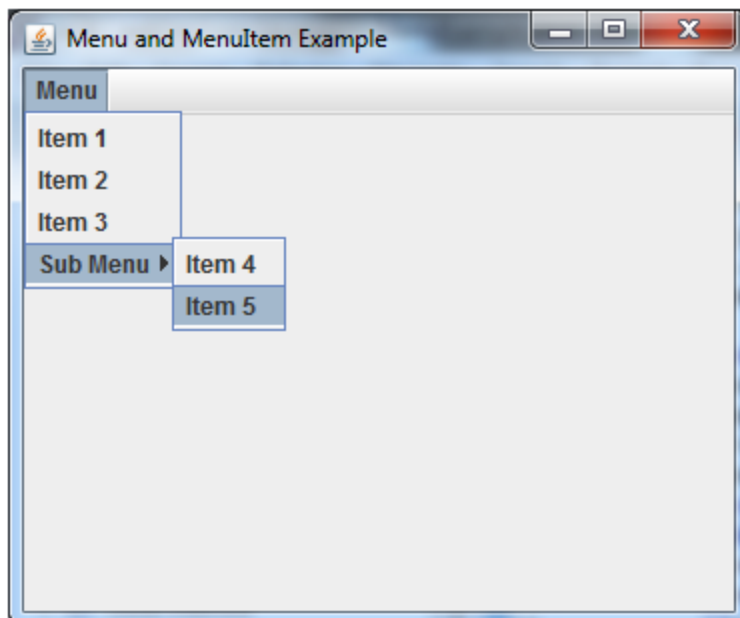
```
import javax.swing.*;
class MenuExample
{
    JMenu menu, submenu;
    JMenuItem i1, i2, i3, i4, i5;
    MenuExample(){
        JFrame f= new JFrame("Menu and MenuItem Example");
        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");
        submenu=new JMenu("Sub Menu");
        i1=new 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

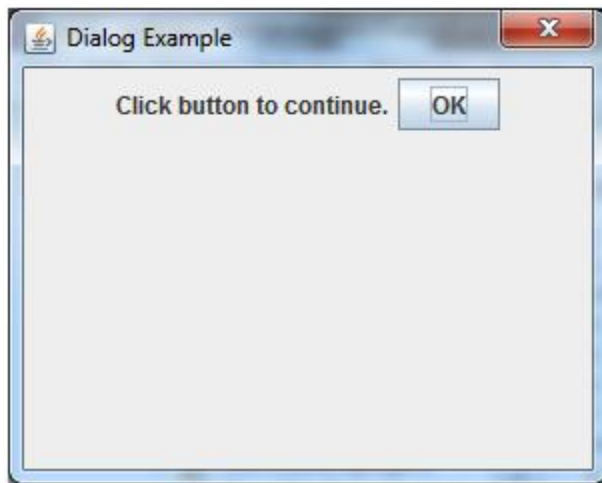
Let's see the declaration for javax.swing.JDialog class.

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

### Java JDialog Example

```
1. import javax.swing.*;
2. import java.awt.*;
3. import java.awt.event.*;
4. public class DialogExample {
5.     private static JDialog d;
6.     DialogExample() {
7.         JFrame f= new JFrame();
8.         d = new JDialog(f , "Dialog Example", true);
9.         d.setLayout( new FlowLayout() );
10.        JButton b = new JButton ("OK");
11.        b.addActionListener ( new ActionListener()
12.        {
13.            public void actionPerformed((ActionEvent e )
14.            {
15.                DialogExample.d.setVisible(false);
16.            }
17.        });
18.        d.add( new JLabel ("Click button to continue."));
19.        d.add(b);
20.        d.setSize(300,300);
21.        d.setVisible(true);
22.    }
23.    public static void main(String args[])
24.    {
25.        new DialogExample();
26.    }
27. }
```

Output:



2.