

# Applet in java

## Introduction:

- An applet in Java is a specialized program designed to run within a web browser, embedded within a webpage.
- Applets are Java programs specifically created to be integrated into web pages, allowing for dynamic content generation within the browser.
- Unlike standalone Java applications, applets operate on the client side, meaning they execute within the user's web browser environment.

## Java Applet Programs Examples With Output

Java applets were small applications written in the [Java programming language](#) that could be downloaded from web servers and run in a web browser.

However, modern browsers have deprecated support for Java applets due to security concerns.

## Example: Simple Java Applet to Display a Message

Here's a basic Java applet code that displays a message "Hello, World!" on the browser window:

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

public class HelloWorldApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello, World!", 50, 50); // Draws the string at x=50, y=50
    }
}
```

## Steps to Run the Java Applet:

Compile the Applet: Save the above code in a file named HelloWorldApplet.java. Open a terminal, navigate to the directory where you saved the file, and compile it using the javac command:

```
javac HelloWorldApplet.java
```

Create HTML File: Create an HTML file (HelloWorldApplet.html) to embed and display the applet in a web browser:

```
<!DOCTYPE html>

<html>

<head>

  <title>HelloWorld Applet</title>

</head>

<body>

  <applet code="HelloWorldApplet.class" width="300" height="200"></applet>

</body>

</html>
```

Run the Applet: Open the HelloWorldApplet.html file with a web browser that supports Java applets (you might need to enable Java in your browser settings). You should see a window displaying the message “Hello, World!”.

**Note:**

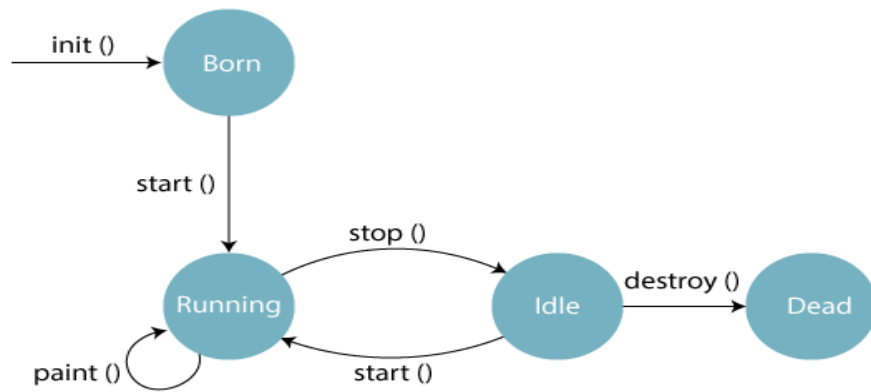
Ensure you have the Java Development Kit (JDK) installed on your machine to compile Java code.

Modern browsers like Chrome, Firefox, and Edge no longer support Java applets. You may need to use an older browser or make specific configurations to run Java applets.

## **Applet Life Cycle in Java**

The life cycle of a Java applet refers to the various stages an applet goes through from its initialization to its termination.

Understanding the applet life cycle is essential for developing applets correctly. Here are the stages involved in the life cycle of a Java applet:



### 1) Initialization:

**Class Loading:** The applet's class is loaded by the Java Virtual Machine (JVM).

**Initialization:** The `init()` method is invoked to initialize the applet. This method is called once in the lifecycle of an applet.

- **Syntax:** `public void init() { // Initialization code here }`

### 2) Starting:

**Start Method:** The `start()` method is called after the `init()` method to indicate that the applet is starting execution.

**Resuming:** If an applet is stopped (using the `stop()` method), it can be restarted using the `start()` method.

```
public void start() { // Start code here }
```

### 3) Running:

**Running State:** The applet remains in this state as long as it is visible on the web page.

**User Interaction:** During this phase, users can interact with the applet, and the applet can process events and update its display as required.

### 4) Stopping:

**Stop Method:** The `stop()` method is called when the applet is no longer visible on the screen or when the web page containing the applet is closed.

**Cleanup:** In this method, you can release resources, stop threads, or perform any cleanup activities necessary when the applet stops.

```
public void stop() { // Stop code here }
```

## 5) Destroying:

**Destroy Method:** The `destroy()` method is invoked when the applet is about to be unloaded from memory.

**Final Cleanup:** You can perform any final cleanup activities, such as releasing resources, closing files, etc., in this method.

```
public void destroy() { // Clean up code here }
```

### Example:

Here's a simple example illustrating the life cycle methods of a Java applet:

In this example, you can observe the sequence of method calls when the applet goes through its life cycle stages.

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

public class AppletLifeCycleDemo extends Applet {

    public void init() {
        System.out.println("Initializing Applet...");
    }

    public void start() {
        System.out.println("Starting Applet...");
    }

    public void stop() {
        System.out.println("Stopping Applet...");
    }

    public void destroy() {
        System.out.println("Destroying Applet...");
    }
}
```

```
public void paint(Graphics g) {  
    g.drawString("Applet Life Cycle Demo", 50, 50);  
}  
}
```

## Types of Applet in Java

- Java applets are specialized programs designed to run within a web browser environment.
- They offer interactive and dynamic content to web pages, making user interactions more engaging.
- Applets are categorized into different types based on their execution context and origin.
- Here are the two primary types of applets:

### 1) Local Applet:

#### Definition:

- A local applet is an applet that is stored and executed from the user's local machine or computer system.
- When a user accesses a web page containing a local applet, the applet is downloaded to the user's computer and then executed within the browser.
- Generally offers faster response times as it operates directly on the user's system resources.

Example: An applet designed to interact with local hardware devices, such as printers or scanners, would typically be a local applet.

### 2) Remote Applet:

#### Definition:

- A remote applet is an applet that is stored on a remote server and is executed on the client-side after being downloaded from that server.
- These applets are accessed through URLs and are not stored locally on the user's machine.

- Since remote applets are not stored locally, they can be accessed from any machine with an internet connection, making them highly portable.

## Example: QuickTime Applet

In the late 1990s and early 2000s, QuickTime applets were used to embed multimedia content within web pages. These applets allowed users to play videos directly in their browsers without needing to download files first. The QuickTime applet was hosted on remote servers, and users accessed it via URLs embedded in HTML pages.

### Simple Applet Program in Java

Creating a simple applet in Java involves defining a class that extends the Applet class provided by the java.applet package. Below is a basic example of how you can create a simple applet program in Java:

```
import java.applet.Applet;

import java.awt.Graphics;

// Defining a simple applet by extending the Applet class

public class SimpleAppletExample extends Applet {

    // Overriding the paint method to draw something on the applet window

    @Override

    public void paint(Graphics g) {

        // Using the Graphics object to draw a string on the applet window

        g.drawString("Welcome to Simple Applet Programming!", 50, 50);

    }

    // This method is invoked when the applet is started

    @Override

    public void start() {

        // Initialization code, if any, can be placed here

    }

}
```

```
// This method is invoked when the applet is stopped

@Override

public void stop() {

    // Cleanup code, if any, can be placed here

}

}
```

Steps to Compile and Run the Applet:

Create a Java File: **SimpleAppletExample.java**.

Compile the Java File: **javac SimpleAppletExample.java**

Run the Applet: After successfully compiling, run the applet using the appletviewer tool (available in JDK) as follows:

**appletviewer SimpleAppletExample.java**

When you run the applet using the appletviewer tool, a window should appear displaying the text “Welcome to Simple Applet Programming!” as specified in the paint() method of the applet class.

### Displaying Graphics in Applet:

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

The Commonly used methods of Graphics class:

- **drawString(String str, int x, int y):** is used to draw the specified string.
- **drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
- **fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
- **drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
- **fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and

specified width and height.

- **drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
- **setColor(Color c):** is used to set the graphics current color to the specified color.
- **setFont(Font font):** is used to set the graphics current font to the specified font.

### **Example: GraphicsDemo.java**

```
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);
    }
}
```

\*\*\*\*\*

## Java AWT Tutorial

**Java AWT** (Abstract Window Toolkit) is *an API to develop Graphical User Interface (GUI) or windows-based applications* in Java.

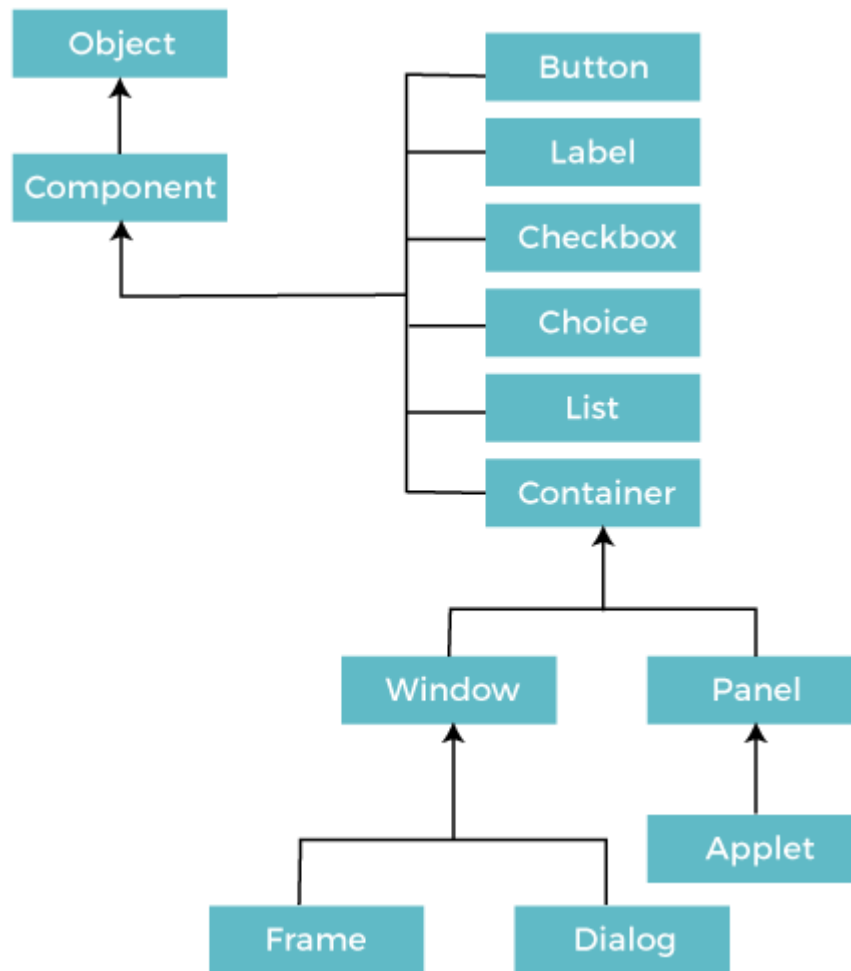
Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

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.





## Components

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

## 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**.

- It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

### Types of containers:

There are four types of containers in Java AWT:

1. Window
2. Panel
3. Frame
4. Dialog

### 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.
- We need to create an instance of Window class to create this container.

### Panel

- The Panel is the container that doesn't contain title bar, border or menu bar.
- It is generic container for holding the components.
- It can have other components like button, text field etc.
- An instance of Panel class creates a container, in which we can add components.

### Frame

- The Frame is the container that contain title bar and border and can have menu bars.
- It can have other components like button, text field, scrollbar etc.
- Frame is most widely used container while developing an AWT application.

## Useful Methods of Component Class

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

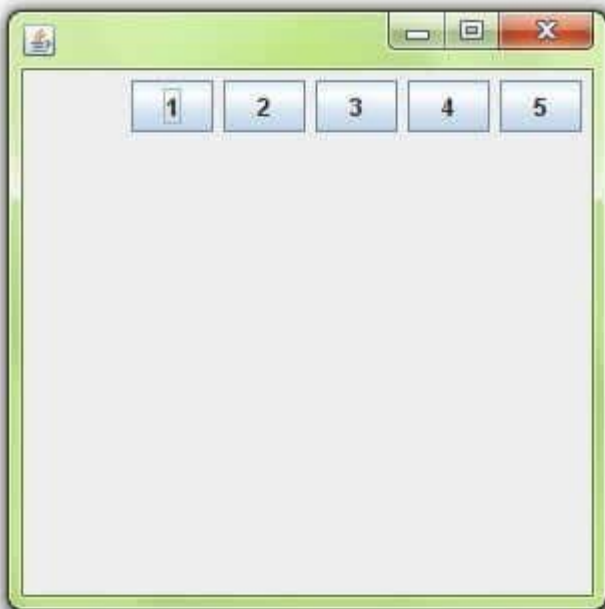
- **FlowLayout:**
  - It arranges the components in a container like the words on a page.
  - It fills the top line from **left to right and top to bottom**.
  - The components are arranged in the order as they are added i.e. first components appears at top left, if the container is not wide enough to display all the components, it is wrapped around the line. Vertical and horizontal gap between components can be controlled.
  - The components can be **left, center or right aligned**.

## 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.
4. **import** java.awt.\*;
5. **import** javax.swing.\*;
- 6.
7. **public class** MyFlowLayout{
8. JFrame f;
9. MyFlowLayout(){
10. f=**new** JFrame();

```
11.
12. JButton b1=new JButton("1");
13. JButton b2=new JButton("2");
14. JButton b3=new JButton("3");
15. JButton b4=new JButton("4");
16. JButton b5=new JButton("5");
17.
18. // adding buttons to the frame
19. f.add(b1); f.add(b2); f.add(b3); f.add(b4); f.add(b5);
20.
21. // setting flow layout of right alignment
22. f.setLayout(new FlowLayout(FlowLayout.RIGHT));
23.
24. f.setSize(300,300);
25. f.setVisible(true);
26. }
27. public static void main(String[] args) {
28.     new MyFlowLayout();
29. }
30. }
```

**Output:**



- **BorderLayout:**
  - It arranges all the components along the edges or the middle of the container i.e. **top, bottom, right and left** edges of the area.
  - The components added to the top or bottom gets its preferred height, but its width will be the width of the container and also the components added to the left or right gets its preferred width, but its height will be the remaining height of the container.
  - The components added to the center gets neither its preferred height or width. It covers the remaining area of the container.

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

1. **import** java.awt.\*;
2. **import** javax.swing.\*;
- 3.
4. **public class** Border
5. {
6. JFrame f;
7. Border()
8. {
9.   f = **new** JFrame();
- 10.
11.   *// creating buttons*
12.   JButton b1 = **new** JButton("NORTH"); *// the button will be labeled as NORTH*

```
13. JButton b2 = new JButton("SOUTH"); // the button will be labeled as SOUTH
14. JButton b3 = new JButton("EAST"); // the button will be labeled as EAST
15. JButton b4 = new JButton("WEST"); // the button will be labeled as WEST
16. JButton b5 = new JButton("CENTER"); // the button will be labeled as CENTER
17.
18. f.add(b1, BorderLayout.NORTH); // b1 will be placed in the North Direction
19. f.add(b2, BorderLayout.SOUTH); // b2 will be placed in the South Direction
20. f.add(b3, BorderLayout.EAST); // b2 will be placed in the East Direction
21. f.add(b4, BorderLayout.WEST); // b2 will be placed in the West Direction
22. f.add(b5, BorderLayout.CENTER); // b2 will be placed in the Center
23.
24. f.setSize(300, 300);
25. f.setVisible(true);
26. }
27. public static void main(String[] args) {
28.     new Border();
29. }
30. }
```

**Output:**



•

- **GridLayout:**
- It arranges all the components in a grid of **equally sized cells**, adding them from the **left to right** and **top to bottom**.
- Only one component can be placed in a cell and each region of the grid will have the same size.
- When the container is resized, all cells are automatically resized. The order of placing the components in a cell is determined as they were added.

Example of GridLayout class: Using GridLayout(int rows, int columns) Constructor

FileName: MyGridLayout.java

```

1. import java.awt.*;
2. import javax.swing.*;
3. public class MyGridLayout{
4.     JFrame f;
5.     MyGridLayout(){
6.         f=new JFrame();
7.         JButton b1=new JButton("1");
8.         JButton b2=new JButton("2");
9.         JButton b3=new JButton("3");
10.        JButton b4=new JButton("4");
11.        JButton b5=new JButton("5");
12.        JButton b6=new JButton("6");
13.        JButton b7=new JButton("7");
14.        JButton b8=new JButton("8");
15.        JButton b9=new JButton("9");
16.        // adding buttons to the frame
17.        f.add(b1); f.add(b2); f.add(b3);
18.        f.add(b4); f.add(b5); f.add(b6);
19.        f.add(b7); f.add(b8); f.add(b9);
20.
21.        // setting grid layout of 3 rows and 3 columns
22.        f.setLayout(new GridLayout(3,3));
23.        f.setSize(300,300);
24.        f.setVisible(true);
25.    }
26.    public static void main(String[] args) {
27.        new MyGridLayout();
28.    }

```

29. }

**Output:**



- **BoxLayout:**
  1. It arranges multiple components in either **vertically or horizontally**, but not both.
  2. The components are arranged from **left to right or top to bottom**.
  3. If the components are aligned **horizontally**, the height of all components will be the same and equal to the largest sized components.
  4. If the components are aligned **vertically**, the width of all components will be the same and equal to the largest width components.

Fields of BoxLayout Class

1. **public static final int X\_AXIS:** Alignment of the components are horizontal from left to right.
2. **public static final int Y\_AXIS:** Alignment of the components are vertical from top to bottom.

**Constructor of BoxLayout class**

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

Example of BoxLayout class with Y-AXIS:

**FileName:** BoxLayoutExample1.java



```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class BorderLayoutExample1 extends Frame {
5.     Button buttons[];
6.
7.     public BorderLayoutExample1 () {
8.         buttons = new Button [5];
9.
10.        for (int i = 0;i<5;i++) {
11.            buttons[i] = new Button ("Button " + (i + 1));
12.            // adding the buttons so that it can be displayed
13.            add (buttons[i]);
14.        }
15.        // the buttons will be placed horizontally
16.        setLayout (new BorderLayout (this, BorderLayout.Y_AXIS));
17.        setSize(400,400);
18.        setVisible(true);
19.    }
20.    // main method
21.    public static void main(String args[]){
22.        BorderLayoutExample1 b=new BorderLayoutExample1();
23.    }
24. }
```

**Output:**



- **CardLayout:**
  1. It arranges two or more components having the same size.
  2. The components are **arranged in a deck**, where all the cards of the same size and the **only top card are visible at any time**.
  3. The first component added in the container will be kept at the top of the deck.
  4. The default gap at the left, right, top and bottom edges are zero and the card components are displayed either **horizontally or vertically**.

### 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

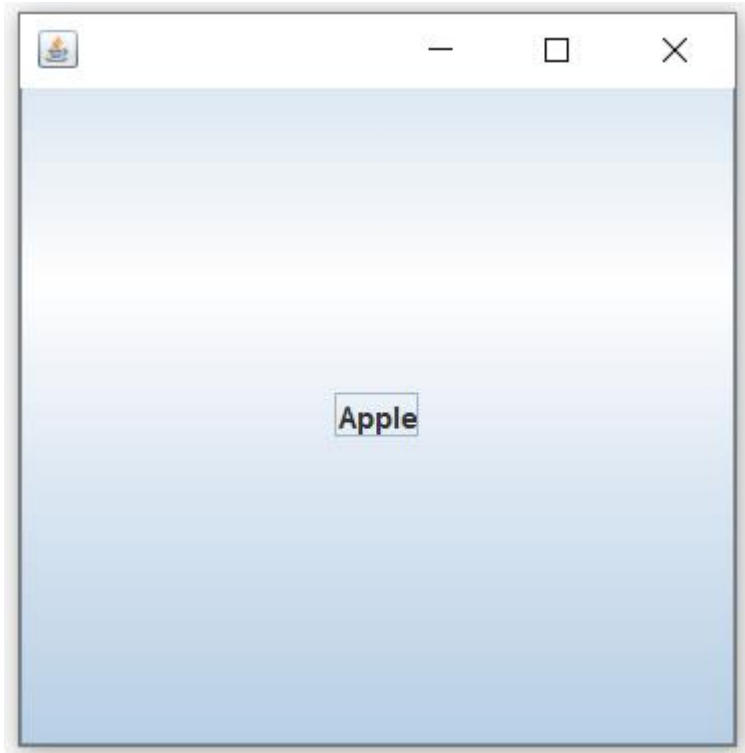
The following program uses the next() method to move to the next card of the container.

**FileName:** CardLayoutExample1.java

```
1. // import statements
2. import java.awt.*;
3. import javax.swing.*;
4. import java.awt.event.*;
5.
6. public class CardLayoutExample1 extends JFrame implements ActionListener
7. {
8.
9.     CardLayout crd;
10.
11. // button variables to hold the references of buttons
12. JButton btn1, btn2, btn3;
13. Container cPane;
14.
15. // constructor of the class
16. CardLayoutExample1()
17. {
18.
19.     cPane = getContentPane();
20.
21. //default constructor used
22. // therefore, components will
23. // cover the whole area
24. crd = new CardLayout();
25.
26. cPane.setLayout(crd);
27.
28. // creating the buttons
29. btn1 = new JButton("Apple");
30. btn2 = new JButton("Boy");
31. btn3 = new JButton("Cat");
```

```
32.
33. // adding listeners to it
34. btn1.addActionListener(this);
35. btn2.addActionListener(this);
36. btn3.addActionListener(this);
37.
38. cPane.add("a", btn1); // first card is the button btn1
39. cPane.add("b", btn2); // first card is the button btn2
40. cPane.add("c", btn3); // first card is the button btn3
41.
42. }
43. public void actionPerformed(ActionEvent e)
44. {
45. // Upon clicking the button, the next card of the container is shown
46. // after the last card, again, the first card of the container is shown upon clicking
47. crd.next(cPane);
48. }
49.
50. // main method
51. public static void main(String args[])
52. {
53. // creating an object of the class CardLayoutExample1
54. CardLayoutExample1 crdl = new CardLayoutExample1();
55.
56. // size is 300 * 300
57. crdl.setSize(300, 300);
58. crdl.setVisible(true);
59. crdl.setDefaultCloseOperation(EXIT_ON_CLOSE);
60. }
61. }
```

**Output:**



### **GridBagLayout:**

1. It is a powerful layout which arranges all the components in a grid of cells and maintains the aspect ratio of the object whenever the container is resized.
2. In this layout, cells may be different in size.
3. It assigns a consistent horizontal and vertical gap among components.
4. It allows us to specify a default alignment for components within the columns or rows.

### **AWT Button**

In Java, AWT contains a Button Class. It is used for creating a labelled button which can perform an action.

### **AWT Button Class Declaration:**

```
public class Button extends Component implements Accessible
```

Example:

Lets take an example to create a button and it to the frame by providing coordinates.

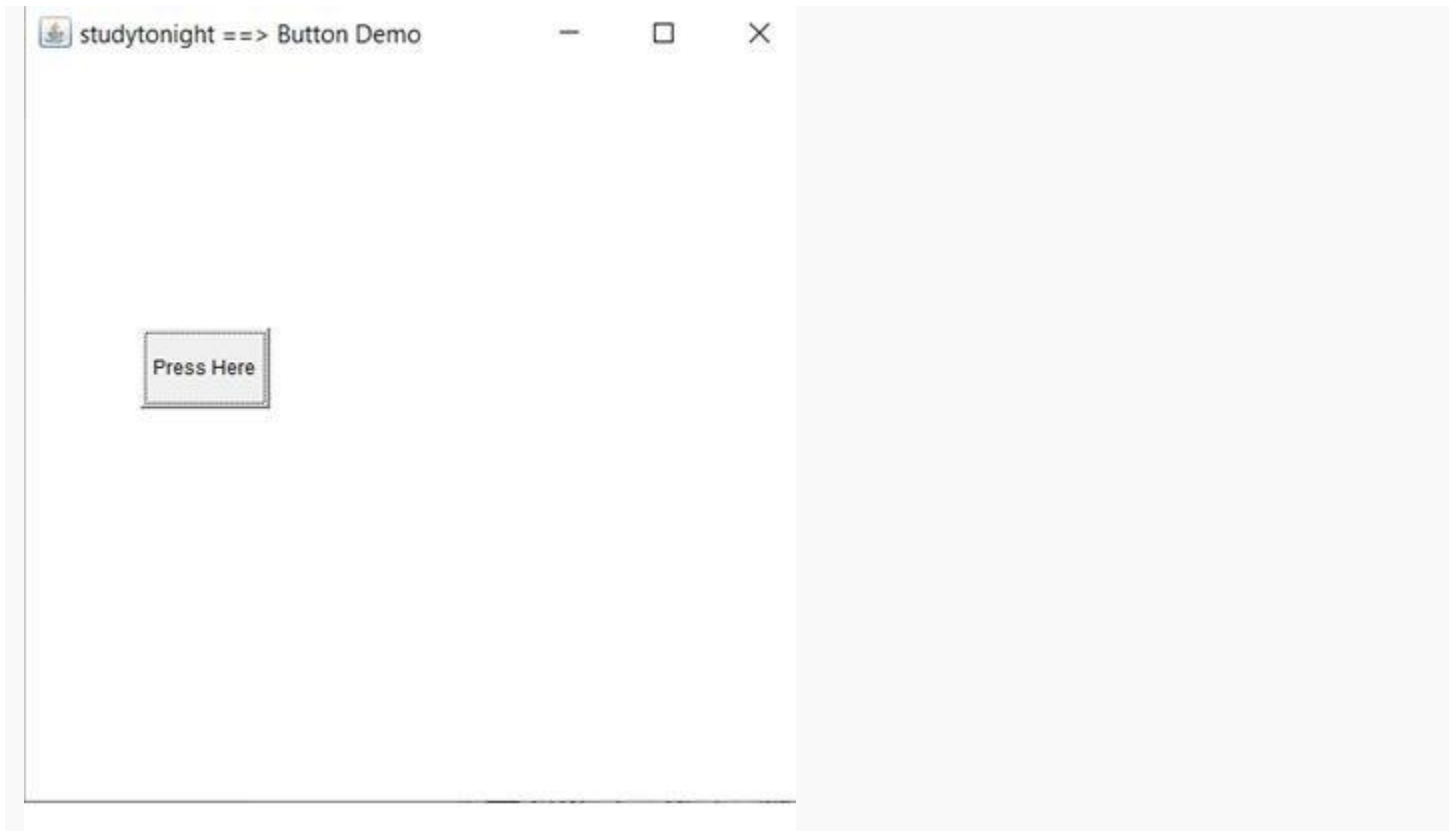
```
import java.awt.*;  
public class ButtonDemo1
```

```
{  
  
public static void main(String[] args)  
  
{  
  
    Frame f1=new Frame("studytonight ==> Button Demo");  
  
    Button b1=new Button("Press Here");  
  
    b1.setBounds(80,200,80,50);  
  
    f1.add(b1);  
  
    f1.setSize(500,500);  
  
    f1.setLayout(null);  
  
    f1.setVisible(true);  
  
}  
  
}
```

Copy



```
C:\Windows\System32\cmd.exe - java ButtonDe...  
D:\Studytonight>javac ButtonDemo1.java  
D:\Studytonight>java ButtonDemo1  
_
```



---

## AWT Label

In Java, AWT contains a Label Class. It is used for placing text in a container. Only Single line text is allowed and the text can not be changed directly.

### Label Declaration:

```
public class Label extends Component implements Accessible
```

Example:

In this example, we are creating two labels to display text to the frame.

```
import java.awt.*;  
  
class LabelDemo1  
{
```

```
public static void main(String args[])
{
    Frame l_Frame= new Frame("studytonight ==> Label Demo");

    Label lab1,lab2;

    lab1=new Label("Welcome to studytonight.com");

    lab1.setBounds(50,50,200,30);

    lab2=new Label("This Tutorial is of Java");

    lab2.setBounds(50,100,200,30);

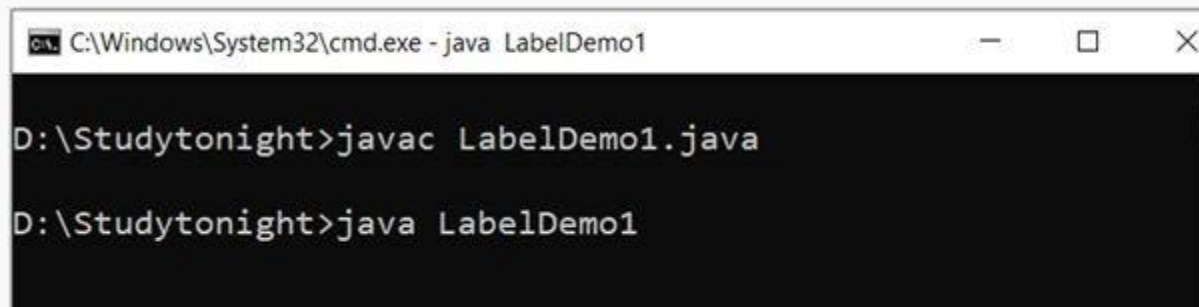
    l_Frame.add(lab1);

    l_Frame.add(lab2);

    l_Frame.setSize(500,500);

    l_Frame.setLayout(null);

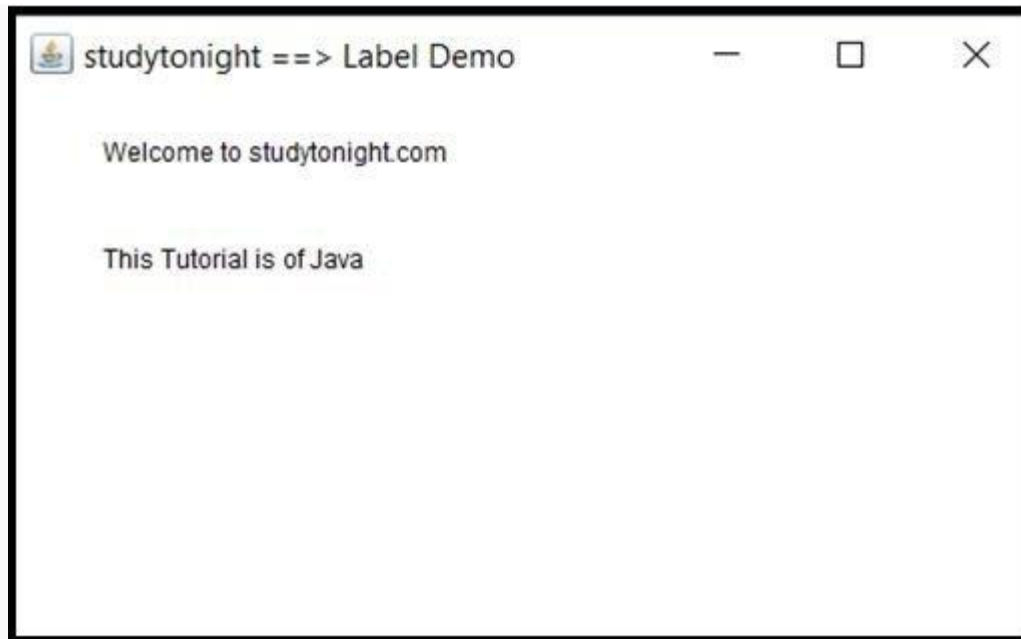
    l_Frame.setVisible(true);
}
}
```



The screenshot shows a Windows command prompt window with the title "C:\Windows\System32\cmd.exe - java LabelDemo1". The window contains the following text:

```
D:\Studytonight>javac LabelDemo1.java
D:\Studytonight>java LabelDemo1
```





## AWT TextField

In Java, AWT contains a `TextField` Class. It is used for displaying single line text.

### TextField Declaration:

```
public class TextField extends TextComponent
```

Example:

We are creating two textfields to display single line text string. This text is editable in nature, see the below example.

```
import java.awt.*;

class TextFieldDemo1 {

public static void main(String args[]){

    Frame TextF_f= new Frame("studytonight ==>TextField");

    TextField text1,text2;

    text1=new TextField("Welcome to studytonight");

    text1.setBounds(60,100, 230,40);
```

```
text2=new TextField("This tutorial is of Java");

text2.setBounds(60,150, 230,40);

TextF_f.add(text1);

TextF_f.add(text2);

TextF_f.setSize(500,500);

TextF_f.setLayout(null);

TextF_f.setVisible(true);

}

}
```

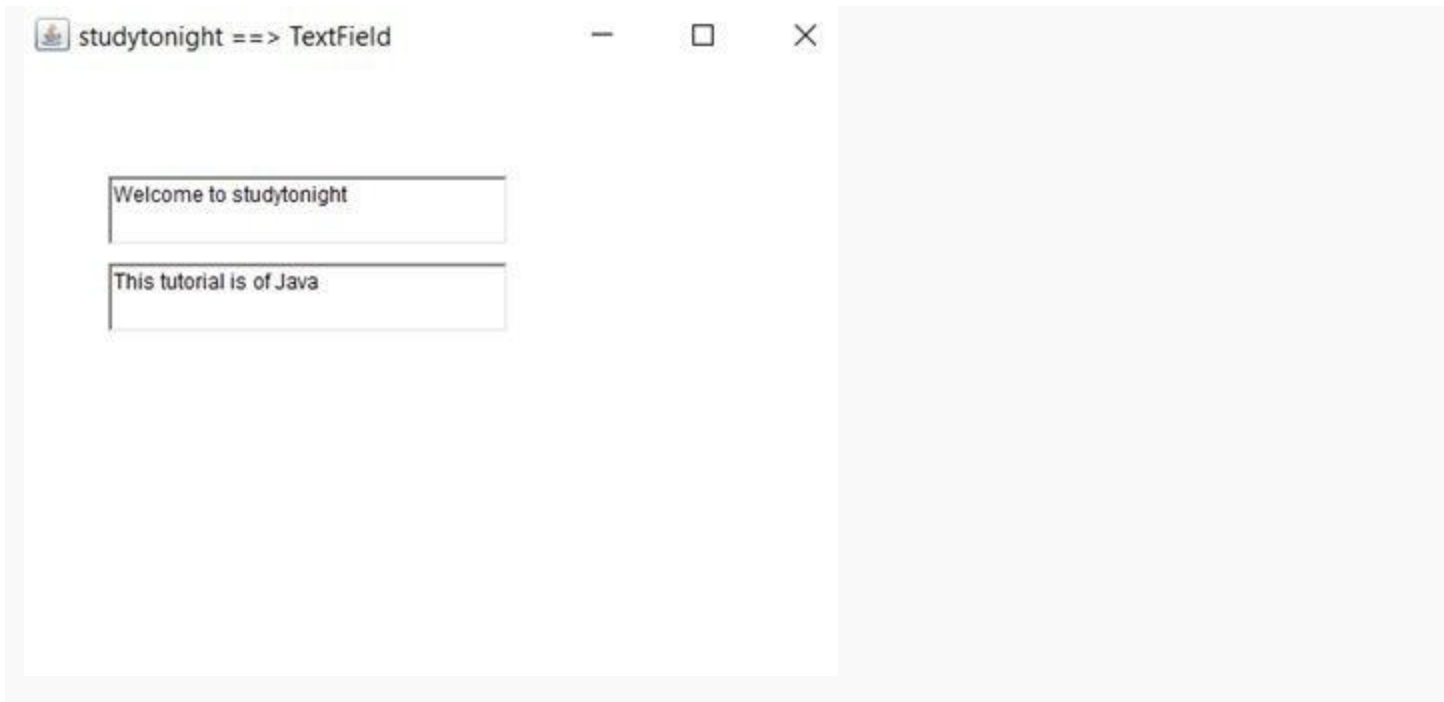
Copy



```
C:\Windows\System32\cmd.exe - java TextFieldDemo1

D:\Studytonight>javac TextFieldDemo1.java

D:\Studytonight>java TextFieldDemo1
```



## AWT TextArea

In Java, AWT contains a `TextArea` Class. It is used for displaying multiple-line text.

### TextArea Declaration:

```
public class TextArea extends TextComponent
```

Example:

In this example, we are creating a `TextArea` that is used to display multiple-line text string and allows text editing as well.

```
import java.awt.*;

public class TextAreaDemo1
{
    TextAreaDemo1()
    {
        Frame textArea_f= new Frame();
```

```
    TextArea area=new TextArea("Welcome to studytonight.com");

    area.setBounds(30,40, 200,200);

    textArea_f.add(area);

    textArea_f.setSize(300,300);

    textArea_f.setLayout(null);

    textArea_f.setVisible(true);

}

public static void main(String args[])

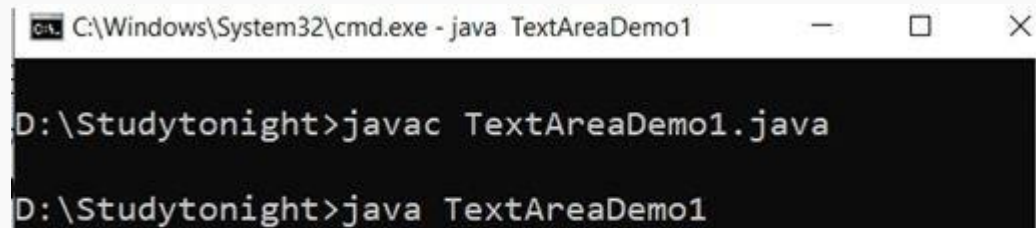
{

    new TextAreaDemo1();

}

}
```

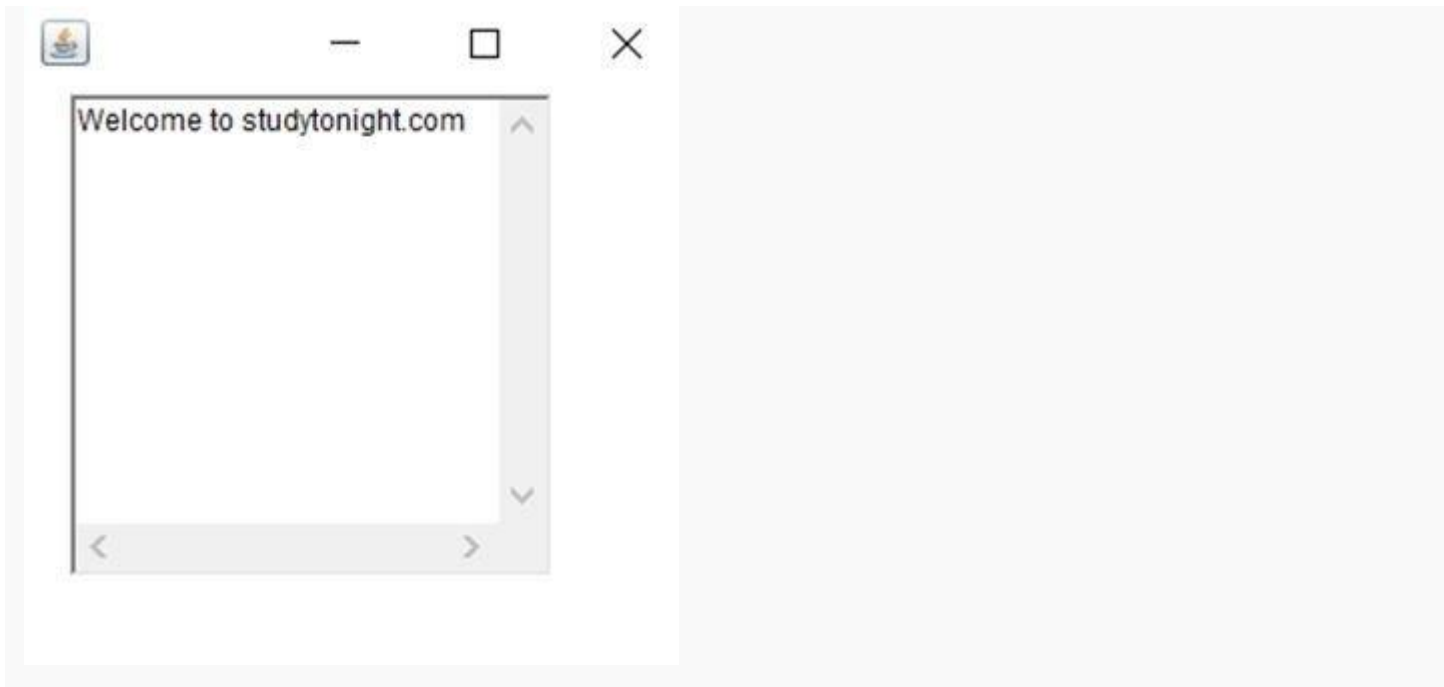
Copy



```
C:\Windows\System32\cmd.exe - java TextAreaDemo1

D:\Studytonight>javac TextAreaDemo1.java

D:\Studytonight>java TextAreaDemo1
```



## AWT Checkbox

In Java, AWT contains a Checkbox Class. It is used when we want to select only one option i.e true or false. When the checkbox is checked then its state is "on" (true) else it is "off"(false).

## Checkbox Syntax

```
public class Checkbox extends Component implements ItemSelectable, Accessible
```

Example:

In this example, we are creating checkbox that are used to get user input. If checkbox is checked it returns true else returns false.

```
import java.awt.*;

public class CheckboxDemo1
{
    CheckboxDemo1(){
        Frame checkB_f= new Frame("studytonight ==>Checkbox Example");

        Checkbox ckbox1 = new Checkbox("Yes", true);
```

```
ckbox1.setBounds(100,100, 60,60);

Checkbox ckbox2 = new Checkbox("No");

ckbox2.setBounds(100,150, 60,60);

checkB_f.add(ckbox1);

checkB_f.add(ckbox2);

checkB_f.setSize(400,400);

checkB_f.setLayout(null);

checkB_f.setVisible(true);

}

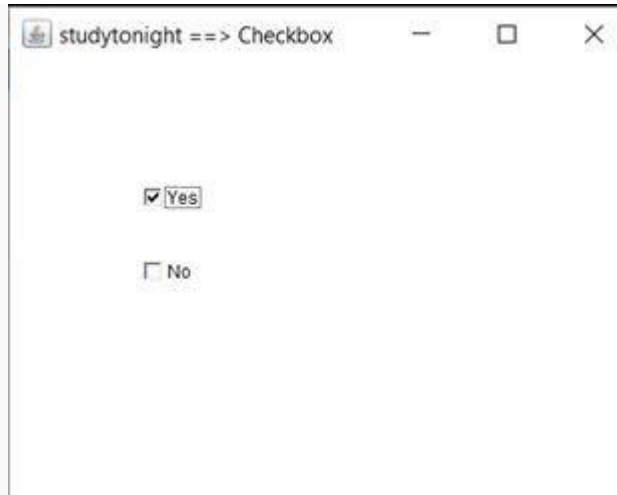
public static void main(String args[])

{

    new CheckboxDemo1();

}

}
```



## AWT CheckboxGroup

In Java, AWT contains a `CheckboxGroup` Class. It is used to group a set of `Checkbox`. When `Checkboxes` are grouped then only one box can be checked at a time.

### CheckboxGroup Declaration:

```
public class CheckboxGroup extends Object implements Serializable
```

Example:

This example creates a `checkboxgroup` that is used to group multiple `checkbox` in a single unit. It is helpful when we have to select single choice among the multiples.

```
import java.awt.*;

public class CheckboxGroupDemo
{
    CheckboxGroupDemo(){
        Frame ck_groupf= new Frame("studytonight ==>CheckboxGroup");

        CheckboxGroupobj = new CheckboxGroup();

        Checkbox ckBox1 = new Checkbox("Yes", obj, true);

        ckBox1.setBounds(100,100, 50,50);
    }
}
```

```
Checkbox ckBox2 = new Checkbox("No", obj, false);

ckBox2.setBounds(100,150, 50,50);

ck_groupf.add(ckBox1);

ck_groupf.add(ckBox2);

ck_groupf.setSize(400,400);

ck_groupf.setLayout(null);

ck_groupf.setVisible(true);

}

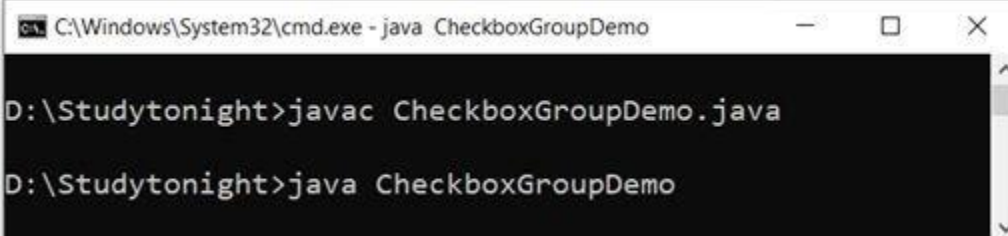
public static void main(String args[])

{

    new CheckboxGroupDemo();

}

}
```

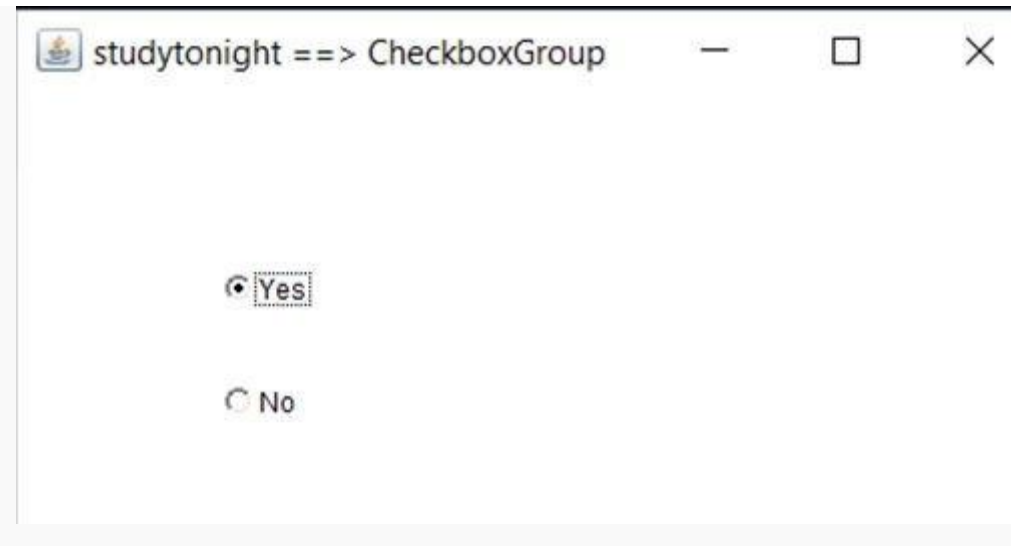


```
C:\Windows\System32\cmd.exe - java CheckboxGroupDemo

D:\Studytonight>javac CheckboxGroupDemo.java

D:\Studytonight>java CheckboxGroupDemo
```





### AWT Choice

In Java, AWT contains a Choice Class. It is used for creating a drop-down menu of choices. When a user selects a particular item from the drop-down then it is shown on the top of the menu.

#### Choice Declaration:

```
public class Choice extends Component implements ItemSelectable, Accessible
```

Example:

In this example, we are creating drop-down menu that is used to get user choice from multiple choices.

```
import java.awt.*;

public class ChoiceDemo
{
    ChoiceDemo()
    {
        Frame choice_f= new Frame();

        Choice obj=new Choice();

        obj.setBounds(80,80, 100,100);
```

```
obj.add("Red");

obj.add("Blue");

obj.add("Black");

obj.add("Pink");

obj.add("White");

obj.add("Green");

choice_f.add(obj);

choice_f.setSize(400,400);

choice_f.setLayout(null);

choice_f.setVisible(true);

}

public static void main(String args[])

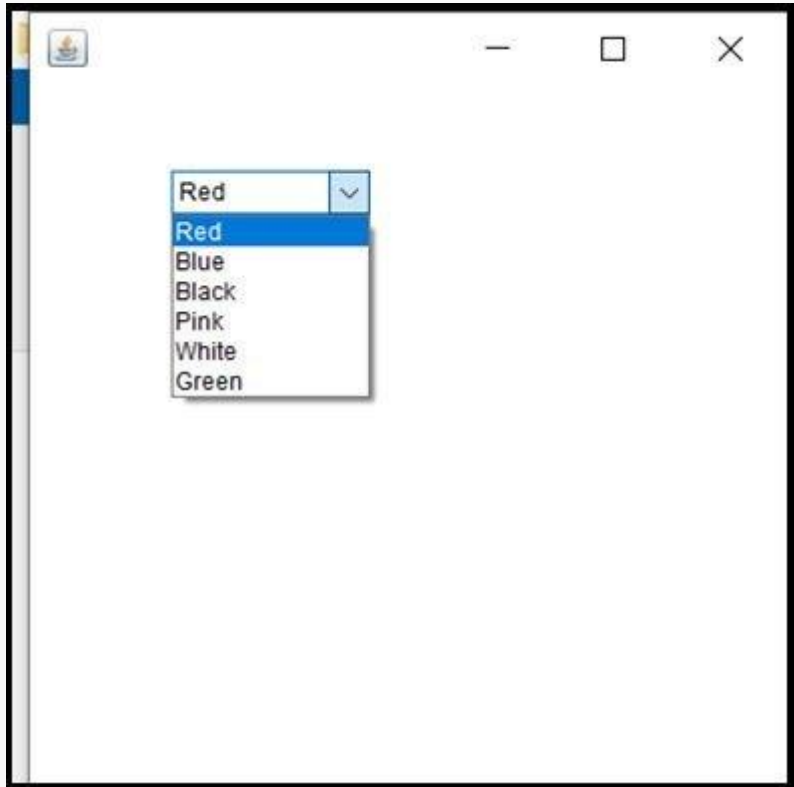
{

    new ChoiceDemo();

}

}
```

```
C:\Windows\System32\cmd.exe - java ChoiceDemo
D:\Studytonight>javac ChoiceDemo.java
D:\Studytonight>java ChoiceDemo
```



### AWT List

In Java, AWT contains a List Class. It is used to represent a list of items together. One or more than one item can be selected from the list.

#### List Declaration:

```
public class List extends Component implements ItemSelectable, Accessible
```

Example:

In this example, we are creating a list that is used to list out the items.



```
import java.awt.*;

public class ListDemo
{
    ListDemo()
    {
        Frame list_f= new Frame();

        List obj=new List(6);

        obj.setBounds(80,80, 100,100);

        obj.add("Red");

        obj.add("Blue");

        obj.add("Black");

        obj.add("Pink");

        obj.add("White");

        obj.add("Green");

        list_f.add(obj);

        list_f.setSize(400,400);

        list_f.setLayout(null);

        list_f.setVisible(true);

    }

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

```
new ListDemo();
```

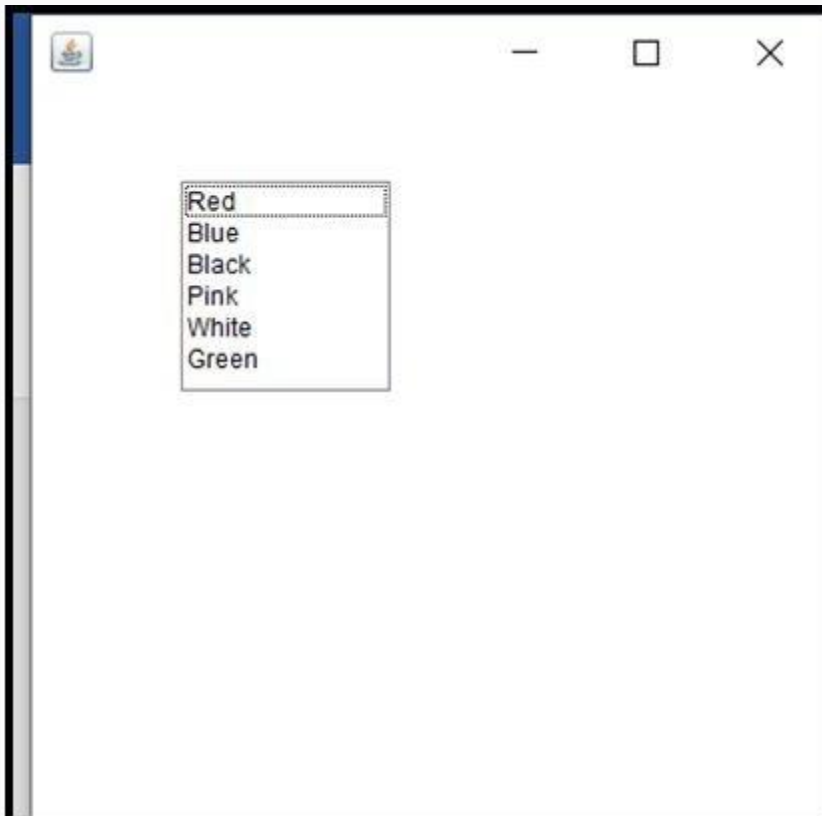
```
}
```

```
}
```

Copy



```
C:\Windows\System32\cmd.exe - java ListDemo  
D:\Studytonight>javac ListDemo.java  
D:\Studytonight>java ListDemo
```



## Delegation Event Model in Java

The Event model is based on two things, i.e., Event Source and Event Listeners.

- Event Source means any object which creates the message or event.
- Event Listeners are the object which receives the message or event.

Now, coming to the Delegation Event Model in Java, the Delegation event model is based upon Event Source, Event Listeners, and Event Objects.

- Event Source is the class used to broadcast the events.
- Event Listeners are the classes that receive notifications of events.
- Event Object is the class object which describes the event.

## Java GUI Event Handling

Any program that uses GUI (graphical user interface) such as Java application written for windows, is event driven. Event describes the change in state of any object. **For Example :** Pressing a button, Entering a character in Textbox, Clicking or Dragging a mouse, etc.

### Components of Event Handling

Event handling has three main components,

- **Events :** An event is a change in state of an object.
- **Events Source :** Event source is an object that generates an event.
- **Listeners :** A listener is an object that listens to the event. A listener gets notified when an event occurs.

How Events are handled?

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

### Important Event Classes and Interface

Event Classes	Description	Listener Interface
---------------	-------------	--------------------

<b>ActionEvent</b>	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
<b>MouseEvent</b>	generated when mouse is dragged, moved,clicked,pressed or released and also when it enters or exit a component	MouseListener
<b>KeyEvent</b>	generated when input is received from keyboard	KeyListener
<b>ItemEvent</b>	generated when check-box or list item is clicked	ItemListener
<b>TextEvent</b>	generated when value of textarea or textfield is changed	TextListener
<b>MouseEvent</b>	generated when mouse wheel is moved	MouseWheelListener
<b>WindowEvent</b>	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
<b>ComponentEvent</b>	generated when component is hidden, moved, resized or set visible	ComponentEventListener
<b>ContainerEvent</b>	generated when component is added or removed from container	ContainerListener
<b>AdjustmentEvent</b>	generated when scroll bar is manipulated	AdjustmentListener

<b>FocusEvent</b>	generated when component gains or loses keyboard focus	FocusListener
-------------------	--	---------------

---

### Steps to handle events:

1. Implement appropriate interface in the class.
  2. Register the component with the listener.
- 

### Example of Event Handling

```
import java.awt.*;

import java.awt.event.*;

import java.applet.*;

import java.applet.*;

import java.awt.event.*;

import java.awt.*;

public class Test extends Applet implements KeyListener

{

    String msg="";

    public void init()

    {
```



```
        addKeyListener(this);

    }

    public void keyPressed(KeyEvent k)

    {

        showStatus("KeyPressed");

    }

    public void keyReleased(KeyEvent k)

    {

        showStatus("KeyReleased");

    }

    public void keyTyped(KeyEvent k)

    {

        msg = msg+k.getKeyChar();

        repaint();

    }

    public void paint(Graphics g)

    {

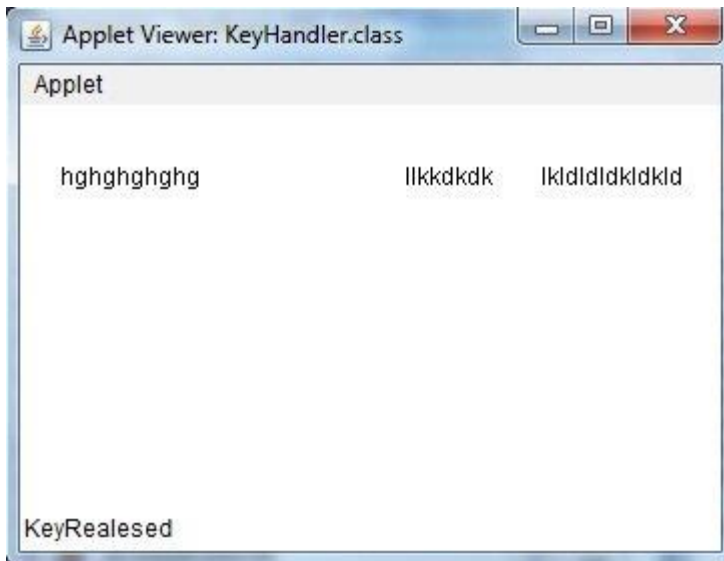
        g.drawString(msg, 20, 40);

    }

}
```

## HTML code:

```
<applet code="Test" width=300, height=100>
</applet>
```



---

## Java Adapter Classes with Example

- The listener class that implements the Listener interface must provide bodies for all of the methods of that interface.
- It is not a problem for all the semantic listener interfaces such as ActionEvent, ItemEvent, TextEvent, AdapterEvent as each of them declares only one method.
- However, for all the low-level listener interfaces where each interface contains multiple methods, implementing each method can be somewhat tedious, especially when we have to define methods in which we are not interested.
- For example: Suppose we are interested in setting up only one listener interface method windowClosing() of the WindowListener interface that causes the program to terminate.
- In that case, we would not only need to provide code for windowClosing() method but also need to write empty bodies for the other methods available in the WindowListener interface.

```
class WindowEventFrame extends Frame implements WindowListener {
    .....
    public void windowClosing(WindowEvent e){
        System.exit(0);
    }
}
```

```

public void windowOpened(WindowEvent e){}
public void windowClosed(WindowEvent e){}
public void windowActivated(WindowEvent e){}
public void windowDeactivated(WindowEvent e){}
public void windowIconified(WindowEvent e){}
public void windowDeiconified(WindowEvent e){}
.....
}

```

- To avoid this unnecessary code, the [java.awt.event](#) package provides adapter classes for various event-listener types.
- The event listener interfaces that contain more than one method have a corresponding event adapter class that implements the interface and defines each method in the interface with an empty method body.
- For example, the adapter class for the WindowListener interface is WindowAdapter.
- Now instead of implementing an event listener interface, you can extend the corresponding event adapter class and define only those methods in which you are interested.
- For example, The following code segment shows that a class MyWindowAdapter extends WindowAdapter and implements the windowClosing() method.

```

class MyWindowAdapter extends WindowAdapter {
    public void windowClosing(WindowEvent e){
        System.exit(0);
    }
}

```

This class can be registered using the following statement,

```
this.addWindowListener(new MyWindowAdapter());
```

However, if the class is a subclass of a Frame or Applet class, then you cannot define the class as a subclass of WindowAdapter.

In such a case, you can use inner classes.

So the preceding MyWindowAdapter class and addWindowListener statement replaced with the following statements,

```

this.addWindowListener( new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});

```

You can read this syntax as defining an anonymous inner class as a subclass of WindowAdapter, create an instance of this inner class and use this instance as an argument to the addWindowListener () method.

The following program demonstrates how the adapter class is created and used.

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class AdapterExample extends JFrame
{
    AdapterExample()
    {
        this.addWindowListener( new WindowAdapter() {
            public void windowClosing(WindowEvent e)
                {
                    System.exit(0);
                }
        });
    }
}
class AdapterClassJavaExample
{
    public static void main(String [] args)
    {
        AdapterExample frame = new AdapterExample();
        frame.setTitle("Adapter Class Java Example");
        frame.setBounds(100,200,200,200);
        frame.setVisible(true);
    }
}
```

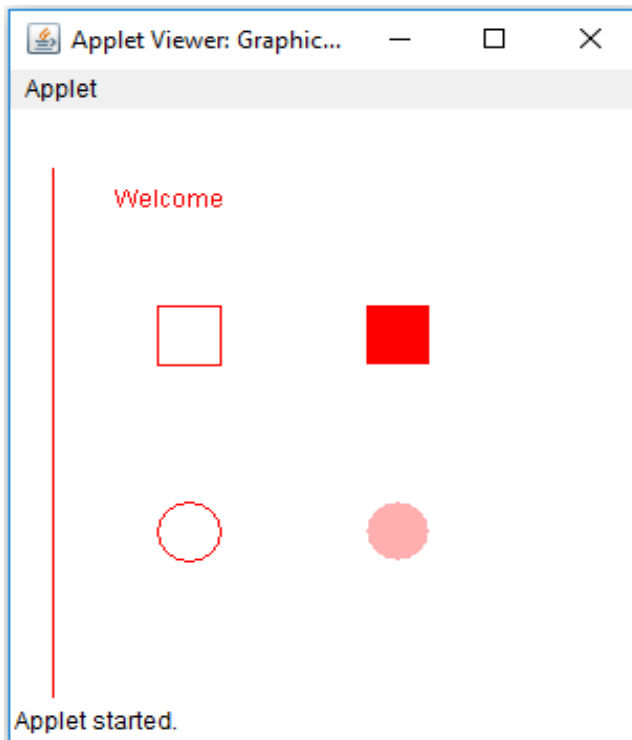
## GraphicsDemo.html

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

### Execution:

```
D:\> javac GraphicsDemo.java
```

```
D:\> appletviewer GraphicsDemo.html
```



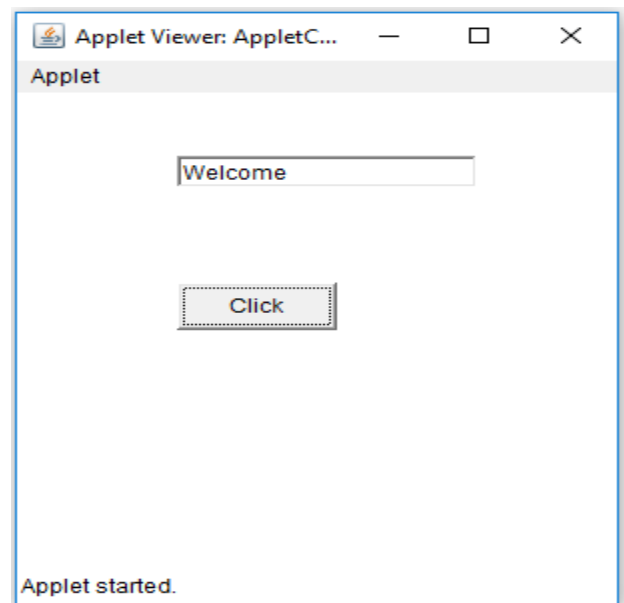
### Components of Applet:

- The components of **AWT** are the components of **Applet**, i.e. we can use AWT components (Button, TextField, Checkbox, TextArea, Choice & etc....) 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 components and event handling in applet that prints a message by click on the button.

### Example: AppletComponents.java

```
import java.applet.*;
import java.awt.*; import
java.awt.event.*;
public class AppletComponents extends Applet implements ActionListener{ Button b;
TextField tf;
public void init(){
tf=new TextField();
tf.setBounds(80,40,150,20);
b=new Button("Click");
b.setBounds(80,120,80,30);
add(b);add(tf);
b.addActionListener(this);
setLayout(null);
}
public void actionPerformed(ActionEvent e)
{
tf.setText("Welcome");
}
}
```



### AppletComponents.html

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

### Execution:

```
D:\>javac AppletComponents.java
D:\>appletviewer AppletComponents.html
*****
```

## Swing

Swing is a framework or API that is used to create GUI (or) window-based applications. It is an advanced version of AWT (Abstract Window 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.	AWT	Swing
1)	AWT components are <b>platform-dependent</b> .	Swing components are <b>platform-independent</b> .
2)	AWT components are <b>heavyweight</b> .	Swing components are <b>lightweight</b> .
3)	AWT provides <b>less components</b> than Swing.	Swing provides <b>more powerful components</b> such as tables, lists, scrollpanes, colorchooser and` etc.
4)	AWT <b>doesn't follows MVC</b> (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing <b>follows MVC</b> .

### **Commonly used Methods of Component class**

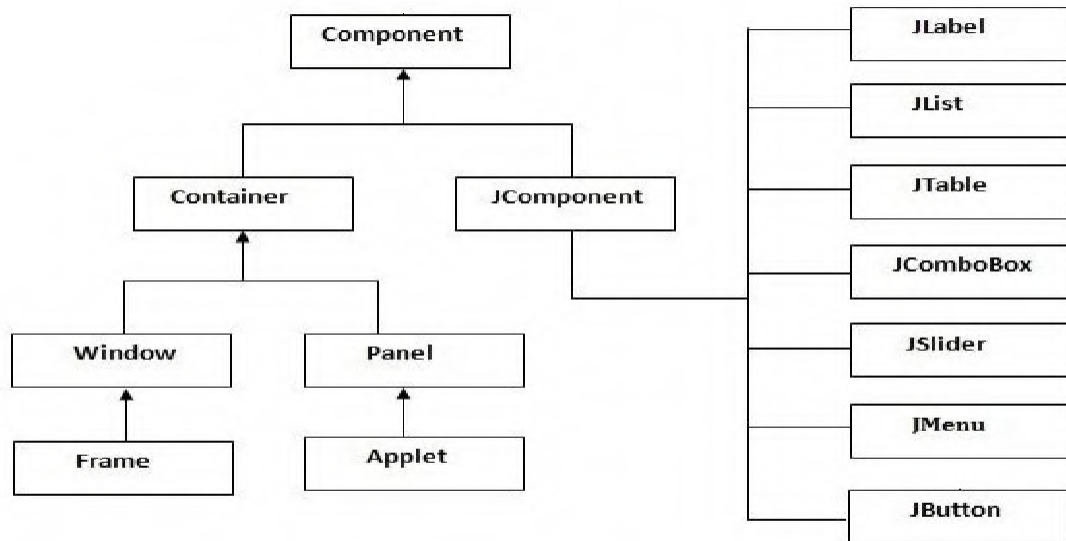
Method	Description
<b>add(Component c)</b>	inserts a component on this component.
<b>setSize(int width,int height)</b>	sets the size (width and height) of the component.
<b>setLayout(LayoutManager m)</b>	defines the layout manager for the component.
<b>setVisible(boolean status)</b>	changes the visibility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

**setTitle(String text)**

Sets the title for component



## Swing Hierarchy



---

To create simple swing example, you need a frame.

- ❖ In swing, we use **JFrame class** to create a frame.

There are two ways to create a frame in swing.

- By extending JFrame class (inheritance)

Ex:

```
class Example extends JFrame
```

```
{
```

```
.....
```

```
}
```

- By creating the object of JFrame class (association)Ex:

```
class Example
```

```
{
```

```
JFrame obj=new JFrame();
```

```
.....
```

```
}
```

## A Simple Swing Example

We can write the code of swing inside the main() or constructor.

### In Main() Method:

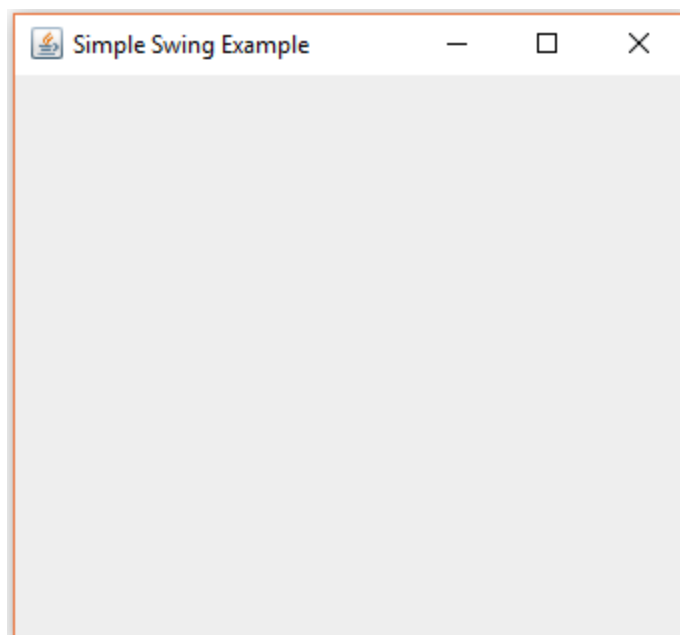
#### SwingExample.java

```
import javax.swing.*; public
class SwingExample
{
public static void main(String[] args)
{
    JFrame f=new JFrame("Simple Swing Example");f.setSize(300,300);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

(OR)

#### In Constructor()

```
import javax.swing.*;
class SwingExample extends JFrame
{
    SwingExample()
    {
        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
        setTitle("SwingExample ");//Set Title
    }
    public static void main(String args[]){
        SwingExample f=new SwingExample();
    }
}
```



## Components of Swing:

### 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.

#### Syntax:

```
JButton b=new JButton("Text");  
(Or)  
JButton b1,b2;  
b1=new JButton("Text");  
b.setBounds(50,100,80,30);
```

### JLabel:

The 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.

#### Syntax:

```
JLabel l1=new JLabel("Text");  
(or)  
JLabel l1,l2;  
l1=new JLabel("Text");
```

### JTextField:

The JTextField class is a text component that allows the editing of a single line text.

#### Syntax:

```
JTextField t1=new JTextField("Text");  
(or)  
JTextField t1,t2;  
t1=new JTextField("Text");
```

### JTextArea :

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

**Syntax:**

```
JTextArea t1=new JTextArea("Text");  
(or)  
JTextArea t1,t2;  
t1=new JTextArea("Text");
```

## **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".

### **Syntax:**

```
JCheckBox c1=new JCheckBox("Text");  
(or)  
JCheckBox c1,c2;  
c1=new JCheckBox("Text");
```

**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 one or more items from list of items.

### **Syntax:**

```
DefaultListModel<String> l1 = new DefaultListModel<>();  
l1.addElement("Item1");  
l1.addElement("Item2");  
l1.addElement("Item3");  
l1.addElement("Item4");  
JList list = new JList<>(l1);
```

## **JPasswordField:**

The JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text.

### **Syntax:**

```
JPasswordField pwd = new JPasswordField();  
pwd.setBounds(100,50,80,30);
```

## **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.

### **Syntax:**

```
ButtonGroup bg=new ButtonGroup();  
JRadioButton r1=new JRadioButton("Male");  
JRadioButton r2=new JRadioButton("Female");
```

```
bg.add(r1);bg.add(r2);
```

## **JComboBox:**

The JComboBox class is used to show popup menu of items. Item selected by user is shown on the top of a menu.(like Choice class in AWT)

### **Syntax:**

```
String country[]={ "India", "Aus", "U.S.A", "England", "Newzealand" };JComboBox  
cb=new JComboBox(country);  
cb.setBounds(50, 50,90,20);
```

## **JTable:**

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

### **Syntax:**

```
String data[][]= { {"521","Madhu","43400"}, {"512","Hari","54500"},  
                  {"509","Ganesh","70000" } };  
String column[]={ "ID", "NAME", "SALARY" };  
JTable jt=new JTable(data,column);  
jt.setBounds(30,40,200,300);
```

## **JPanel:**

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

### **Syntax:**

```
JPanel panel=new JPanel();  
panel.setBounds(40,80,200,200);  
panel.setBackground(Color.gray);  
JButton b1=new JButton("Button 1");  
b1.setBounds(50,100,80,30);  
panel.add(b1);
```

## **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.

- Unlike JFrame, it doesn't have maximize and minimize buttons.

**Syntax:**

```
JFrame f= new JFrame();  
JDialog d=new JDialog(f , "Dialog", true);  
JButton b = new JButton ("OK"); d.add(b);
```



**Example:** An example for **JButton** Component in swing.

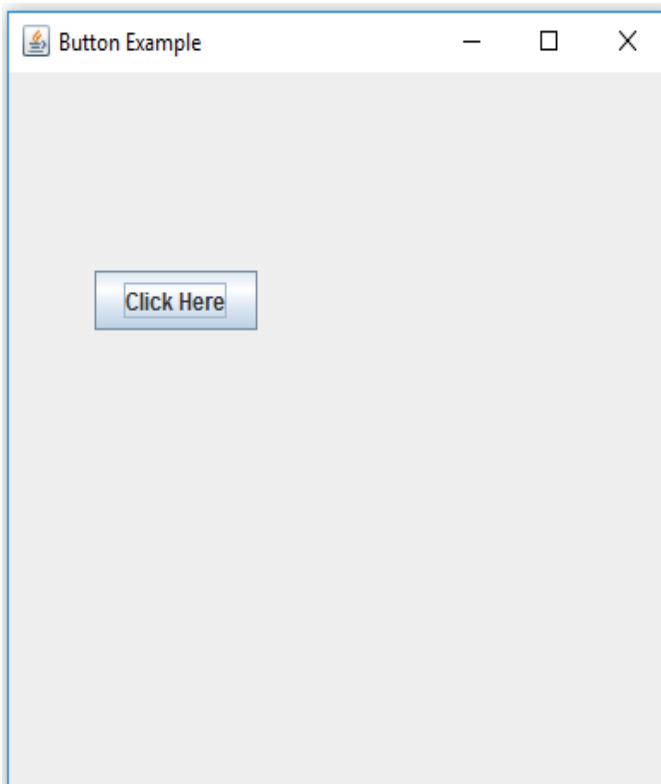
### **JButtonExample.java**

```
import javax.swing.*;
public class JButtonExample
{
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);
}
}
```

### **Execution:**

D:/>javac JButtonExample.java

D:/>java JButtonExample



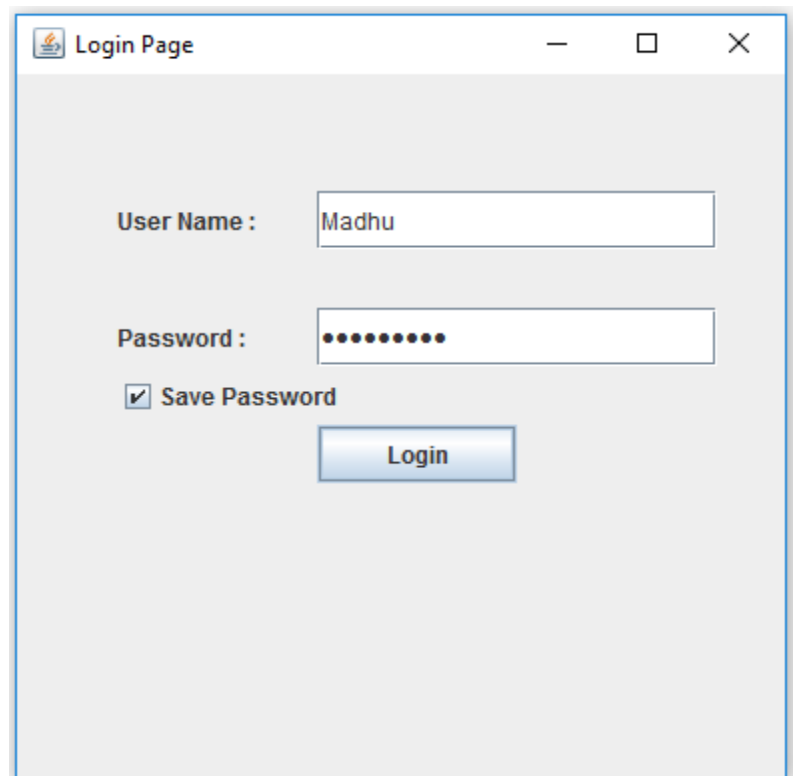
**Example:** An example for Login page which includes **JLabel, JTextField, JPasswordField, JCheckBox and JButton** Components.

### LoginExample.java

```
import javax.swing.*;
class LoginExample extends JFrame
{
public static void main(String args[])
{
    JLabel l1,l2;
    JTextField t1;
    JPasswordField p1;
    JCheckBox cb;
    JButton b;
    JFrame f=new JFrame("Login Page");
    l1=new JLabel("User Name :");
    l1.setBounds(50,60,100,30);
    t1=new JTextField("");
    t1.setBounds(150,60, 200,30); l2=new
    JLabel("Password :");
    l2.setBounds(50,120,100,30); p1=new
    JPasswordField("");
    p1.setBounds(150,120, 200,30); cb=new
    JCheckBox("Save Password");
    cb.setBounds(50,150,200,30);
    b=new JButton("Login");
    b.setBounds(150,180,100,30);
    f.add(l1);f.add(l2);
    f.add(t1);f.add(p1);
    f.add(cb);f.add(b);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

### Execution:

```
D:/>javac LoginExample.java
D:/>java LoginExample
```



**Example:** An example for **JPanel** Class component of swing

**JPanelExample.java**

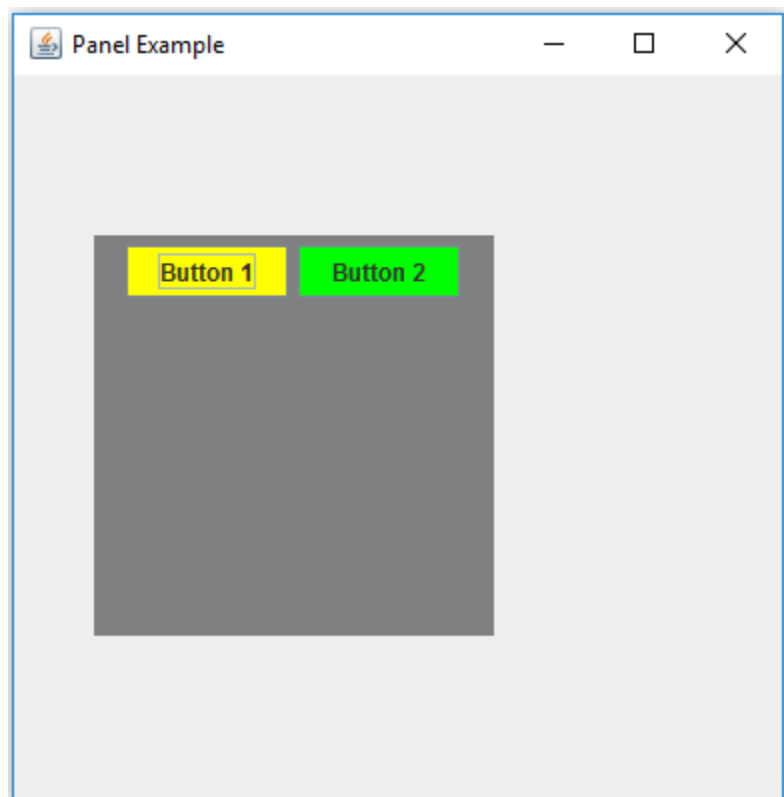
```
import java.awt.*;
import javax.swing.*;

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

**Execution:**

D:/> javac JPanelExample.java

D:/> java JPanelExample



**Example:** An example for **JRadioButton** Class component of swing

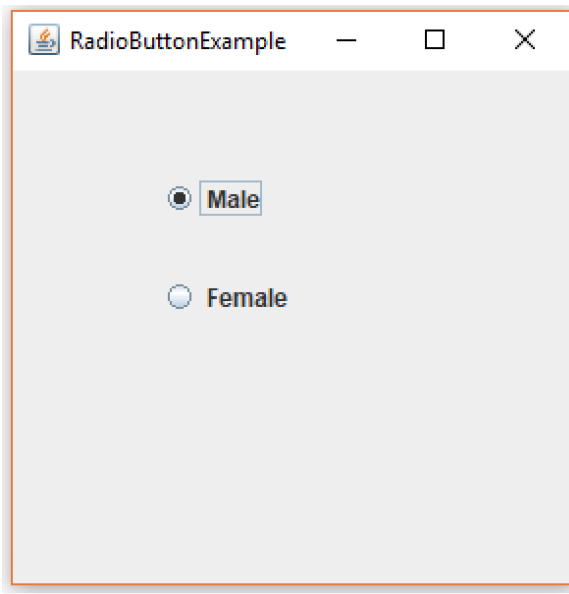
### JRadioButtonExample.java

```
import javax.swing.*;
public class JRadioButtonExample
{
public static void main(String[] args)
{
JFrame f=new JFrame();
JRadioButton r1=new JRadioButton(" Male");
JRadioButton r2=new JRadioButton(" 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);
f.setTitle("RadioButtonExample");
}
}
```

### Output:

Javac JRadioButtonExample.java

Java JRadioButtonExample



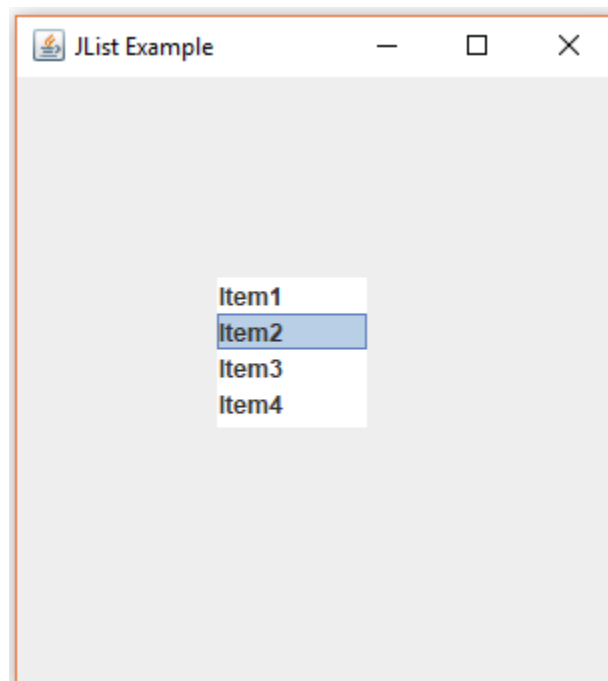
**Example:** An example for **JList** Class component of swing

**JListExample.java**

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

**Output:**

```
Javac JListExample.java
Java JListExample
```



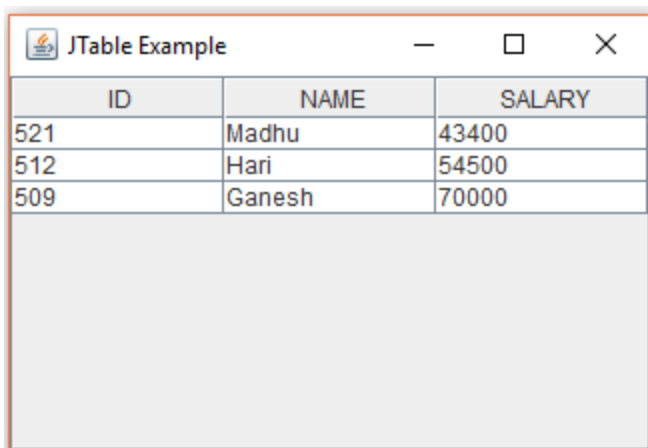
**Example:** An example for **JTable** Class component of swing

```
JTableExample.java import
javax.swing.*; public class
JTableExample

{
public static void main(String[] args)
{
    JFrame f=new JFrame();
    String data[][]={{ "521","Madhu","43400"},
                    {"512","Hari","54500"},
                    {"509","Ganesh","70000"} };
    String column[]={ "ID","NAME","SALARY"};
    JTable jt=new JTable(data,column);
    jt.setBounds(30,40,200,300);
    JScrollPane sp=new JScrollPane(jt);
    f.add(sp);
    f.setSize(300,400);
    f.setVisible(true);
    f.setTitle("JTable Example");
}
}
```

**Output:**

Javac JTableExample.java  
Java JTableExample



ID	NAME	SALARY
521	Madhu	43400
512	Hari	54500
509	Ganesh	70000

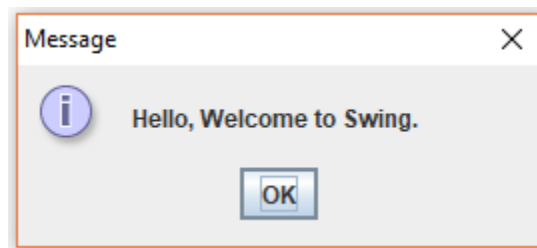
**Example:** An example for **JOptionPane** Class component of swing

### **JOptionPaneExample.java**

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

### **Output:**

```
Javac JOptionPaneExample.java
Java JOptionPaneExample
```



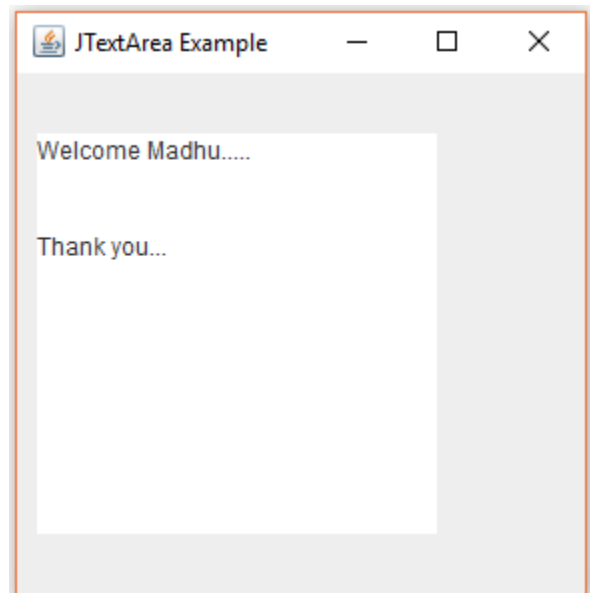
**Example:** An example for **JTextArea** Class component of swing

### **JTextAreaExample.java**

```
import javax.swing.*;
class JTextAreaExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("JTextArea Example");
        JTextArea area=new JTextArea("Welcome");
        area.setBounds(10,30, 200,200);
        f.add(area);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

### **Output:**

```
Javac JTextAreaExample.java
Java JTextAreaExample
```



**Example:** An example for **JProgressBar** Class component of swing

### **JProgressBarExample.java**

```
import javax.swing.*;
public class JProgressBarExample extends JFrame
{
    JProgressBar jb;
    int i=0,num=0;
    JProgressBarExample()
    {
        jb=new JProgressBar(0,2000);
        jb.setBounds(40,40,160,30);
        jb.setValue(0);
        jb.setStringPainted(true);
        add(jb);
        setSize(250,150);
        setLayout(null);
        setTitle("JProgressBar");
    }
    public void iterate()
    {
        while(i<=2000)
        {
            jb.setValue(i);
            i=i+20;
            try{Thread.sleep(150);}catch(Exception e){}
        }
    }
    public static void main(String[] args)
    {
        JProgressBarExample m=new JProgressBarExample();
        m.setVisible(true);
        m.iterate();
    }
}
```

### **Output:**

```
Javac JProgressBarExample.java
Java JProgressBarExample
```

