



SANJIVANI
GROUP OF INSTITUTES



SANJIVANI K. B. P. POLYTECHNIC, KOPARGAON

With NBA ACCREDITED programs , Approved by AICTE, New Delhi,
Recognized by Govt. of Maharashtra, Affiliated to Maharashtra State Board of
Technical Education, Mumbai, ISO 9001:2015 Certified Institute

Department:- Computer Technology

Class:- TYCM-A and B

Name of Subject:- AJP

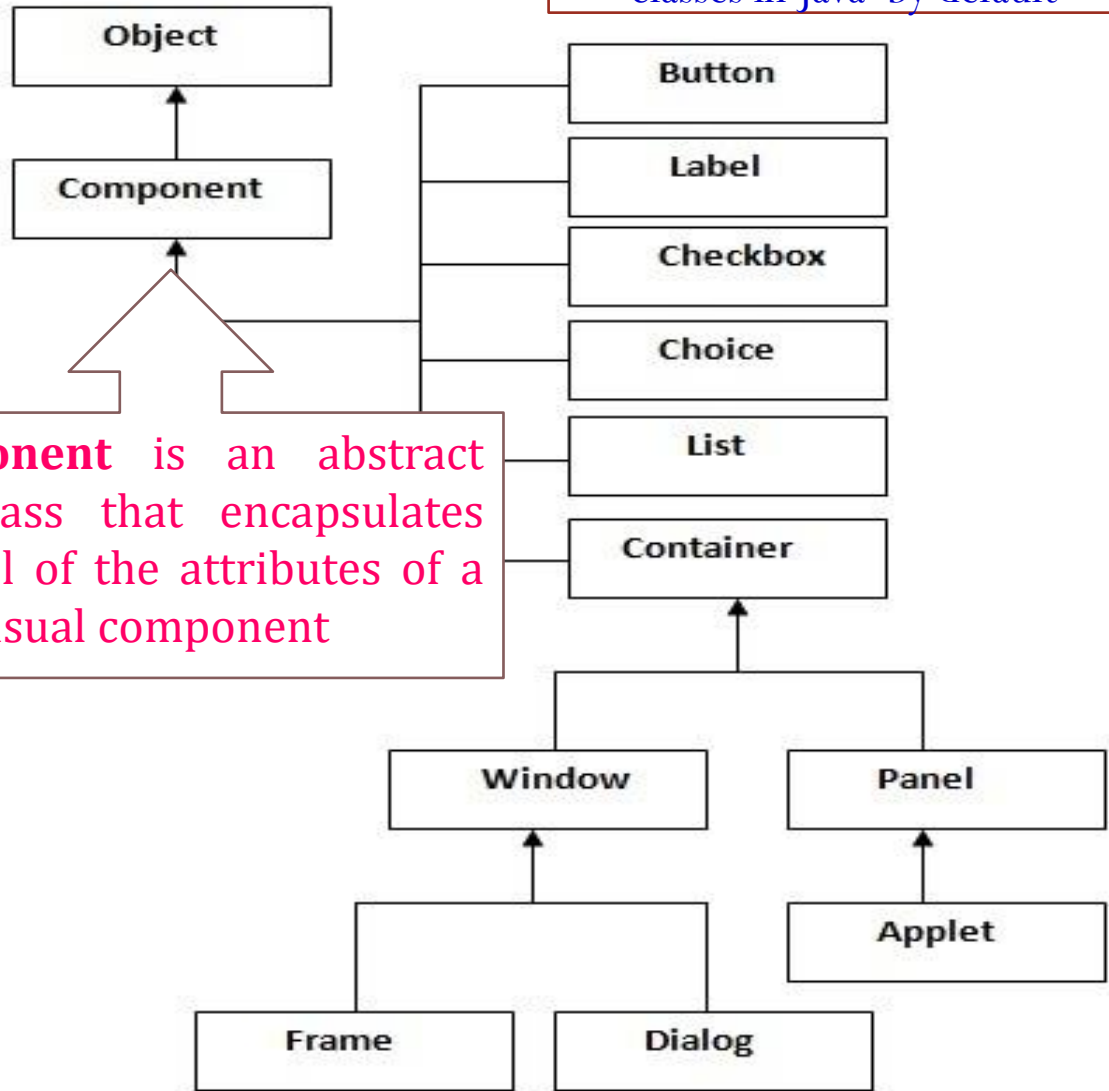
MSBTE Subject Code:- 22517

Unit -1 Abstract Windowing Toolkit

1.1 Component, Container, Window, Frame , Panel

Java AWT Hierarchy

Object is the **Top most class** and Parent of all the classes in Java by default



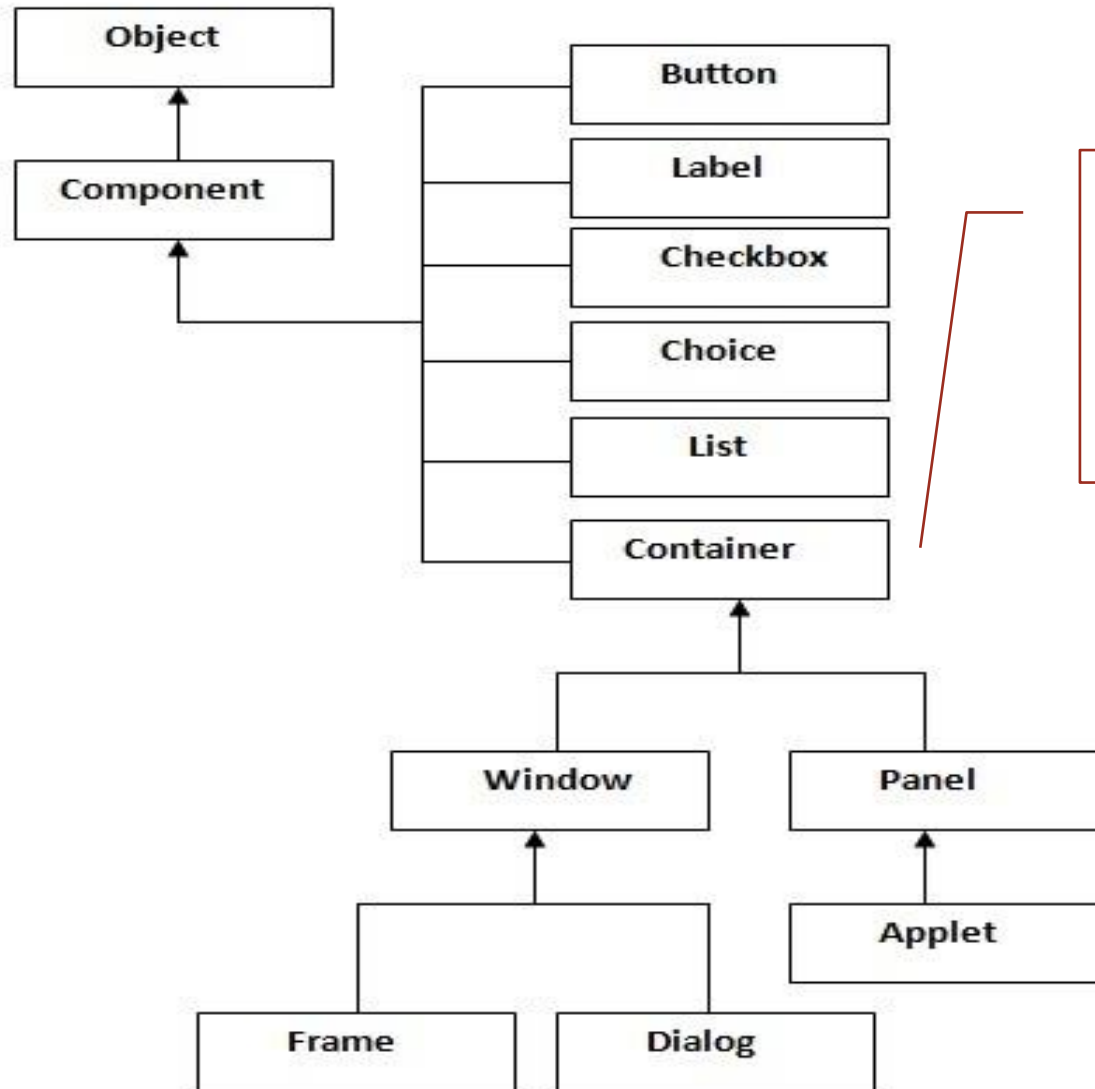
Component is an abstract class that encapsulates all of the attributes of a visual component

2. Component-

Methods of Component class are –

- 1. void add (Component c)** – Add / insert a component on another component.
- 2. void setSize(int width, int height)** – Sets the size of the Component.
- 3. void setLayout(LayoutManager m)** – Sets the layout manager for the component.
- 4. void setVisible(boolean b)** – Sets the visibility of the component. It is by default false.
- 5. void remove(Component c)**- Remove a component.
- 6. void setBounds(int x, int y, int width, int height)** - Used to set the location and size of a single component and is only useful if null layout is used by the container that holds this component.

Java AWT Hierarchy

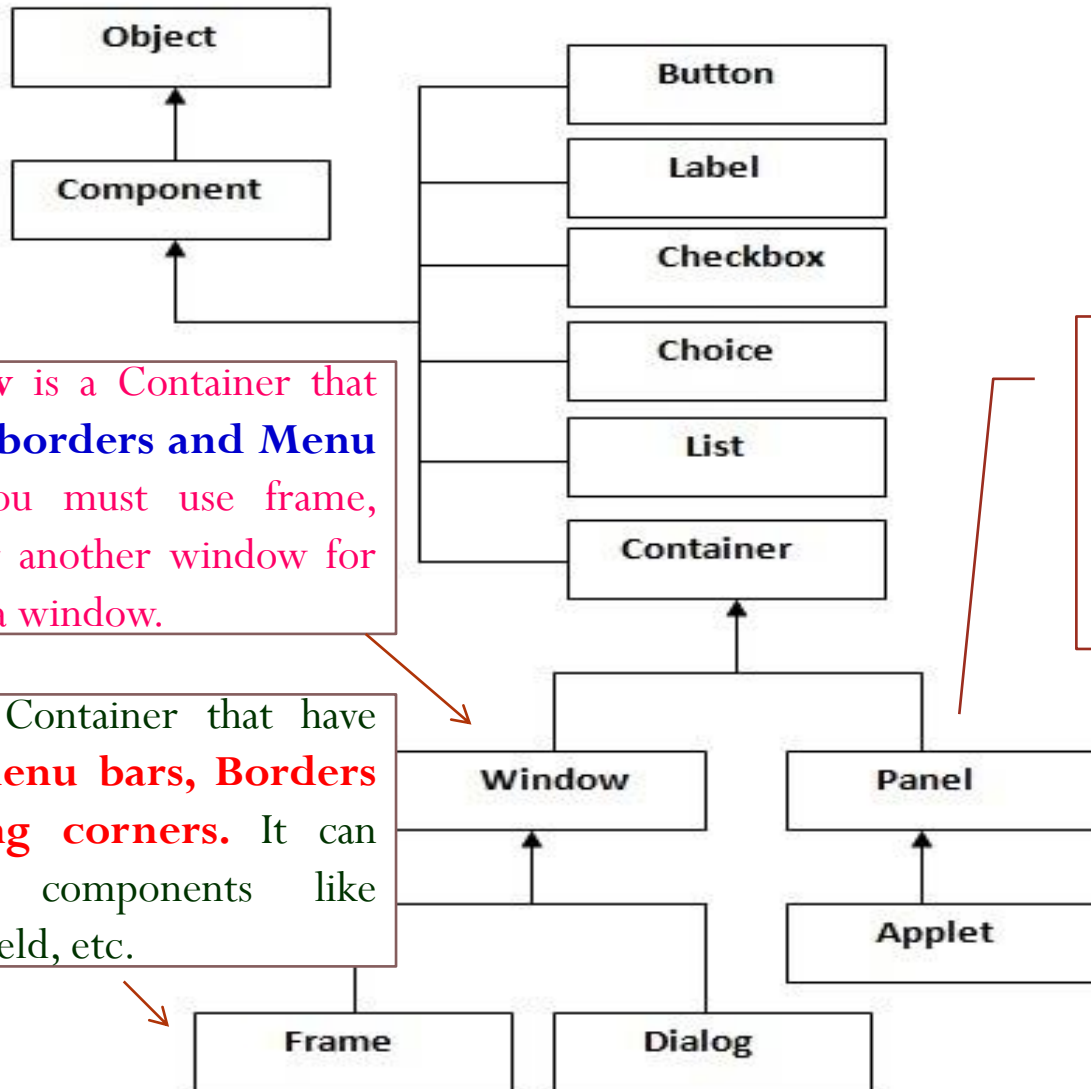


Container is a Component in AWT that can contain another components like buttons, textfield, labels etc. The classes that extends Container class are known as Container such as Frame, Dialog and Panel.

3. Container

1. Containers can contain Components and are themselves components. Other Container objects can be stored inside of a Container. (like Pocket inside a Pocket)
2. A container is responsible for positioning any components that it contains.
3. It does this through the use of various Layout Managers.
4. Containers usually handle Events that occurred to the components.
5. The Containers class is defined in the **java.awt** package from which two commonly used containers - **Frame and Panel** are derived directly or indirectly.

Java AWT Hierarchy



Window is a Container that have **no borders and Menu bars**. You must use frame, dialog or another window for creating a window.

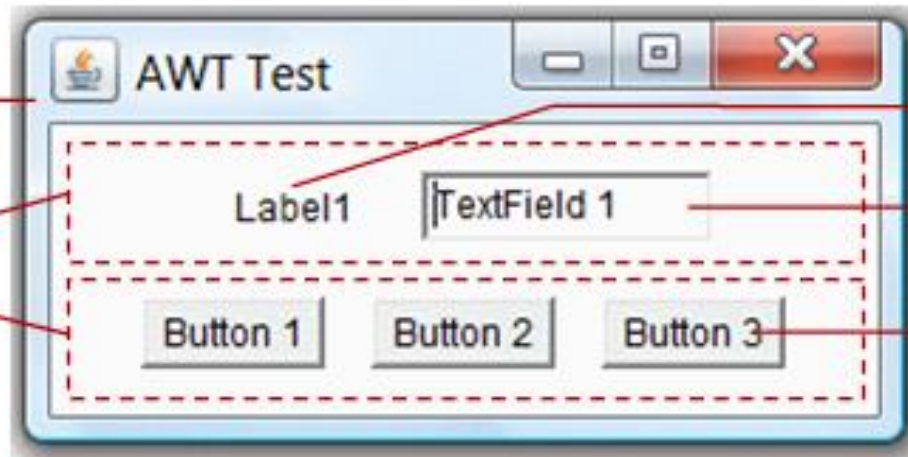
Frame is a Container that have **Title bar, Menu bars, Borders and Resizing corners**. It can have other components like buttons, textfield, etc.

Panel is the Container that does not contain **Title bar, Menu bars or border**. It can have other components like buttons, textfield, etc.

Containers

Frame
(Top-level container)

Panel
(Partitions)



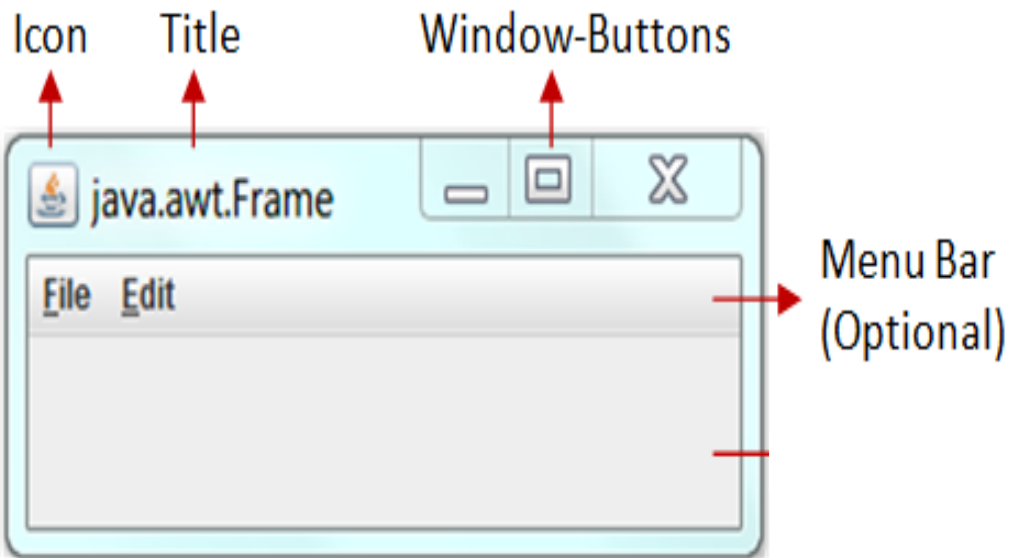
Components

Label

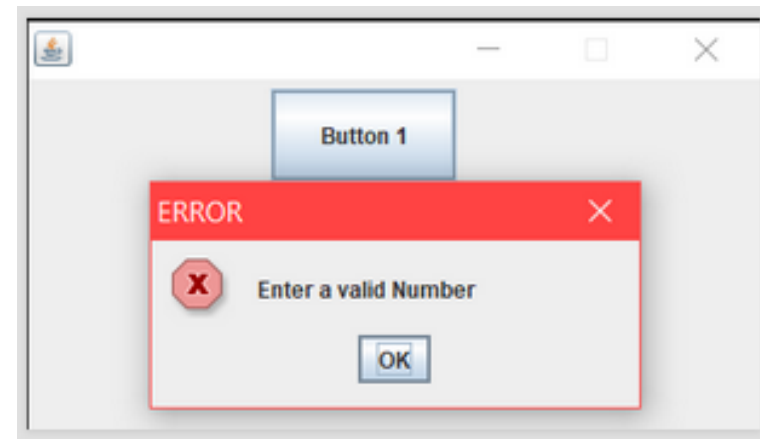
TextField

Button

Frame



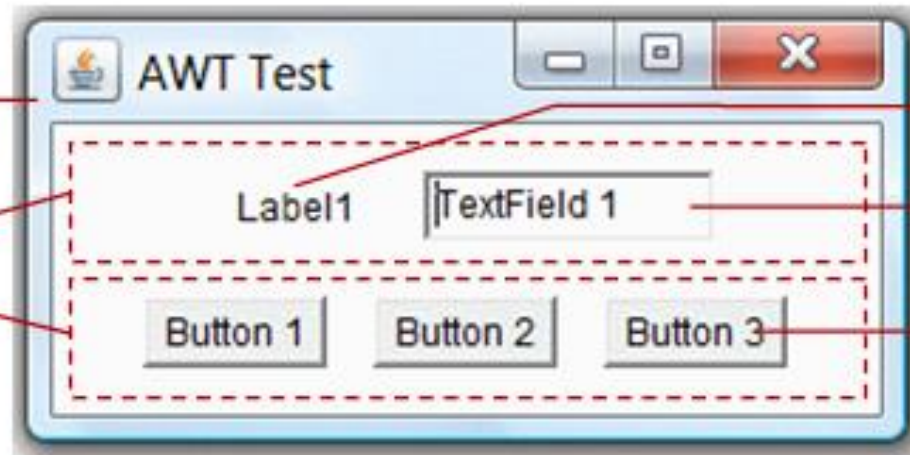
Dialog Box



Containers

Frame
(Top-level container)

Panel
(Partitions)

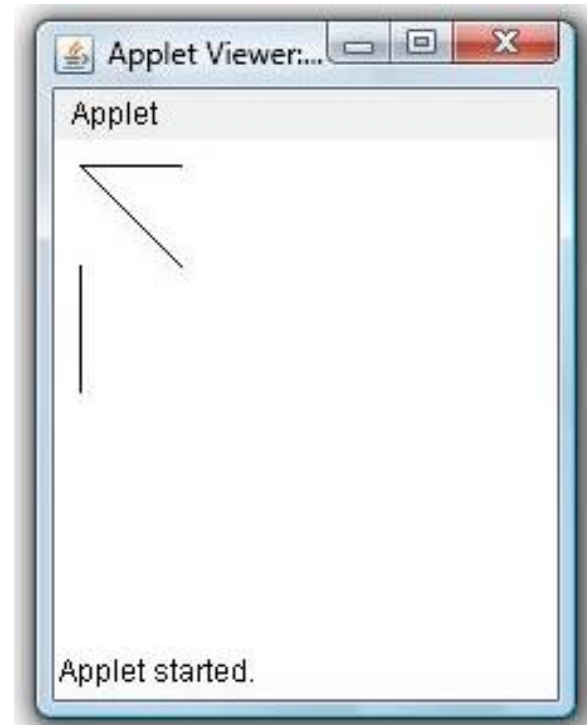
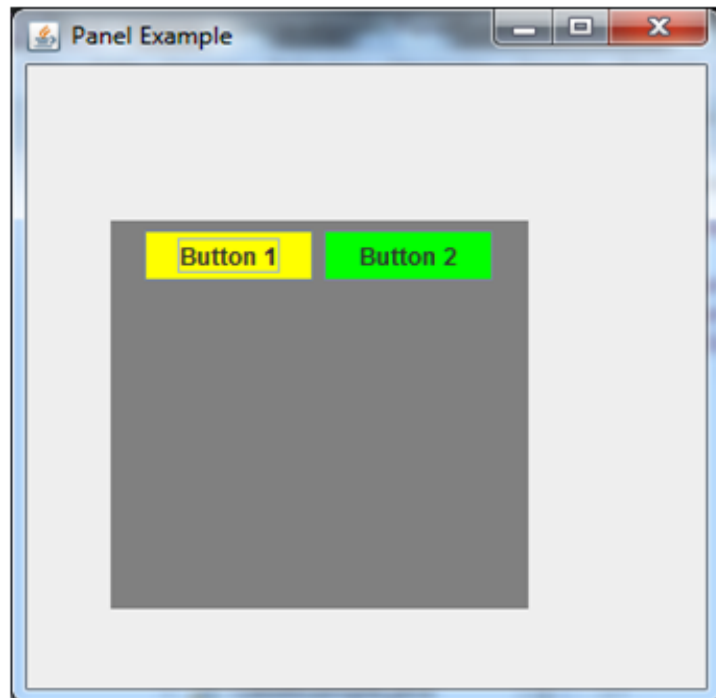


Components

Label

TextField

Button



1.2 Creating Windows program and Applet

Default Settings of Frame

1. Created Frame is invisible

- **setVisible(true);**

2. Can't Close

- **Ctrl + c to close**

3. Size of Frame is 0, 0

- **setSize(300, 500);**

4. No Title

- **setTitle("My Frame");**

Let's see a simple example of AWT where we are creating instance of Frame class.

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

Output -

```
public class MyFrame
```

```
{
```

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

```
    {
```

```
        Frame f = new Frame( );
```

```
        f.setVisible(true); //now frame will be visible, by default not visible
```

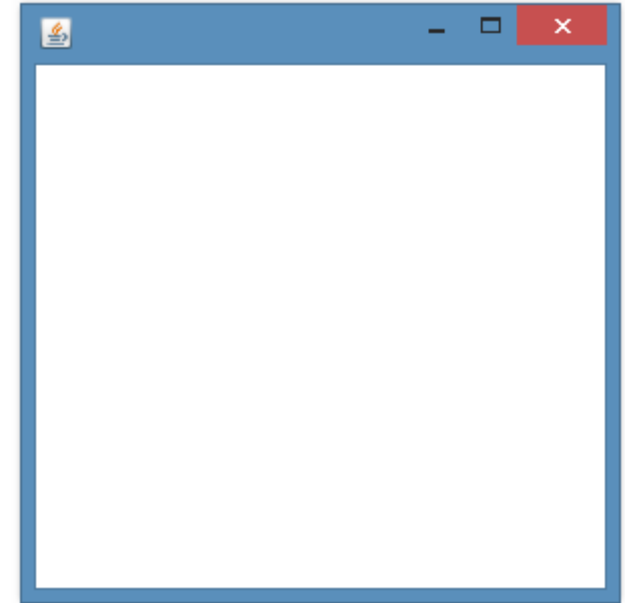
```
    }
```

```
}
```



Let's see a simple example of AWT where we are creating instance of Frame class.

Output -



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

```
public class MyFrame
```

```
{
```

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

```
    {
```

```
        Frame f = new Frame( );
```

```
        f.setVisible(true); //now frame will be visible, by default not visible
```

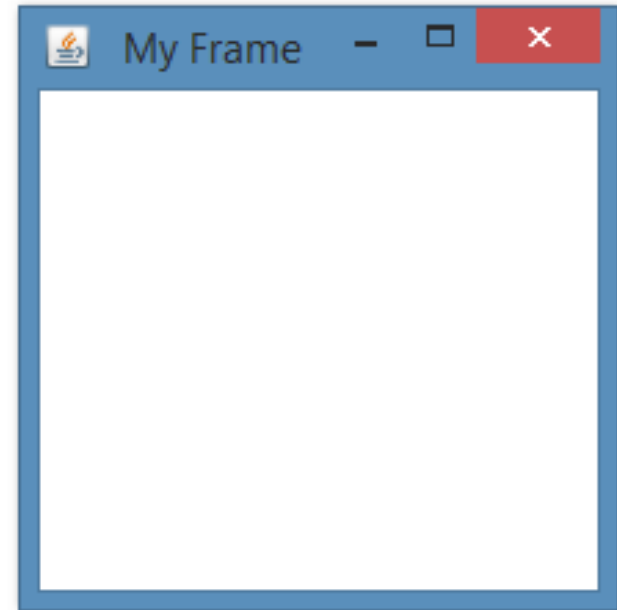
```
        f.setSize(300,300); // frame size 300 width and 300 height
```

```
    }
```

```
}
```

Let's see a simple example of AWT where we are creating instance of Frame class.

Output -



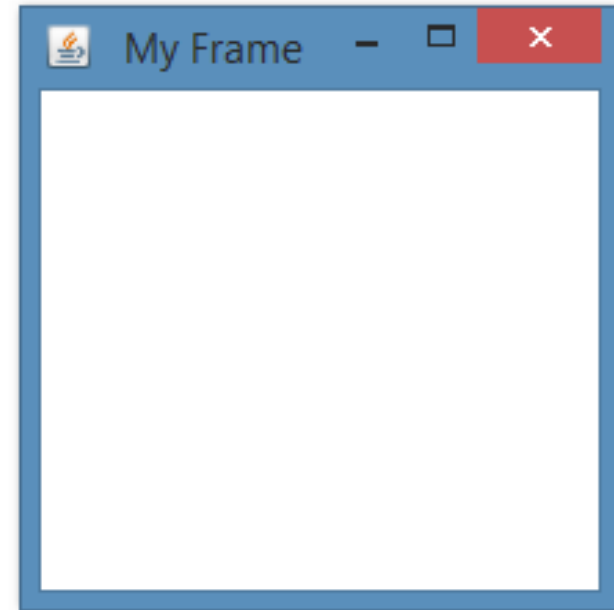
```
import java.awt.*;

public class MyFrame
{
    public static void main(String args[ ])
    {
        Frame f = new Frame( );
        f.setVisible(true); //now frame will be visible, by default not visible
        f.setSize(300, 300); // frame size 300 width and 300 height
        f.setTitle("My Frame");
    }
}
```

Let's see a simple example of AWT where we are inheriting Frame class.

Output -

```
import java.awt.*;  
  
public class MyFrame extends Frame  
{  
    MyFrame()  
    {  
        setVisible(true);  
        setSize(300, 300);  
        setTitle("My Frame");  
    }  
    public static void main(String args[ ])  
    {  
        MyFrame f = new MyFrame();  
  
    }  
}
```



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

Output -

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

```
public class AwtFrame
```

```
{
```

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

```
{
```

```
    Frame f = new Frame( "Java AWT Frame");
```

```
    Label l1 = new Label ("Welcome to Sanjivani", Label.CENTER);
```

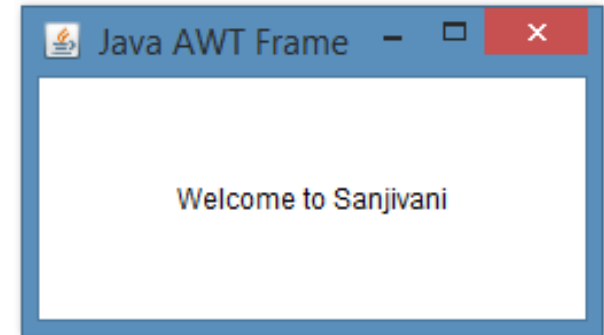
```
    f.add(l1); // adding Label into frame
```

```
    f.setSize(300, 300); // frame size 300 width and 300 height
```

```
    f.setVisible(true); //now frame will be visible, by default not visible
```

```
}
```

```
}
```



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

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

```
public class Testawt
```

```
{
```

```
    Testawt ( )
```

```
    {
```

```
        Frame f = new Frame( );
```

```
        Button b = new Button ("click me");
```

```
        b.setBounds(20, 20, 60, 50); //setting button position
```

```
        f.add(b);
```

```
        f.setSize(300, 300);
```

```
        f.setLayout(null);
```

```
        f.setVisible(true);
```

```
    }
```

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

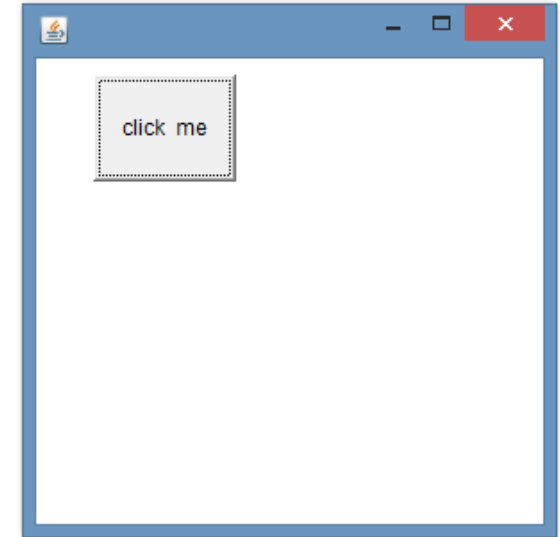
```
    {
```

```
        Testawt t = new Testawt( );
```

```
    }
```

```
}
```

Output -

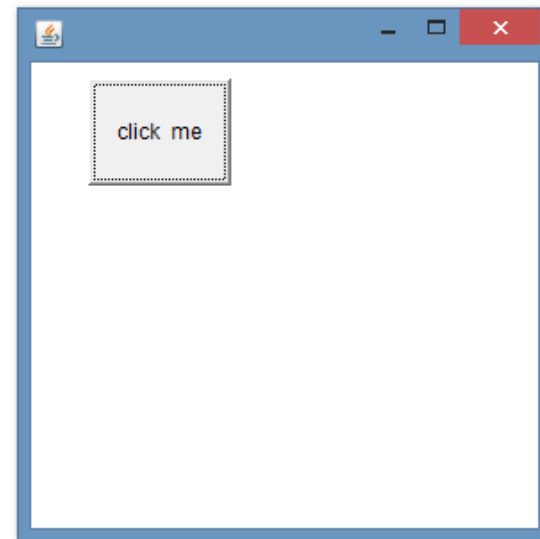


Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

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

```
public class First extends Frame
{
    First()
    {
        setSize(300, 300);
        setLayout(null);
        setVisible(true);
        Button b = new Button ("click me");
        b.setBounds(40, 70, 60, 50);
        add(b);
    }
    public static void main(String args[])
    {
        First t = new First();
    }
}
```

Output -

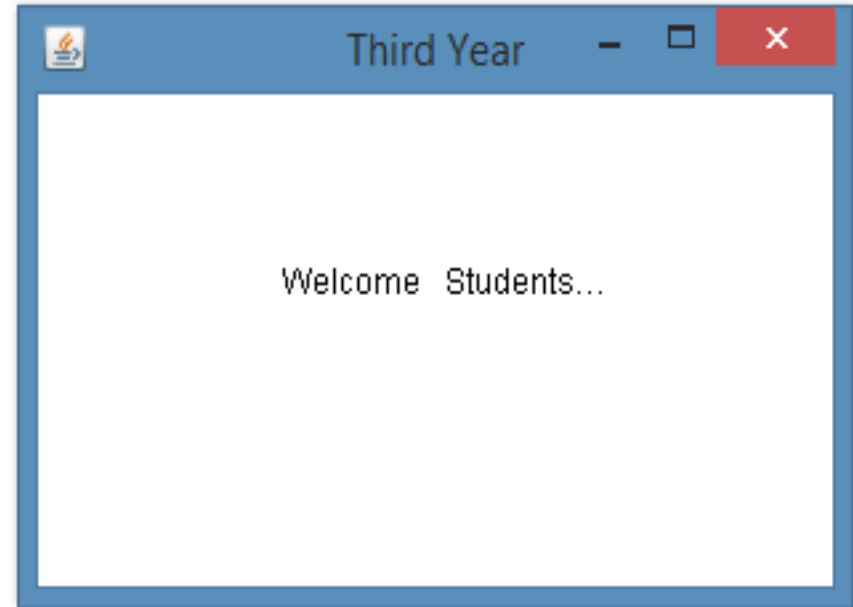


Let's see a simple example of AWT where we are inheriting Frame class.

```
import java.awt.*;

public class Simple extends Frame
{
    Simple()
    {
        setSize(400, 400);
        setLayout(null);
        setVisible(true);
        setTitle("Third Year");
    }
    public void paint( Graphics g)
    {
        g.drawString(" Welcome Students..." , 100, 100);
    }
    public static void main(String args[ ])
    {
        Simple s = new Simple();
    }
}
```

Output -

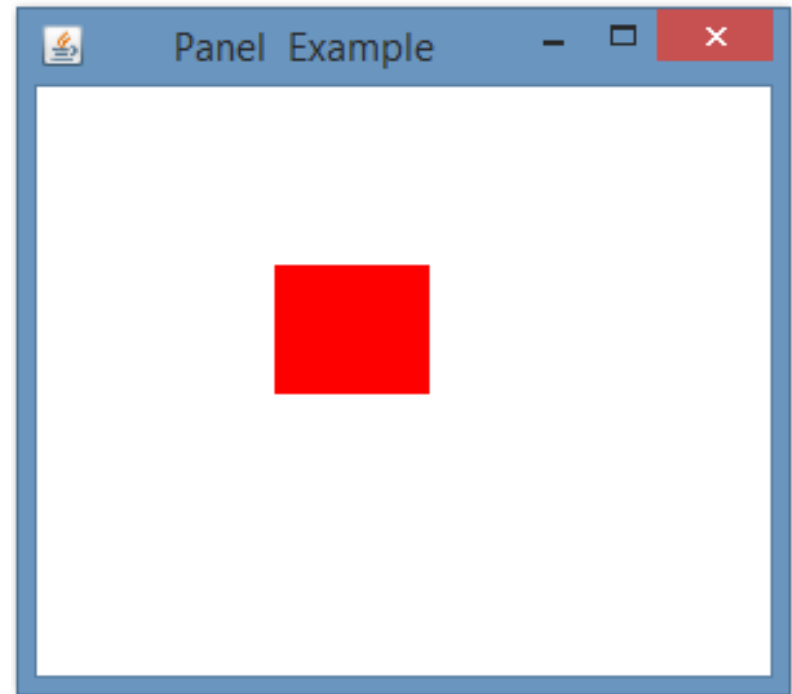


Let's see a simple example of Panel

```
import java.awt.*;

public class SimplePanel extends Frame
{
    SimplePanel ( )
    {
        Frame f = new Frame( " Panel Example");
        Panel p = new Panel();
        p.setBounds(20, 20, 60, 50);
        p.setBackground(Color.red);
        f.add(p);
        f.setSize(300, 300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[ ])
    {
        SimplePanel sp = new SimplePanel();
    }
}
```

Output -

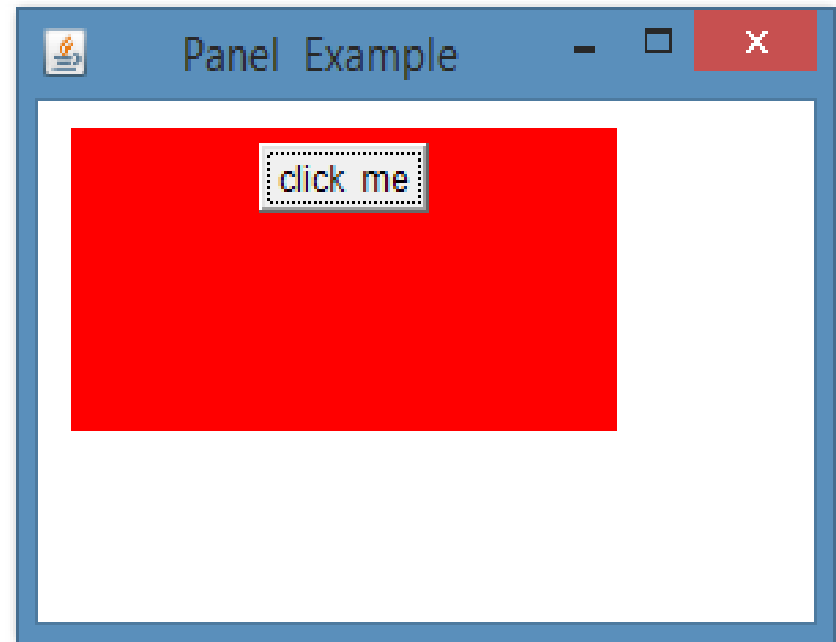


```
import java.awt.*;

public class SimplePanel extends Frame
{
    SimplePanel ( )
    {
        Frame f = new Frame("Panel Example");
        Panel p = new Panel( );
        p.setBounds(20,40, 200, 100); //setting panel position
        p.setBackground(Color.red);
        f.add(p);

        Button b = new Button ("click me");
        b.setBounds(40, 70, 60, 50); //setting button position
        p.add(b); // adding button into frame
        f.setSize(400, 400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[ ])
    {
        SimplePanel sp = new SimplePanel( );
    }
}
```

Output -



Unit -1 Abstract Windowing Toolkit

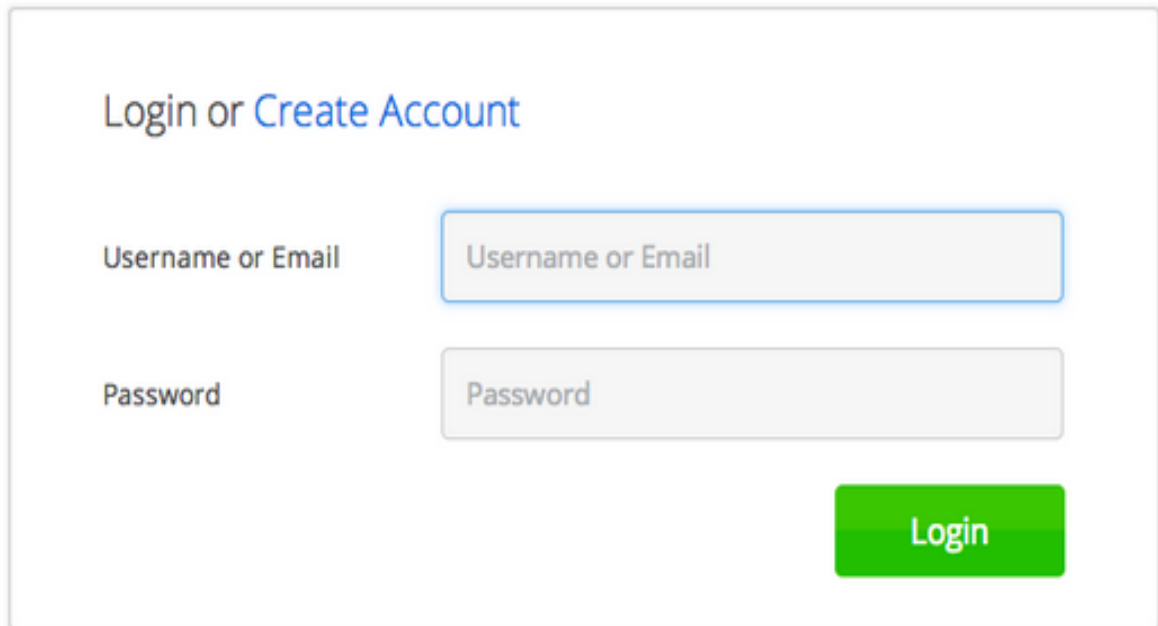
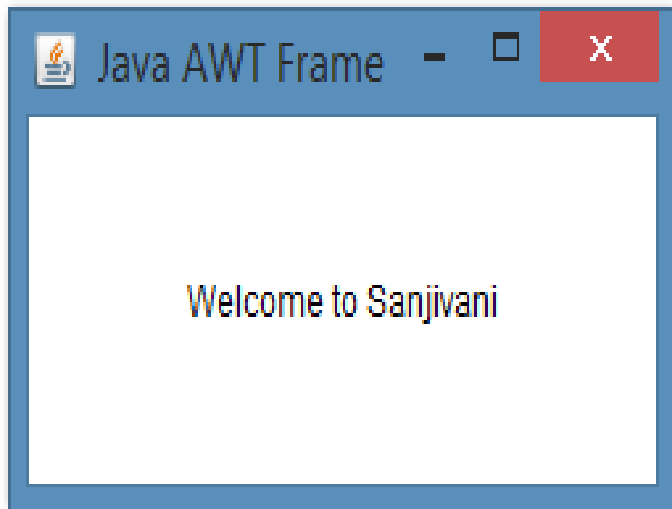
1.3 AWT Control – Label

AWT Controls

1. Controls are the components that allow a user to interact with java application in various ways.
2. Containers (Frame, Panel, Applet) are used to hold components using in a specific layout.
3. **The AWT supports various types of controls such as Labels, Buttons, TextField, TextArea, CheckBox , CheckBoxGroup, Choice , List , ScrollBars and so on.**
4. **AWT classes are contained in the `java.awt` package.**

1. Label

1. It is a container for placing text in a container.
2. **It is used to display a single line of read only text.**
3. Labels are passive controls that do not support any interaction with the user.
4. The text can be changed by an application but a user cannot edit it directly.

A mockup of a login form. At the top, it says "Login or Create Account" in a blue font. Below this, there are two input fields. The first is labeled "Username or Email" and the second is labeled "Password". Both fields are light gray with a blue border. At the bottom right, there is a green button with the text "Login" in white.

1. Label

Constructors -

Sr.	Constructors	Description
1	Label ()	Creates an empty label.
2	Label (String text)	Creates a label with a given text. The text is by default left – justified.
3	Label (String text, int alignment)	<p>Creates a label with a given text and given alignment.</p> <p>Where alignment can be</p> <p>Label.LEFT</p> <p>Label.RIGHT</p> <p>Label.CENTER</p>

1. Label

Methods

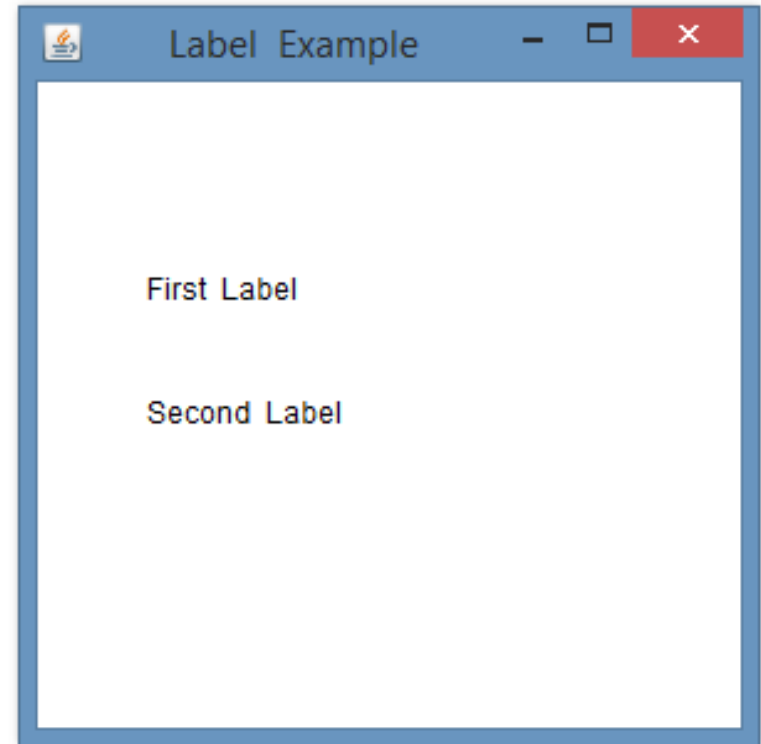
Sr. No	Methods	Description
1	<code>String getText ()</code>	Used to read the Label's text.
2	<code>void setText(String text)</code>	Used to set or change the Label's text.
3	<code>int getAlignment()</code>	Used to gets the current alignment of this label.
4	<code>void setAlignment(int alignment)</code>	Used to set the alignment of the text. Left, Right, Center.

Program - Label

```
import java.awt.*;

public class TestLabel
{
    public static void main(String args[ ])
    {
        Frame f = new Frame( "Label Example");
        Label l1 = new Label ("First Label");
        l1.setBounds(50, 100, 100, 30);
        Label l2 = new Label("Second Label");
        l2.setBounds(50, 150, 100, 30);
        f.add(l1);
        f.add(l2);
        f.setSize(300, 300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

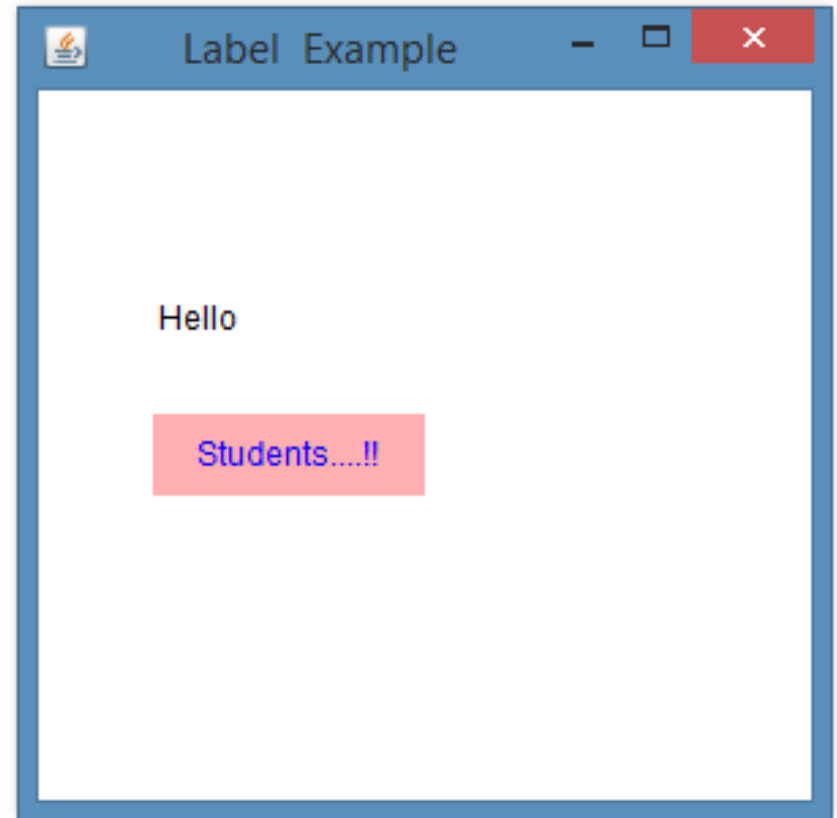
Output -



Program - Label

```
import java.awt.*;
public class LabelExample
{
    public static void main(String args[ ])
    {
        Frame f = new Frame("Label Example");
        Label l1 = new Label ("Hello ");
        l1.setBounds(50, 100, 100, 30);
        Label l2 = new Label ( );
        l2.setText("Students....!!");
        l2.setAlignment(Label.CENTER);
        l2.setBackground(Color.pink);
        l2.setForeground(Color.blue);
        l2.setBounds(50, 150, 100, 30);
        f.add(l1);
        f.add(l2);
        f.setSize(300, 300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output -



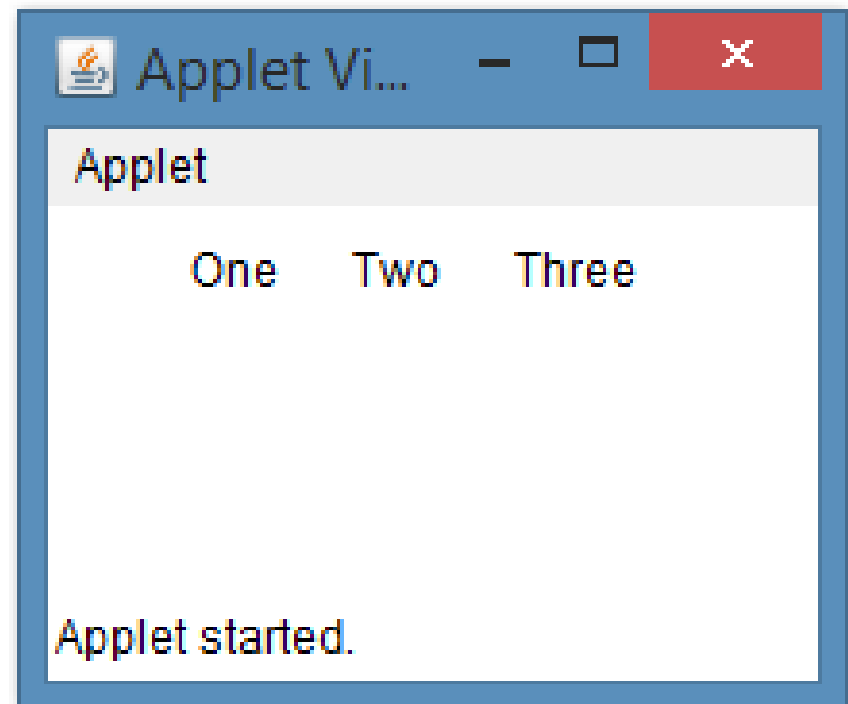
Program – Label using Applet

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

```
/*<applet code="LabelDemo.class" width=200 height=100></applet>*/
```

```
public class LabelDemo extends Applet  
{  
    public void init()  
    {  
        Label l1= new Label("One");  
        Label l2= new Label("Two");  
        Label l3=new Label("Three");  
  
        //add labels to applet window  
        add(l1);  
        add(l2);  
        add(l3);  
    }  
}
```

Output -



Unit -1 Abstract Windowing Toolkit

1.3 AWT Control – Button

2. Button

1. The easiest way to trap the user action.
2. A Push button is a component that contains a label and that generates an event when it is pressed.
3. Button is used to create a Push button control, which can generate an `ActionEvent` when it is clicked or pressed.

Sr. No	Constructors	Description
1	<code>Button ()</code>	Creates an empty Button.
2	<code>Button (String text)</code>	Creates a button with a given text.

2. Button

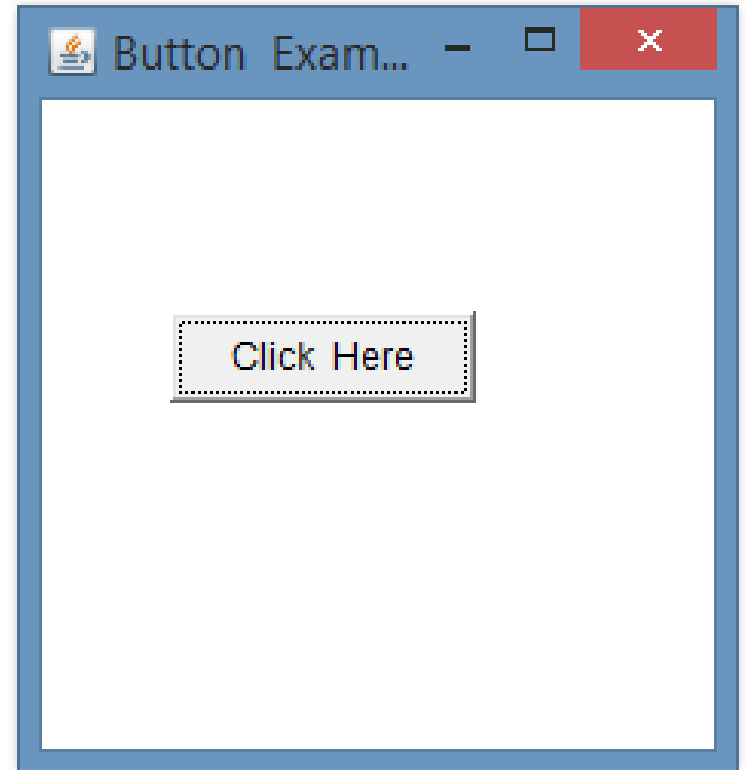
Sr. No	Methods	Description
1	String getLabel()	Get the Label of the Button.
2	void setLabel(String buttonlabel)	Set or change the Label of the Button.
3	void addActionListener (ActionListener l)	Adds the specified action listener to receive action events from this button.
4	void removeActionListener (ActionListener l)	Removes the specified action listener so that it no longer receives action events from this button.
5	String getActionCommand()	Returns the command name of the action event fired by this button.
6	void setActionCommand(String command)	Sets the command name for the action event fired by this button.

Program - Button

```
import java.awt.*;

public class ButtonExample
{
    public static void main(String args[ ])
    {
        Frame f = new Frame( "Button Example");
        Button b = new Button ("Click Here");
        b.setBounds(50, 100, 100, 30);
        f.add(b);
        f.setSize(300, 300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

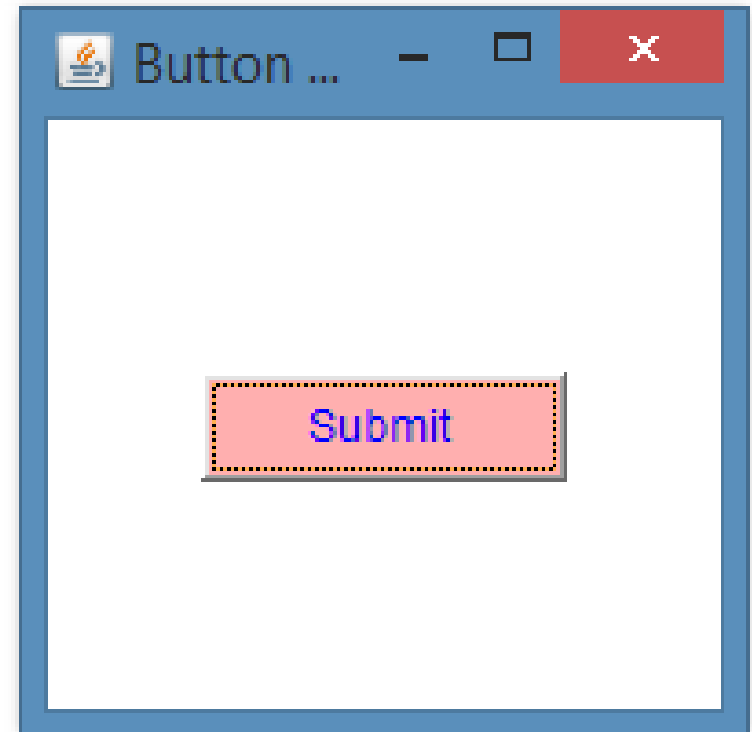
Output -



Program - Button

```
import java.awt.*;  
public class ButtonDemo  
{  
    public static void main(String args[ ])  
    {  
        Frame f = new Frame("Button Example");  
        Button b = new Button ( );  
        b.setLabel("Submit");  
        b.setBackground(Color.pink);  
        b.setForeground(Color.blue);  
        b.setBounds(50, 100, 100, 30);  
        f.add(b);  
        f.setSize(200, 200);  
        f.setLayout(null);  
        f.setVisible(true);  
    }  
}
```

Output -

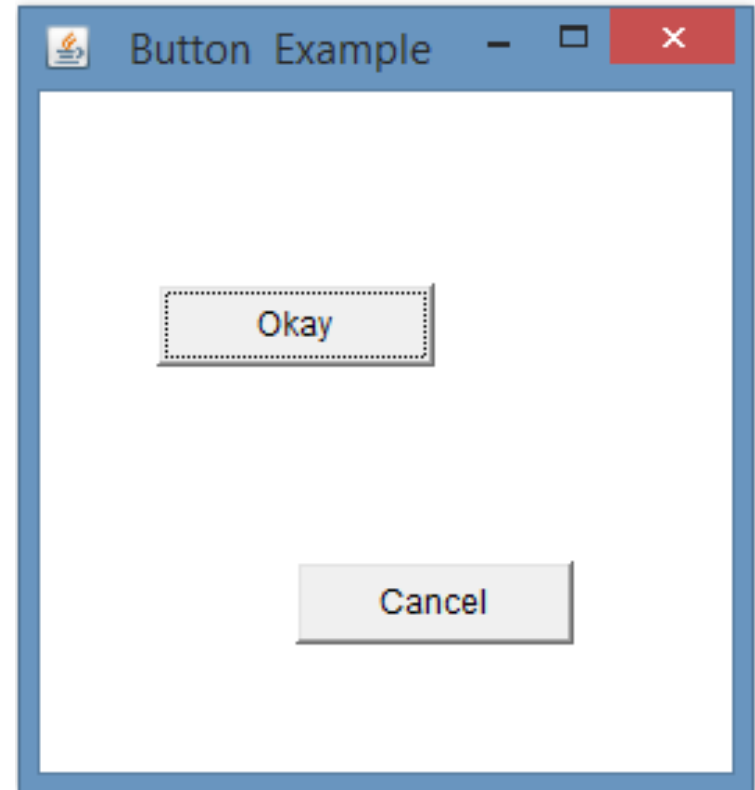


Program - Button

```
import java.awt.*;

public class ButtonEx2
{
    public static void main(String args[ ])
    {
        Frame f = new Frame("Button Example");
        Button b1 = new Button ("Okay");
        b1.setBounds(50, 100, 100, 30);
        Button b2 = new Button ("Cancel");
        b2.setBounds(100, 200, 100, 30);
        f.add(b1);
        f.add(b2);
        f.setSize(300, 300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

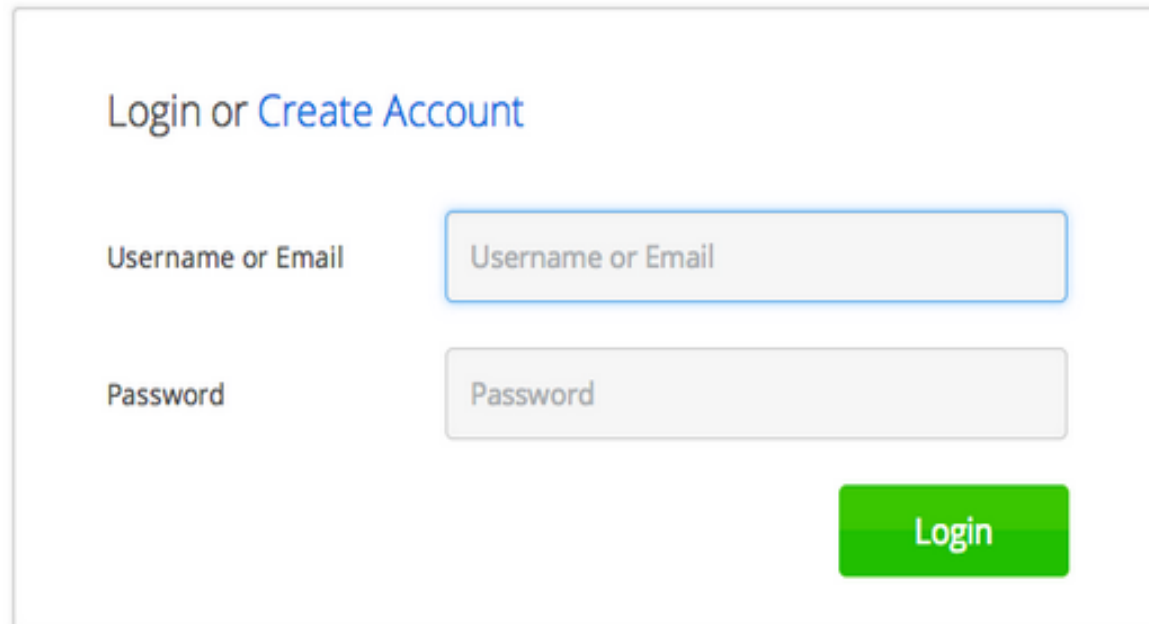
Output -



1.3 AWT Control – Textfield

3. TextField

1. TextField is allow the user to edit the single line of text.
2. Generally accepts one line of input.
3. TextField is allow the user to enter the strings and to edit the text using the arrow keys, cut and paste keys and mouse selections.



Login or [Create Account](#)

Username or Email

Password

[Login](#)

3. TextField

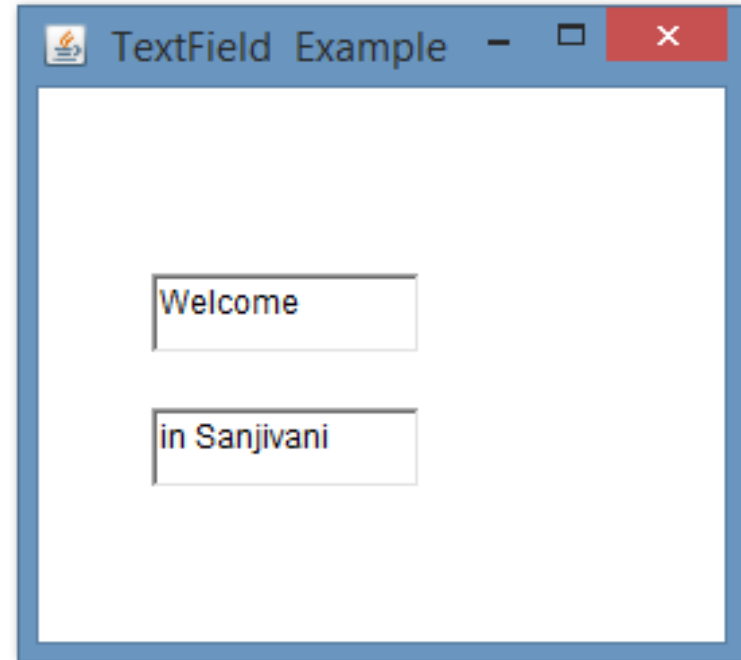
Sr	Constructors	Description
1	TextField()	Creates a new Text field.
2	TextField(String text)	Creates a new Text field initialized with the specified text.
3	TextField(int columns)	Creates a new Text field with the specified number of columns.
4	TextField(String text, int columns)	Creates a new Text field with the given initial text string with the number of columns.

Program - TextField

```
import java.awt.*;

public class TextFieldExample
{
    public static void main(String args[ ])
    {
        Frame f = new Frame( "TextField Example");
        TextField t1 = new TextField ("Welcome");
        t1.setBounds(50, 100, 100, 30);
        TextField t2 = new TextField ("in Sanjivani");
        t2.setBounds(50, 150, 100, 30);
        f.add(t1);
        f.add(t2);
        f.setSize(300, 300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output -

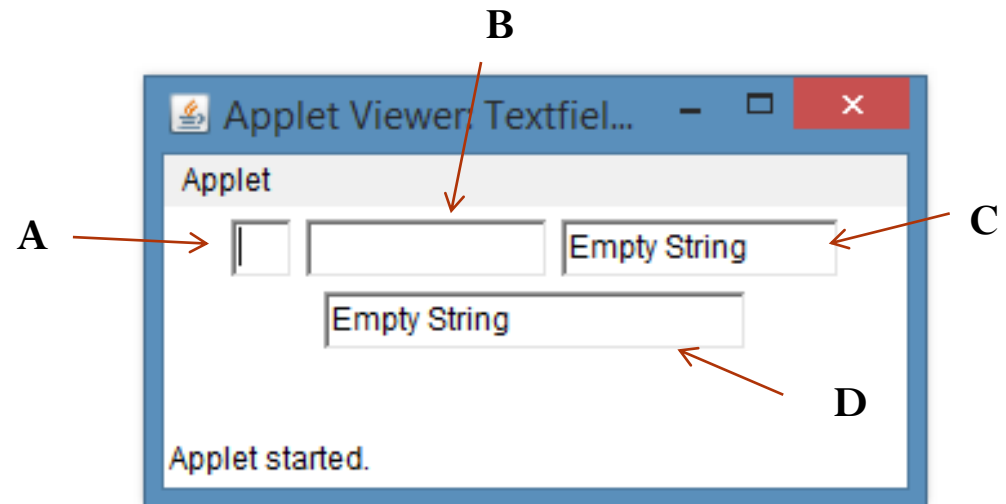


Program - TextField

```
import java.awt.*;
import java.applet.*;
/*<applet code="TextfieldDemo.class" width=200 height=100></applet>*/
public class TextfieldDemo extends Applet
{
    public void init()
    {
        TextField t1= new TextField();           //A
        TextField t2= new TextField(10);         //B
        TextField t3= new TextField("Empty String"); //C
        TextField t4= new TextField("Empty String", 20); //D

        //add TextField to applet window
        add(t1);
        add(t2);
        add(t3);
        add(t4);
    }
}
```

Output -



1.3 AWT Control – TextArea

4. TextArea

1. Sometimes , a single line of text input is not enough for a given task.
2. To handle these situations, the AWT includes a simple multi line editor called TextArea.
3. The user can type here as much as he/she wants.
4. When the text in the text area become larger than the viewable area the scroll bar is automatically appears which help us to scroll the text up and down and right and left.

4. TextArea

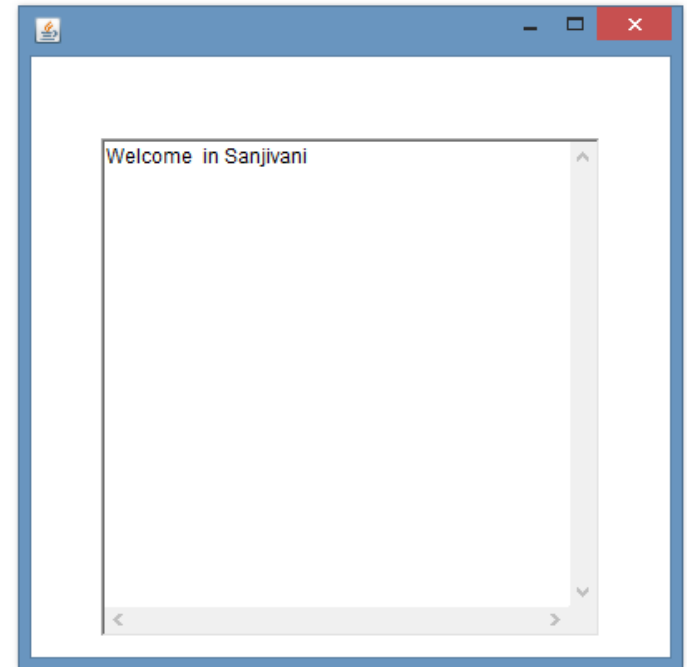
1. Constructors

Sr.	Constructors	Description
1	TextArea()	Creates a new Text area.
2	TextArea(String text)	Creates a new Text area with the specified text.
3	TextArea(int numLines, int numChars)	Here, numLines specifies the height in lines of the text area and numChars specifies its width in characters.
4	TextArea(String text , int numLines, int numChars)	Creates a new Text area with given String, given Height in Lines and width in Characters.
5	TextArea(String text , int numLines, int numChars, int Scrollbars)	Creates a new Text area with given String, given Height in Lines and width in Characters. And Scrollbar visibility.
-	Scrollbar must be one of these values -	SCROLLBARS_BOTH SCROLLBARS_NONE SCROLLBARS_HORIZONTAL_ONLY SCROLLBARS_VERTICAL_ONLY

Program - TextArea

```
import java.awt.*;  
public class TextAreaExample  
{  
    public static void main(String args[ ])  
    {  
        Frame f = new Frame( );  
        TextArea t1 = new Textarea("Welcome in Sanjivani");  
        t1.setBounds(50, 80, 300, 300);  
        f.add(t1);  
        f.add(t2);  
        f.setSize(400, 400);  
        f.setLayout(null);  
        f.setVisible(true);  
    }  
}
```

Output -



```

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

/*<applet code="TextAreaDemo.class" width=200 height=100></applet>*/

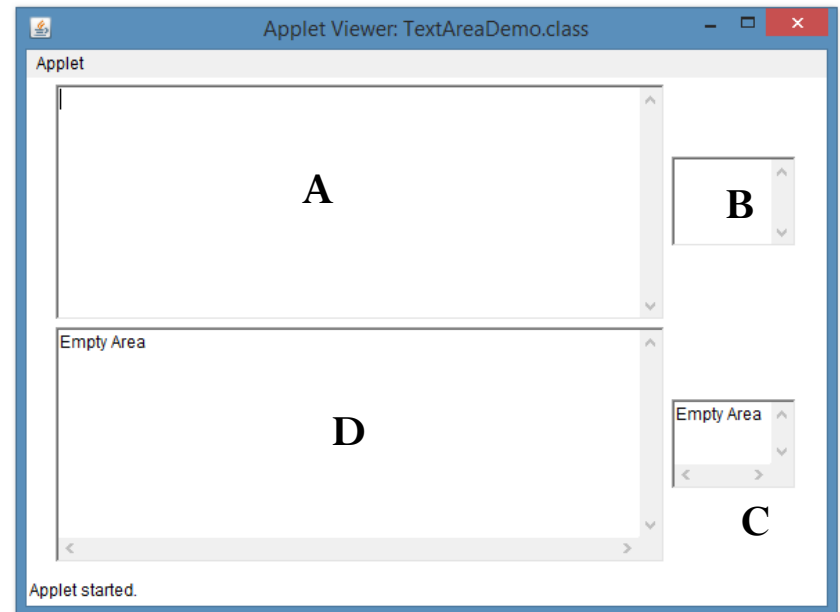
public class TextAreaDemo extends Applet
{
    public void init()
    {
        TextArea t1= new TextArea();           //A
        TextArea t2= new TextArea(3, 10);       //B
        TextArea t3= new TextArea("Empty Area "); //C
        TextArea t4= new TextArea("Empty Area", 3, 10); //D

        //add TextArea to applet window

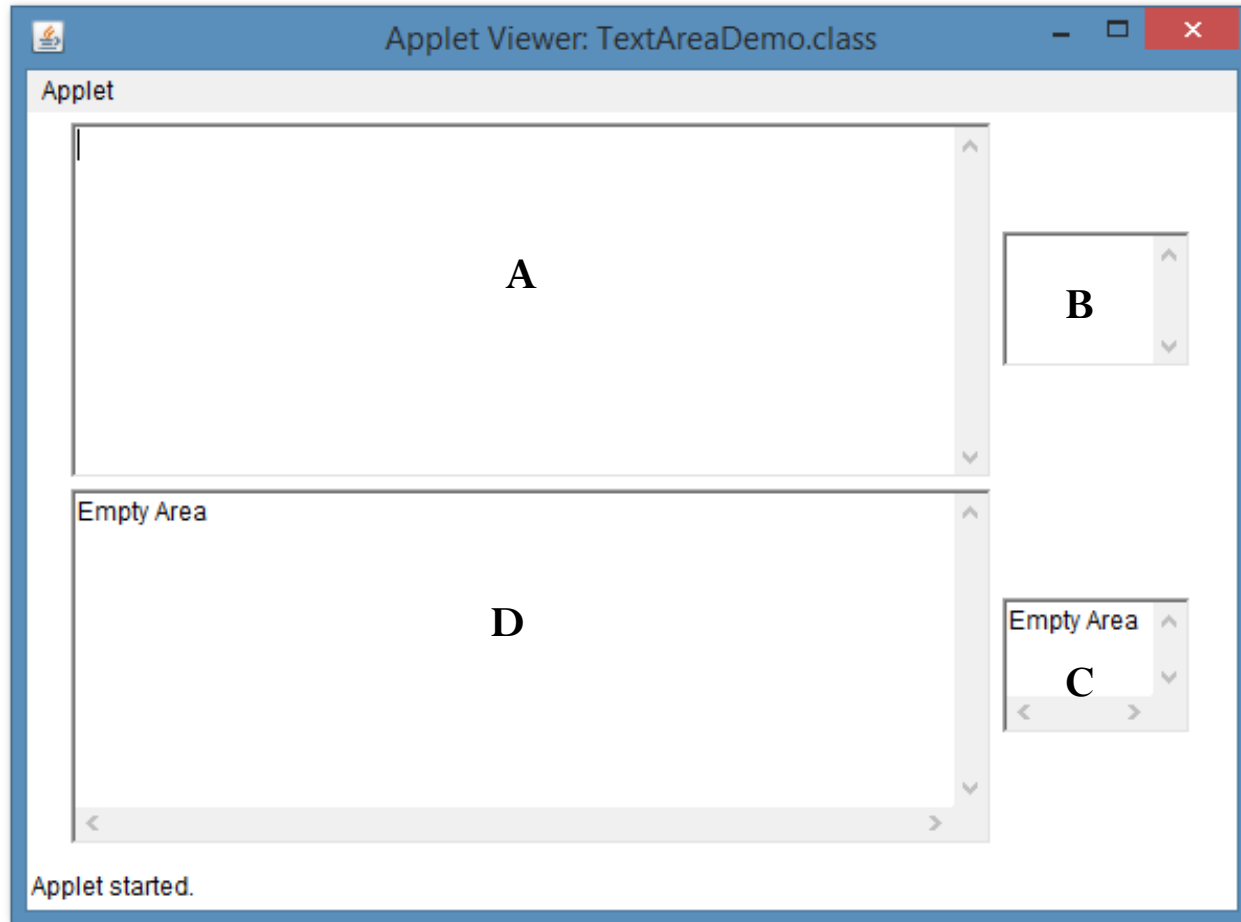
        add(t1);
        add(t2);
        add(t3);
        add(t4);
    }
}

```

Output -



Program - TextField



ut -

1.3 AWT Control – Checkbox, CheckboxGroup

5. CheckBox

1. It is used for selection of option.
2. It is used to turn an option ON (true) or OFF (false).
3. It consists of a small box that can either contains a check mark or not.
4. Clicking on a Checkbox changes its state from “On” to “Off” or from “Off” to “On”.

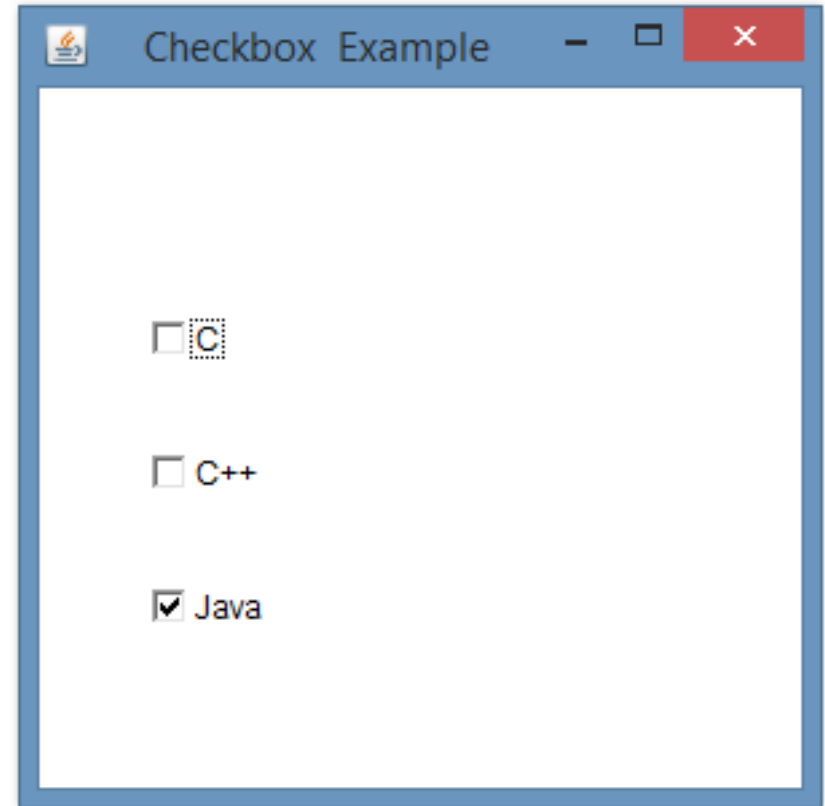
Sr	Constructors	Description
1	Checkbox()	Creates a checkbox with an empty string for its label.
2	Checkbox(String label)	Creates a checkbox with the specified label.
3	Checkbox(String label, boolean state)	Creates a check box with the specified label and sets the specified state.
4	Checkbox(String label, boolean state, CheckboxGroup group)	Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.

Program - Checkbox

```
import java.awt.*;

public class CheckboxEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame("Checkbox Example");
        Checkbox c1 = new Checkbox ("C");
        c1.setBounds(50, 100, 50,50);
        Checkbox c2 = new Checkbox ("C++");
        c2.setBounds(50, 150, 50,50);
        Checkbox c3 = new Checkbox ("Java", true);
        c3.setBounds(50, 200, 50,50);
        f.add(c1);
        f.add(c2);
        f.add(c3);
        f.setSize(300, 300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output -



6. CheckboxGroup (Radio Button)

1. It is used to group the set of Check box.
2. User can select one or more checkboxes at a time.
3. At a time only one check box button is allowed to be in “on” state and remaining check box button in “off” state.
4. These check boxes are called as Radio Button because here only one option will be selected.
5. AWT does not have separate class for creating Radio button. For this we create checkbox and put them in one group.
6. Firstly, define the group to which the checkboxes will belong and then specify the group when the checkboxes are constructed.

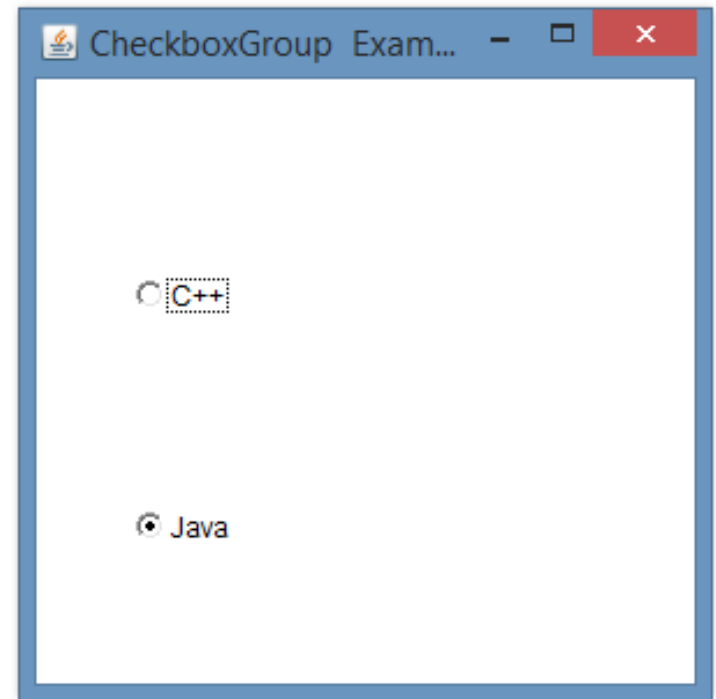
Sr	Constructor	Description
1	CheckboxGroup()	Creates a new instance of CheckboxGroup.

Program - CheckboxGroup

```
import java.awt.*;

public class CheckboxGroupEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame("CheckboxGroup Example");
        CheckboxGroup cbg = new CheckboxGroup ( );
        Checkbox c1 = new Checkbox ("C++",cbg,false);
        c1.setBounds(50, 100, 50,50);
        Checkbox c2 = new Checkbox ("Java",cbg, true);
        c2.setBounds(50, 200, 50,50);
        f.add(c1);
        f.add(c2);
        f.setSize(300, 300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output -



1.3 AWT Control – Choice, List and Scrollbar

7. Choice

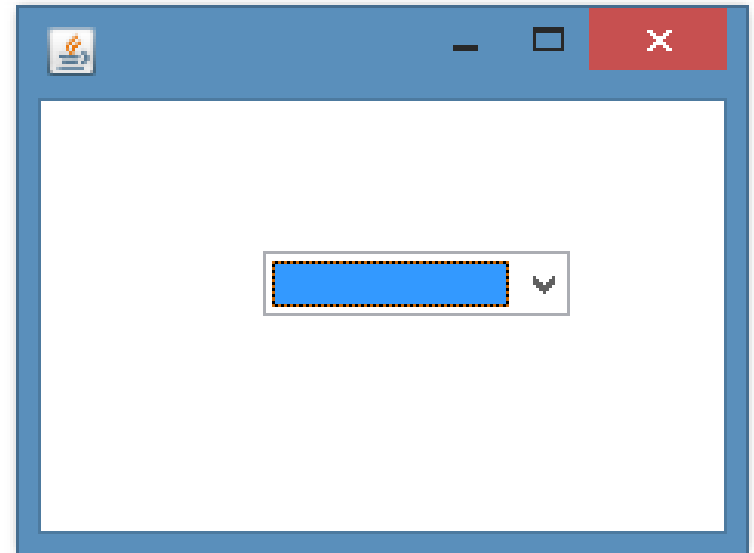
1. The Choice class is used to create a drop-down (pop-up) list of items which enables user to choose item from it.
2. Choice selected by user is shown on the top of a menu.
3. When a Choice component is inactive, it takes up only enough space to show the currently selected item.
4. When the user clicks on it, the whole list of choices pops up, and a new selection can be made.
5. Each item in the list is a string that appears as a left justified label in the order it is added to the Choice object.

Sr	Constructor	Description
1	Choice()	Creates an empty choice box.

Program - Choice

Output -

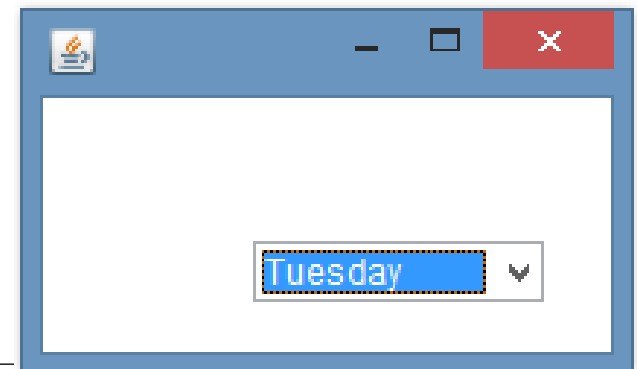
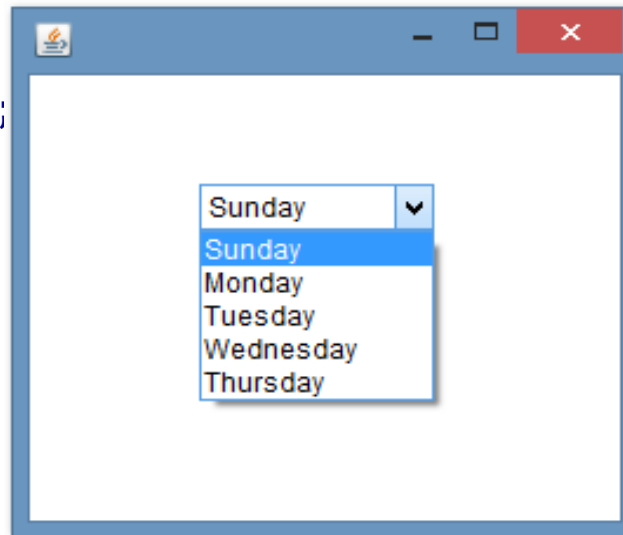
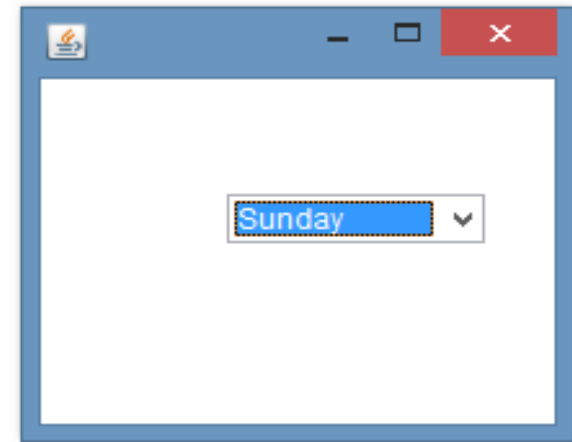
```
import java.awt.*;  
  
public class ChoiceEx  
{  
    public static void main(String args[ ])   
    {  
        Frame f = new Frame( );  
        Choice c = new Choice( );  
        c.setBounds(80, 80, 100,100);  
        f.add(c);  
        f.setSize(300, 300);  
        f.setLayout(null);  
        f.setVisible(true);  
    }  
}
```



Program - Choice

Output -

```
import java.awt.*;  
  
public class ChoiceEx  
{  
    public static void main(String args[ ])   
    {  
        Frame f = new Frame( );  
        Choice c = new Choice( );  
        c.setBounds(80, 80, 100,100);  
        c.add("Sunday");  
        c.add("Monday");  
        c.add("Tuesday");  
        c.add("Wednesday");  
        c.add("Thursday");  
        f.add(c);  
        f.setSize(300, 300);  
        f.setLayout(null);  
        f.setVisible(true);  
    }  
}
```



8. List

1. The List represents a list of text items.
2. User may select one or more items.
3. By the help of list, user can choose either one item or multiple item.
4. Unlike the Choice object, which shows only the single selected item in the menu, a List object can be constructed to show any number of choices in the visible Window.
5. It can also be created to allow multiple selections.

8. List

1. Constructors

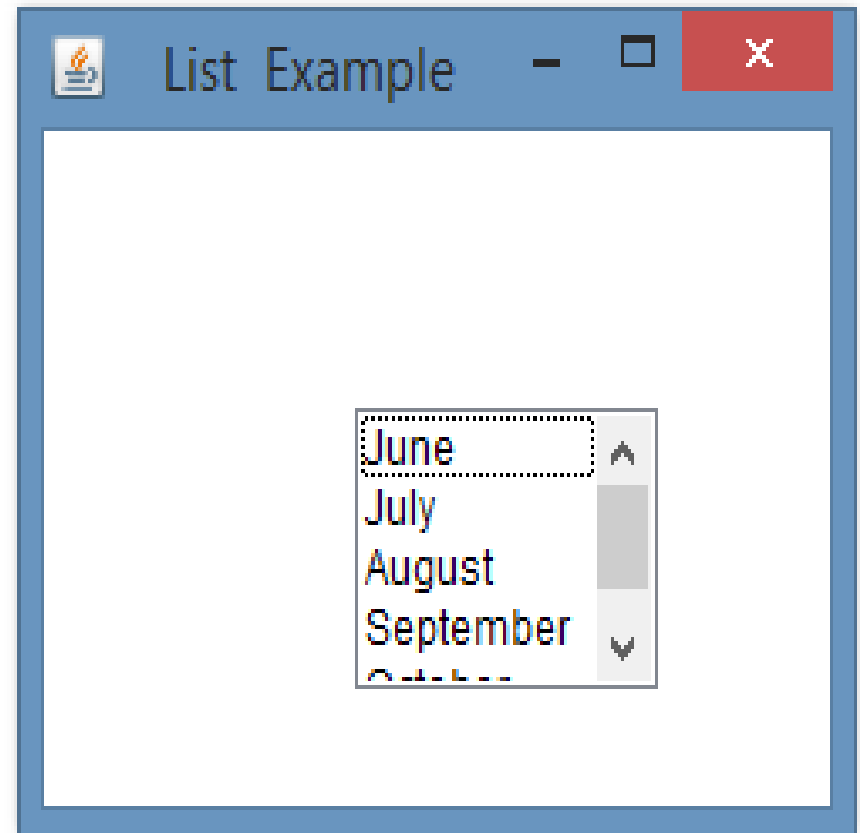
Sr. No	Constructors	Description
1	List()	Creates a new scrolling list. Creates a list control and allow only one item to be selected at one time.
2	List(int rows)	Creates a new scrolling list initialized with the specified number of visible lines.
3	List (int rows, boolean multipleMode)	Creates a new scrolling list initialized to display the specified number of rows.
		Creates a list with rows entries of items in the list. If multipleMode is true – then the user to select two or more items at a time. If it is false - then only one item can be selected.

Program - List

```
import java.awt.*;

public class ListEx1
{
    public static void main(String args[ ])
    {
        Frame f = new Frame("List Example");
        List l1 = new List (5);
        l1.setBounds(100, 100, 90,70);
        l1.add("June");
        l1.add("July");
        l1.add("August");
        l1.add("September");
        l1.add("October");
        f.add(l1);
        f.setSize(300, 300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

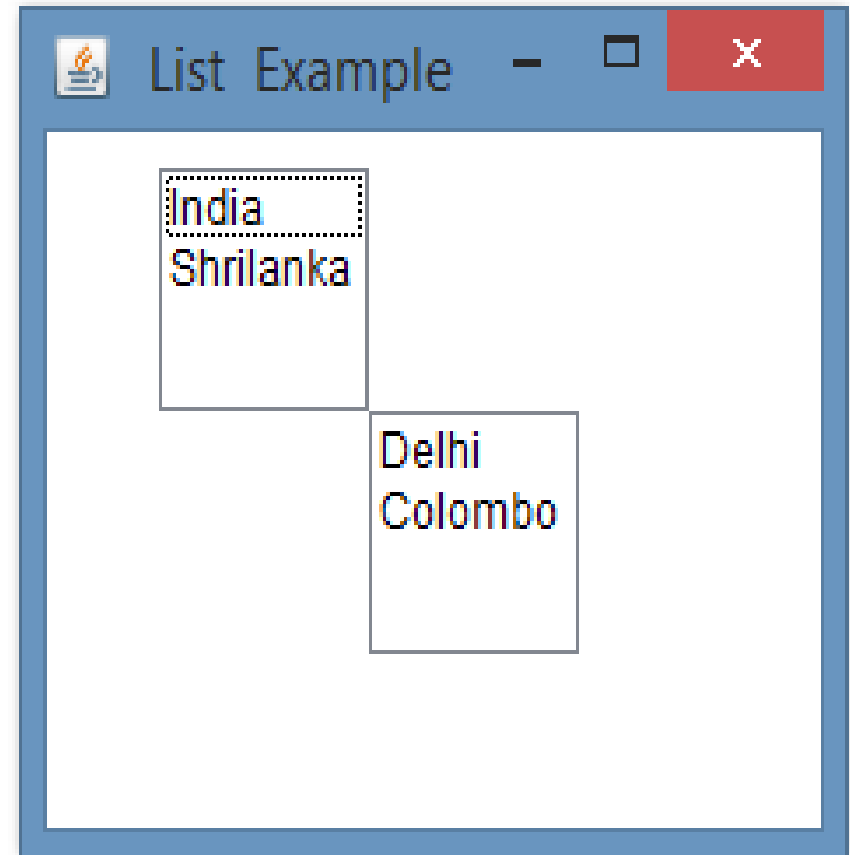
Output -



Program - List

```
import java.awt.*;  
  
public class ListEx  
{  
    public static void main(String args[ ])  
    {  
        Frame f = new Frame("List Example");  
        List country = new List (2);  
        country.setBounds(40, 40, 60,60);  
        List capital = new List (2);  
        capital.setBounds(100, 100, 60,60);  
        country.add("India");  
        country.add("Shrilanka");  
        capital.add("Delhi");  
        capital.add("Colombo");  
        f.add(country);  
        f.add( capital);  
        f.setSize(300, 300);  
        f.setLayout(null);  
        f.setVisible(true);  
    }  
}
```

Output -



9. Scrollbar

1. Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
2. It allows users to scroll the components.
3. It used to add Horizontal and Vertical scrollbar.
4. It allows us to see invisible number of rows and columns.
5. The current value of the scroll bar relative to its minimum and maximum values is indicated by the slider box (or thumb) for the scroll bar.

9. Scrollbar

1. Constructors

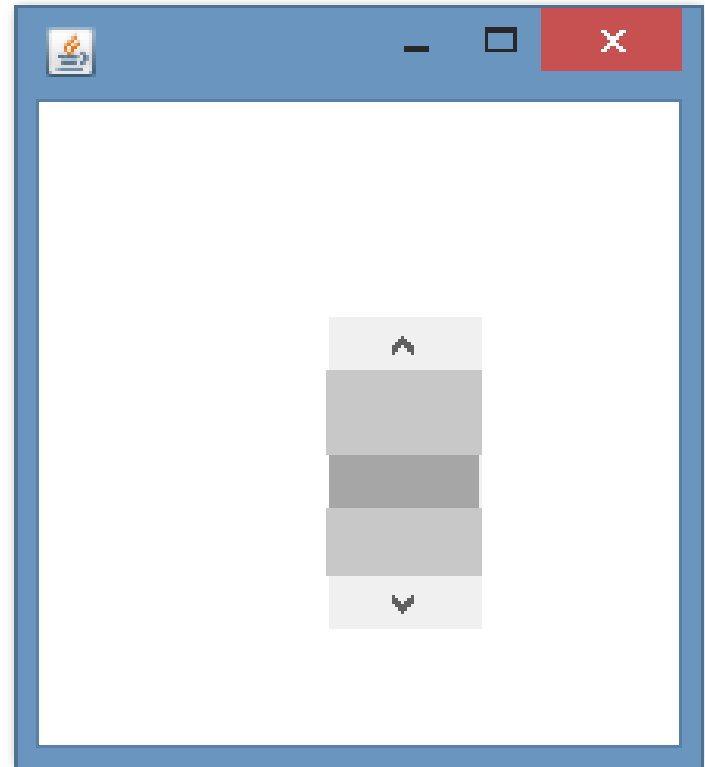
Sr. No	Constructors	Description
1	Scrollbar()	Constructs a new vertical scroll bar.
2	Scrollbar(int style)	creates a scroll bar with specified orientation, that is, horizontal or vertical. Scrollbar.VERTICAL and Scrollbar.HORIZONTAL
3	Scrollbar(int style, int initialvalue, int thumbsize, int min, int max)	specifies the initial value and the maximal value along with the thumb size.

Program - Scrollbar

```
import java.awt.*;

public class ScrollbarEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame( );
        Scrollbar s = new Scrollbar( );
        s.setBounds(100, 100, 50,100);
        f.add(s);
        f.setSize(300, 300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

Output -



Unit -1 Abstract Windowing Toolkit

1.4 Use of Layout Managers

1. Flow Layout

Use of Layout Managers

1. Layout means the arrangement of components within the container in a particular manner.
2. Layout manager is an interface that is implemented by all the classes of layout managers.
3. The java.awt package provides 5 layout managers classes –

1) Flow Layout

2) Grid Layout

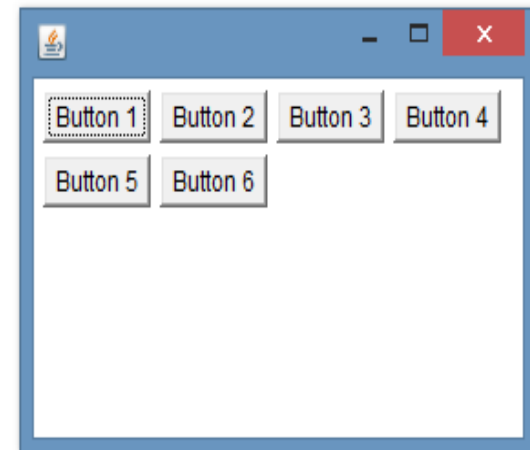
3) Border Layout

4) Card Layout

5) GridBag Layout

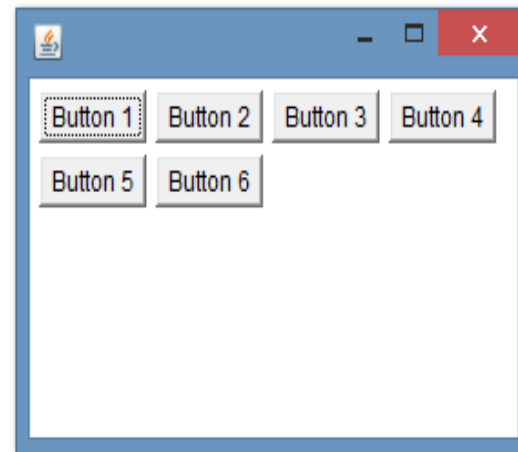
1. Flow Layout

1. The Flow Layout is used to arrange the components in a line, sequence, one after another (in a flow).
2. **The default layout of Applet and Panel is Flow Layout.**
3. Components are laid out from the Upper-left corner, left to right and top to bottom.
4. When no more components fit on a line, the next one appears on the next line.
5. A small space is left between each components, above and below, as well as left and right.



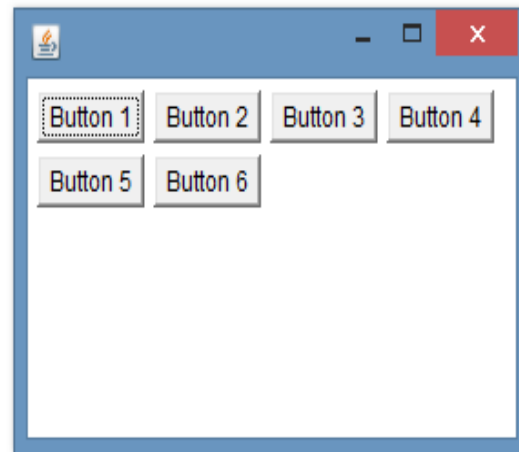
1. Flow Layout

Sr	Constructors	Description
1	FlowLayout()	Constructs a new FlowLayout with centered alignment. The Horizontal and Vertical gap will be 5 pixels.
2	FlowLayout(int align)	Constructs a new FlowLayout with given alignment. The Horizontal and Vertical gap will be 5 pixels.
3	FlowLayout(int align, int hgap, int vgap)	Constructs a new FlowLayout with given alignment, the given horizontal and vertical gap between the components.



1. Flow Layout

Sr. No	Methods	Description
1	<code>int getAlignment()</code>	Gets the Alignment of this layout.
2	<code>int getHgap ()</code>	Gets the horizontal spacing.
3	<code>int getVgap ()</code>	Gets the vertical spacing.
4	<code>int setAlignment(int align)</code>	Used to set the Alignment of this layout.
5	<code>int setHgap (int)</code>	Specifies the horizontal spacing.
6	<code>int setVgap (int)</code>	Specifies the verticals pacing.



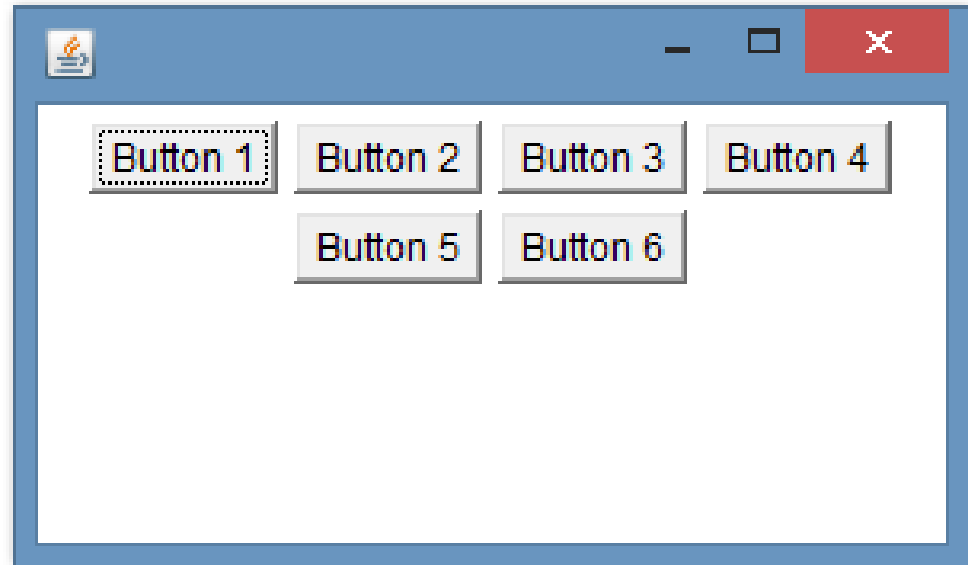
Program - FlowLayout

```
import java.awt.*;

public class FlowLayoutEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame();
        FlowLayout obj = new FlowLayout( );
        Button b1 = new Button ("Button 1");
        Button b2 = new Button ("Button 2");
        Button b3 = new Button ("Button 3");
        Button b4 = new Button ("Button 4");
        Button b5 = new Button ("Button 5");
        Button b6 = new Button ("Button 6");

        f.add(b1); f.add(b2); f.add(b3);
        f.add(b4); f.add(b5); f.add(b6);
        f.setLayout(obj);
        f.setSize(300, 300);
        f.setVisible(true);
    }
}
```

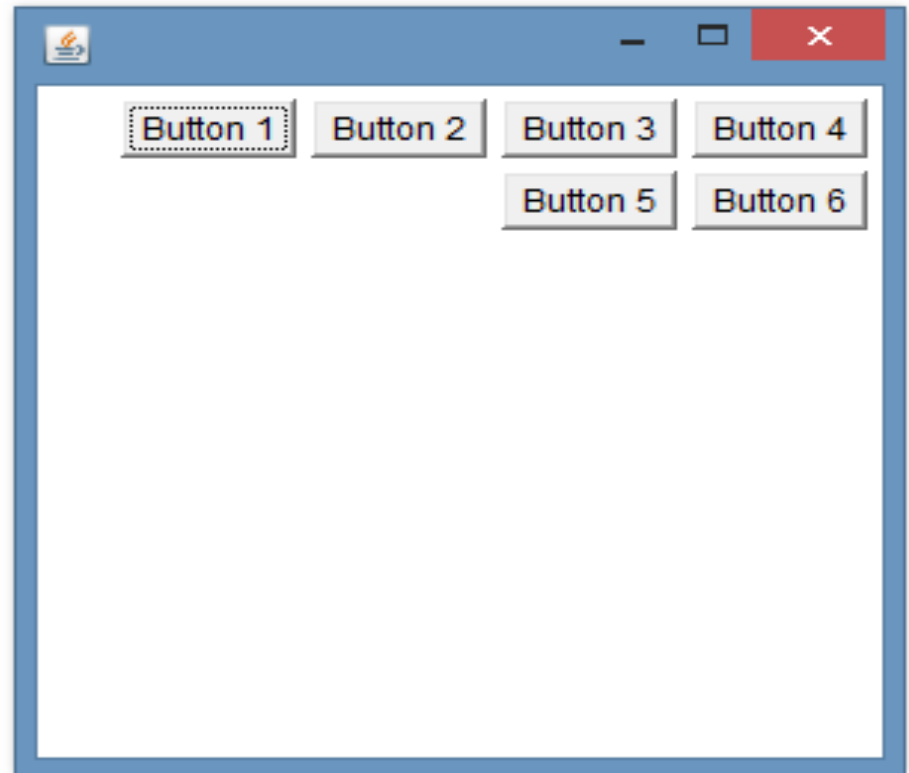
Output -



Program - FlowLayout

```
import java.awt.*;  
  
public class FlowLayoutEx  
{  
    public static void main(String args[ ])  
    {  
        Frame f = new Frame();  
        FlowLayout obj = new FlowLayout(FlowLayout.RIGHT);  
        Button b1 = new Button ("Button 1");  
        Button b2 = new Button ("Button 2");  
        Button b3 = new Button ("Button 3");  
        Button b4 = new Button ("Button 4");  
        Button b5 = new Button ("Button 5");  
        Button b6 = new Button ("Button 6");  
  
        f.add(b1); f.add(b2); f.add(b3);  
        f.add(b4); f.add(b5); f.add(b6);  
        f.setLayout(obj);  
        f.setSize(300, 300);  
        f.setVisible(true);  
    }  
}
```

Output -



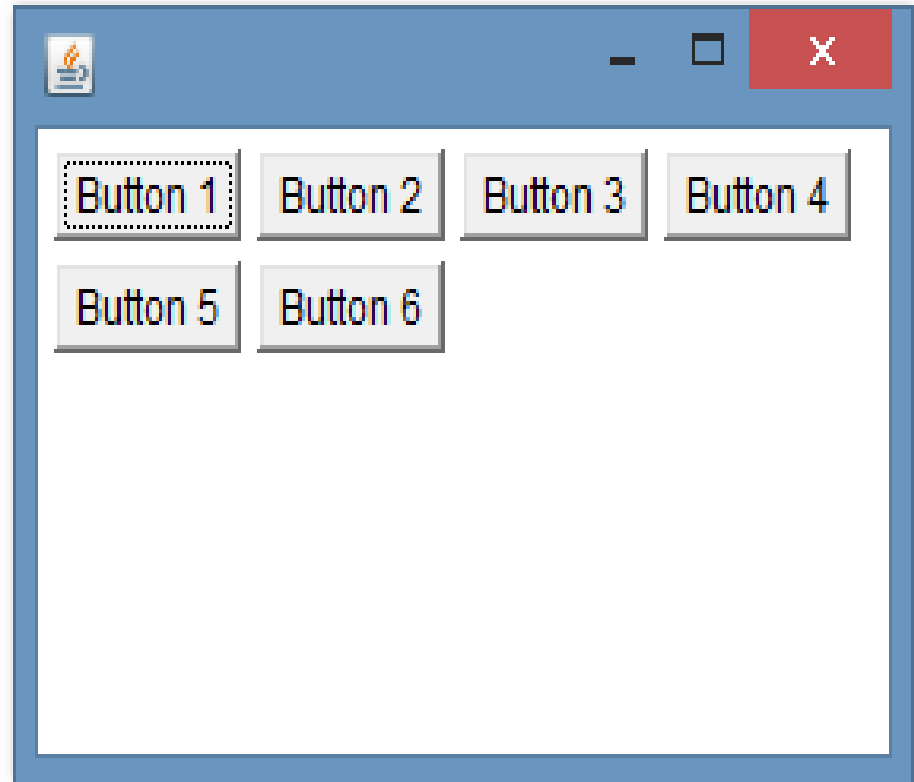
Program - FlowLayout

```
import java.awt.*;

public class FlowLayoutEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame();
        FlowLayout obj = new FlowLayout(FlowLayout.LEFT);
        Button b1 = new Button ("Button 1");
        Button b2 = new Button ("Button 2");
        Button b3 = new Button ("Button 3");
        Button b4 = new Button ("Button 4");
        Button b5 = new Button ("Button 5");
        Button b6 = new Button ("Button 6");

        f.add(b1); f.add(b2); f.add(b3);
        f.add(b4); f.add(b5); f.add(b6);
        f.setLayout(obj);
        f.setSize(300, 300);
        f.setVisible(true);
    }
}
```

Output -



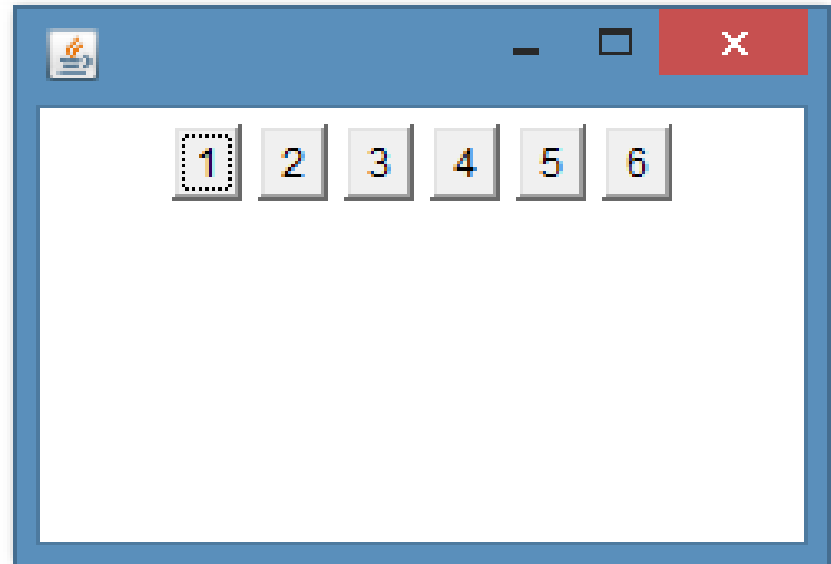
Program - FlowLayout

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

```
public class FlowLayoutEx
```

```
{  
    public static void main(String args[ ])  
    {  
        Frame f = new Frame( );  
        FlowLayout obj = new FlowLayout(FlowLayout.CENTER);  
        Button b1 = new Button ("1");  
        Button b2 = new Button ("2");  
        Button b3 = new Button ("3");  
        Button b4 = new Button ("4");  
        Button b5 = new Button ("5");  
        Button b6 = new Button ("6");  
  
        f.add(b1); f.add(b2); f.add(b3);  
        f.add(b4); f.add(b5); f.add(b6);  
        f.setLayout(obj);  
        f.setSize(300, 300);  
        f.setVisible(true);  
    }  
}
```

Output -



Program - FlowLayout

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

```
public class FlowLayoutEx
```

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

```
{  
    Frame f = new Frame();
```

```
    FlowLayout obj = new FlowLayout(FlowLayout.CENTER,25,25);
```

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

```
    Button b2 = new Button ("2");
```

```
    Button b3 = new Button ("3");
```

```
    Button b4 = new Button ("4");
```

```
    Button b5 = new Button ("5");
```

```
    Button b6 = new Button ("6");
```

```
    f.add(b1); f.add(b2); f.add(b3);
```

```
    f.add(b4); f.add(b5); f.add(b6);
```

```
    f.setLayout(obj);
```

```
    f.setSize(200, 150);
```

```
    f.setVisible(true);
```

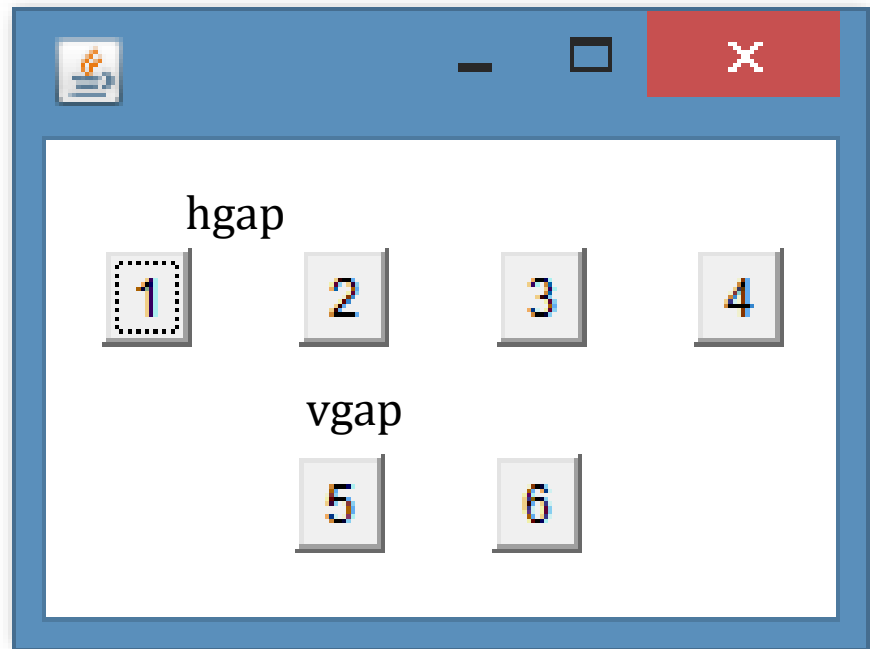
```
}
```

```
}
```

hgap

vgap

Output -



Unit -1 Abstract Windowing Toolkit

1.4 Use of Layout Managers

2.Grid Layout

2. Grid Layout

1. The Grid Layout is used to arrange the components in rectangular grid.
2. It is used to make a bunch of components equal in size and displays them in the requested number of rows and columns.
3. One component is displayed in each rectangle.

Sr	Constructors	Description
1	GridLayout()	Constructs a new Grid Layout with one column per component in a row.
2	GridLayout(int rows, int columns)	Constructs 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)	Constructs a Grid Layout with the given rows and columns along with horizontal and vertical gaps.

2. Grid Layout

Sr. No	Methods	Description
1	int getColumns()	Gets the number of Columns of this layout.
2	int getRows()	Gets the number of rows of this layout.
3	int getHgap ()	Gets the horizontal spacing.
4	int getVgap ()	Gets the vertical spacing.
5	int setColumns(int cols)	Sets the number of Columns in this layout at the specified value.
6	int setRows(int rows)	Sets the number of Rows in this layout at the specified value.
7	int setHgap (int	Specifies the horizontal spacing.
8	int setVgap (int)	Specifies the verticals pacing.

Program - 2. Grid Layout

```
import java.awt.*;

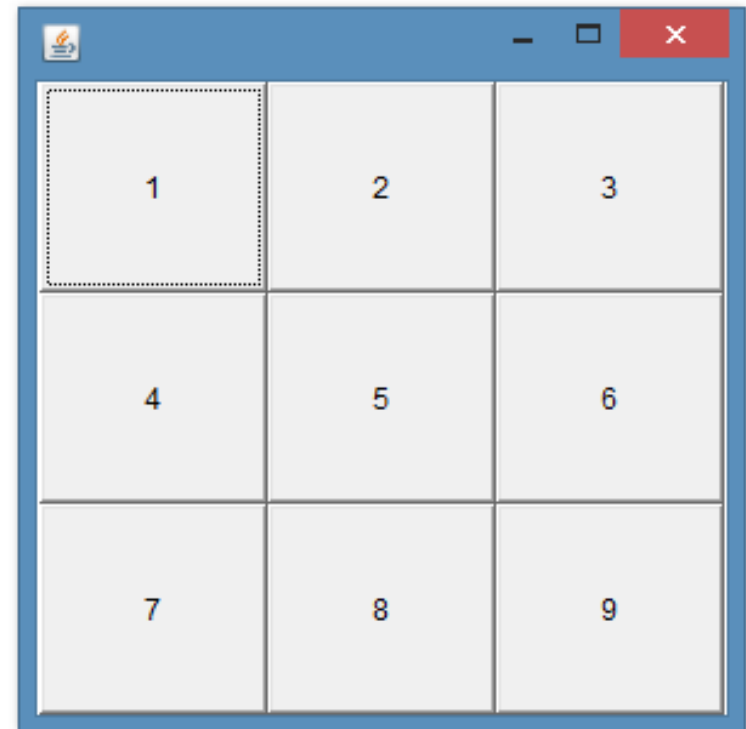
public class GridLayoutEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame();

        GridLayout obj = new GridLayout(3,3);

        Button b1 = new Button ("1");
        Button b2 = new Button ("2");
        Button b3 = new Button ("3");
        Button b4 = new Button ("4");
        Button b5 = new Button ("5");
        Button b6 = new Button ("6");
        Button b7 = new Button ("7");
        Button b8 = new Button ("8");
        Button b9 = new Button ("9");

        f.add(b1); f.add(b2); f.add(b3);
        f.add(b4); f.add(b5); f.add(b6);
        f.add(b7); f.add(b8); f.add(b9);
        f.setLayout(obj);
        f.setSize(500, 500);
        f.setVisible(true);
    }
}
```

Output -



Program - 2. Grid Layout

```
import java.awt.*;

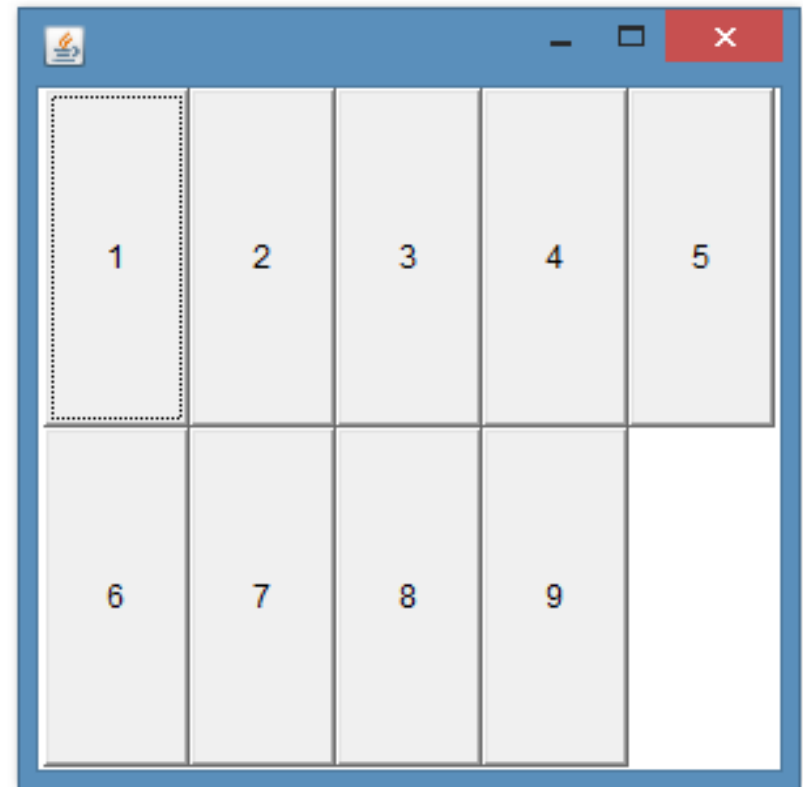
public class GridLayoutEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame();

        GridLayout obj = new GridLayout(2,4);

        Button b1 = new Button ("1");
        Button b2 = new Button ("2");
        Button b3 = new Button ("3");
        Button b4 = new Button ("4");
        Button b5 = new Button ("5");
        Button b6 = new Button ("6");
        Button b7 = new Button ("7");
        Button b8 = new Button ("8");
        Button b9 = new Button ("9");

        f.add(b1); f.add(b2); f.add(b3);
        f.add(b4); f.add(b5); f.add(b6);
        f.add(b7); f.add(b8); f.add(b9);
        f.setLayout(obj);
        f.setSize(500, 500);
        f.setVisible(true);
    }
}
```

Output -

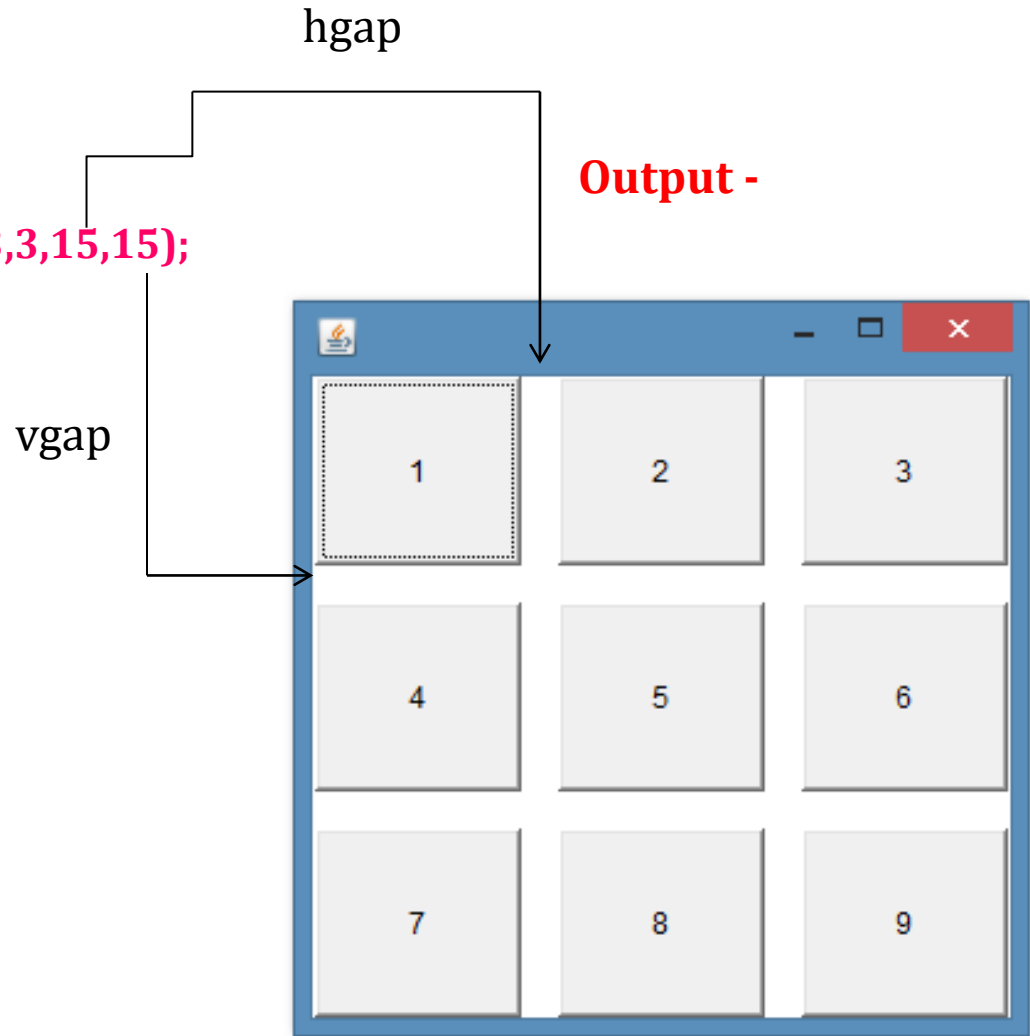


Program - 2. Grid Layout

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

```
public class GridLayoutEx
```

```
{  
    public static void main(String args[ ])   
    {  
        Frame f = new Frame();  
        GridLayout obj = new GridLayout(3,3,15,15);  
        Button b1 = new Button ("Button 1");  
        Button b2 = new Button ("Button 2");  
        Button b3 = new Button ("Button 3");  
        Button b4 = new Button ("Button 4");  
        Button b5 = new Button ("Button 5");  
        Button b6 = new Button ("Button 6");  
        Button b7 = new Button ("Button 7");  
        Button b8 = new Button ("Button 8");  
        Button b9 = new Button ("Button 9");  
  
        f.add(b1); f.add(b2); f.add(b3);  
        f.add(b4); f.add(b5); f.add(b6);  
        f.add(b7); f.add(b8); f.add(b9);  
        f.setLayout(obj);  
        f.setSize(500, 500);  
        f.setVisible(true);  
    }  
}
```



Program - 2. Grid Layout

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

```
public class GridLayoutEx
```

```
{
```

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

```
    {
```

```
        Frame f = new Frame();
```

```
        GridLayout obj = new GridLayout(3, 3, 45, 10);
```

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

```
        Button b2 = new Button ("2");
```

```
        Button b3 = new Button ("3");
```

```
        Button b4 = new Button ("4");
```

```
        Button b5 = new Button ("5");
```

```
        Button b6 = new Button ("6");
```

```
        Button b7 = new Button ("7");
```

```
        Button b8 = new Button ("8");
```

```
        Button b9 = new Button ("9");
```

```
        f.add(b1); f.add(b2); f.add(b3);
```

```
        f.add(b4); f.add(b5); f.add(b6);
```

```
        f.add(b7); f.add(b8); f.add(b9);
```

```
        f.setLayout(obj);
```

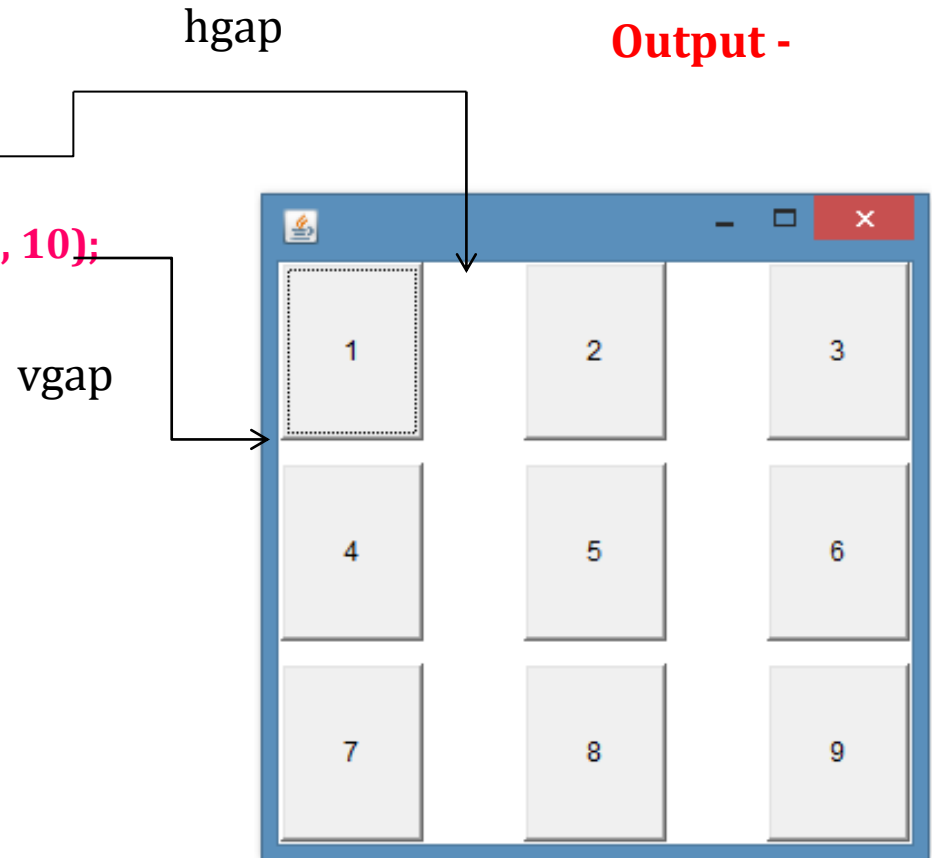
```
        f.setSize(500, 500);
```

```
        f.setVisible(true);
```

```
    }
```

```
}
```

Output -



Unit -1 Abstract Windowing Toolkit

1.4 Use of Layout Managers

3.Border Layout

3. Border Layout

1. It is used to arrange the components in five regions : **North, South, East, West and Center.**
2. Each region (area) may contain one component only.
3. It has four narrow , fixed-width components at the edges and one large area in the center.
4. The four sides are referred to as **North(Upper region), South(Lower region), East(Left region) and West(Right region).**
5. **The middle area is called the center (center region).**
6. **The default layout of Frame and Window is Border Layout.**

3. Border Layout

1. Constructors

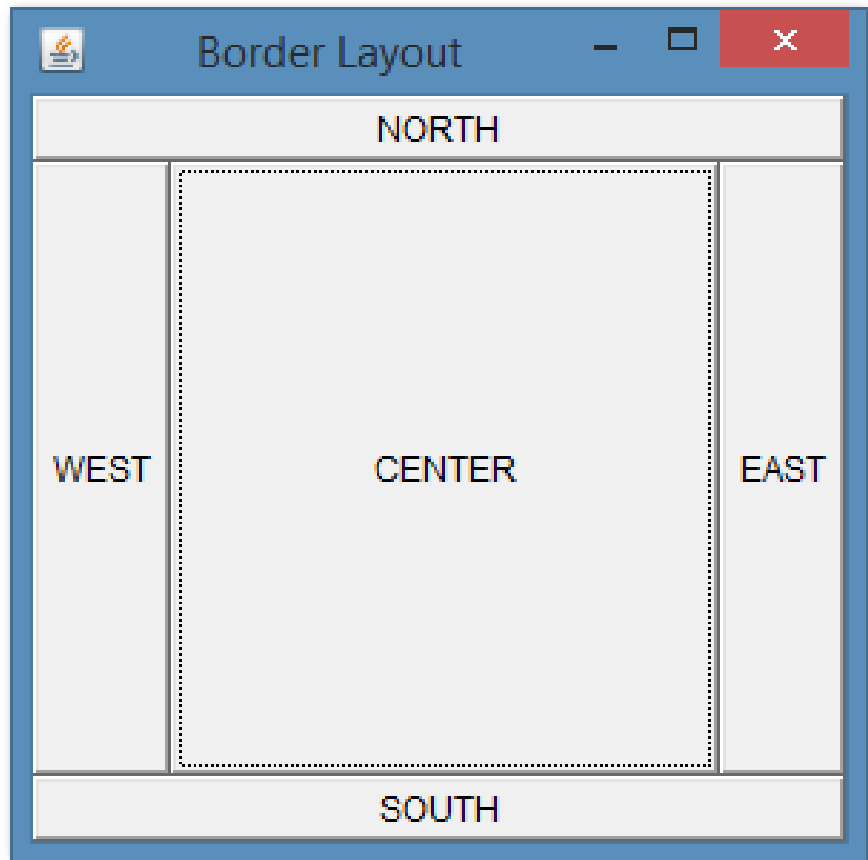
Sr. No	Constructors	Description
1	BorderLayout()	Constructs a new borderlayout with no gaps between the components.
2	BorderLayout(int hgap, int vgap)	Constructs a border layout with the specified gaps between components.

Sr. No	Methods	Description
1	int getHgap ()	Gets the horizontal spacing.
2	int getVgap ()	Gets the vertical spacing.
3	int setHgap (int hgap)	Specifies the horizontal spacing.
4	int setVgap (int vgap)	Specifies the vertical spacing.

Program - 3. Border Layout

```
import java.awt.*;  
public class BorderLayoutEx  
{  
    public static void main(String args[ ])  
    {  
        Frame f = new Frame("Border Layout");  
        BorderLayout obj=new BorderLayout();  
        Button b1 = new Button ("NORTH");  
        Button b2 = new Button ("SOUTH");  
        Button b3 = new Button ("EAST");  
        Button b4 = new Button ("WEST");  
        Button b5 = new Button ("CENTER");  
  
        f.add(b1, BorderLayout.NORTH);  
        f.add(b2,BorderLayout.SOUTH);  
        f.add(b3,BorderLayout.EAST);  
        f.add(b4,BorderLayout.WEST);  
        f.add(b5,BorderLayout.CENTER);  
  
        f.setSize(300,300);  
        f.setVisible(true);  
    }  
}
```

Output -



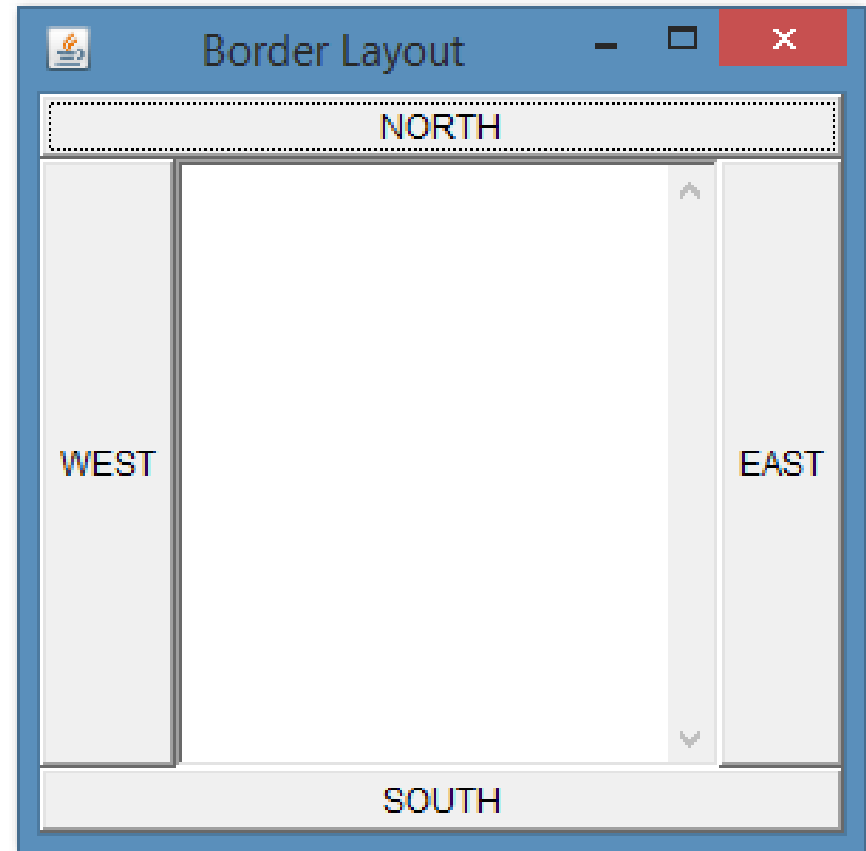
Program - 3. Border Layout

```
import java.awt.*;
public class BorderLayoutEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame("Border Layout");
        BorderLayout obj=new BorderLayout();
        Button b1 = new Button ("NORTH");
        Button b2 = new Button ("SOUTH");
        Button b3 = new Button ("EAST");
        Button b4 = new Button ("WEST");
        TextArea t = new TextArea(4,4);

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

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

Output -

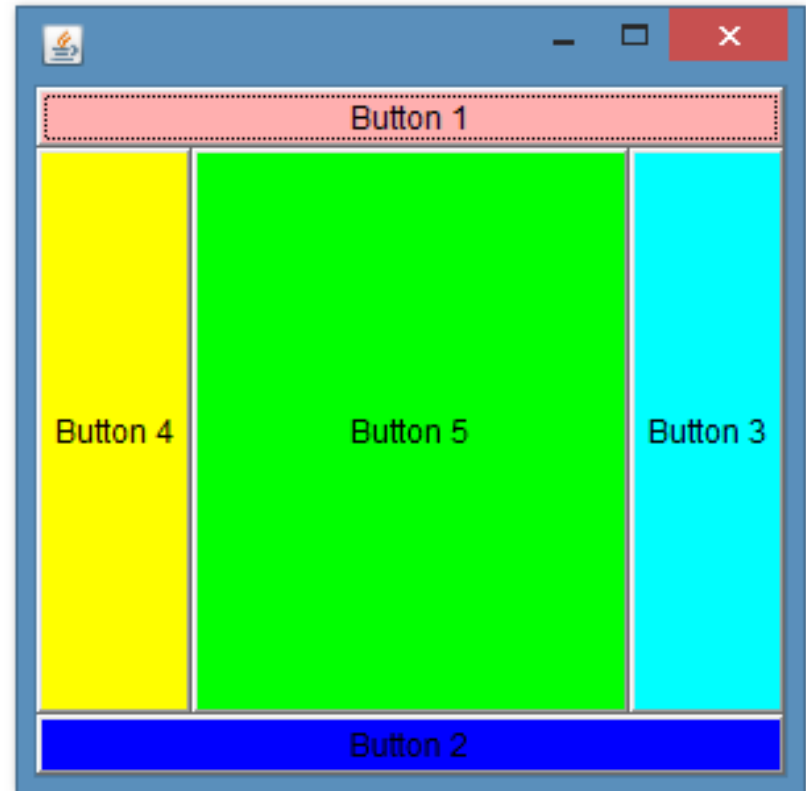


```

import java.awt.*;
public class BorderLayoutEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame();
        f.setLayout(new BorderLayout ( ));
        Button b1 = new Button ("Button 1");
        b1.setBackground(Color.pink);
        Button b2 = new Button ("Button 2");
        b2.setBackground(Color.blue);
        Button b3 = new Button ("Button 3");
        b3.setBackground(Color.cyan);
        Button b4 = new Button ("Button 4");
        b4.setBackground(Color.yellow);
        Button b5 = new Button ("Button 5");
        b5.setBackground(Color.green);
        f.add(b1, BorderLayout.NORTH);
        f.add(b2, BorderLayout.SOUTH);
        f.add(b3, BorderLayout.EAST);
        f.add(b4, BorderLayout.WEST);
        f.add(b5, BorderLayout.CENTER);
        f.setSize(300,300);
        f.setVisible(true);
    }
}

```

Output -



```

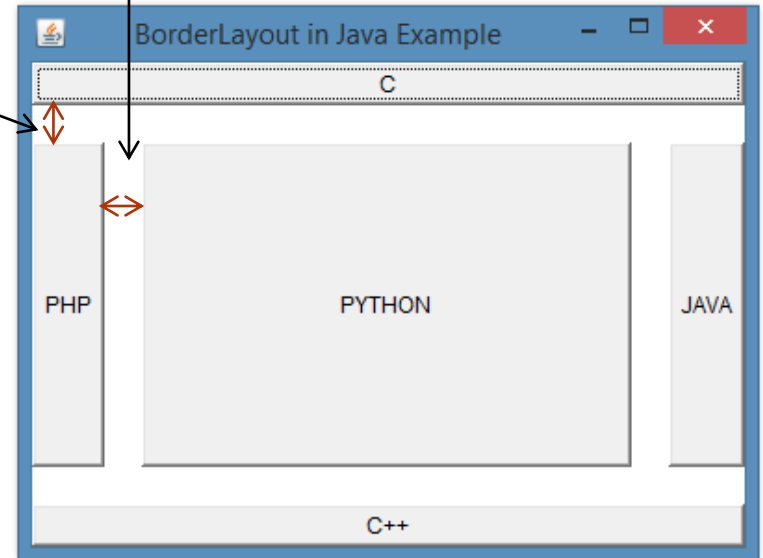
import java.awt.*;
class BorderLayoutEx extends Frame
{
    BorderLayoutEx()
    {
        setLayout(new BorderLayout(20,20));

        add(new Button("C"),BorderLayout.NORTH);
        add(new Button("C++"),BorderLayout.SOUTH);
        add(new Button("JAVA"),BorderLayout.EAST);
        add(new Button("PHP"),BorderLayout.WEST);
        add(new Button("PYTHON"),BorderLayout.CENTER);
    }
}

class BorderLayoutEx1
{
    public static void main(String args[])
    {
        BorderLayoutEx frame = new BorderLayoutEx();
        frame.setTitle("BorderLayout in Java Example");
        frame.setSize(400,300);
        frame.setVisible(true);
    }
}

```

Output -



```

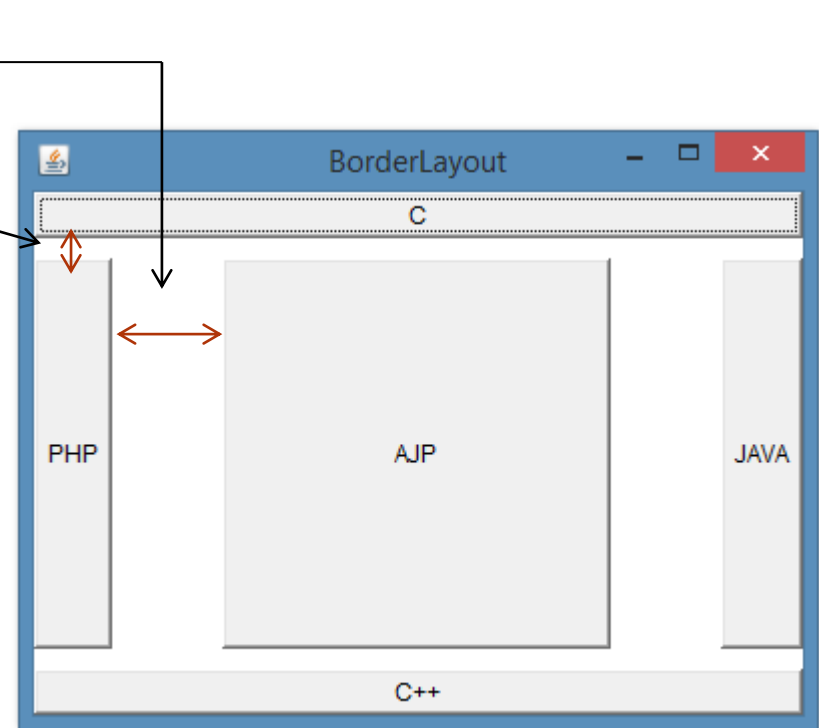
import java.awt.*;
class BorderLayoutEx extends Frame
{
    BorderLayoutEx()
    {
        setLayout(new BorderLayout(55,10));

        add(new Button("C"),BorderLayout.NORTH);
        add(new Button("C++"),BorderLayout.SOUTH);
        add(new Button("JAVA"),BorderLayout.EAST);
        add(new Button("PHP"),BorderLayout.WEST);
        add(new Button("AJP"),BorderLayout.CENTER);
    }
}

class BorderLayoutDemo
{
    public static void main(String args[])
    {
        BorderLayoutEx f = new BorderLayoutEx();
        f.setTitle("BorderLayout");
        f.setSize(400,300);
        f.setVisible(true);
    }
}

```

Output -



Unit -1 Abstract Windowing Toolkit

1.4 Use of Layout Managers

4. Card Layout

4. Card Layout

1. The CardLayout class manages the components in such a way that **only one component is visible at a time.**
2. It treats each component as a card in the container.
3. The first component added to a CardLayout object is the visible component when the container is first displayed.
4. CardLayout manager's usage is very need-based and quite different from others where other layout managers try to display all the components added to the container.
5. Example is MS Excel workbook in which the selected sheet is opened by clicking over sheet tab.

4. Card Layout

1. Constructors

Sr. No	Constructors	Description
1	CardLayout()	creates a card layout with zero horizontal and vertical gap.
2	CardLayout(int hgap, int vgap)	creates a card layout with zero horizontal and vertical gap.

4. Card Layout

Sr. No	Methods	Description
1	public void next (Container parent)	is used to flip to the next card of the given container.
2	public void previous (Container parent)	is used to flip to the previous card of the given container.
3	public void first (Container parent)	is used to flip to the first card of the given container.
4	public void last (Container parent)	is used to flip to the last card of the given container.
5	public void next(Container parent)	is used to flip to the next card of the given container.
6	public void shoe(Container parent, String name)	is used to flip to the specified card of the given name.

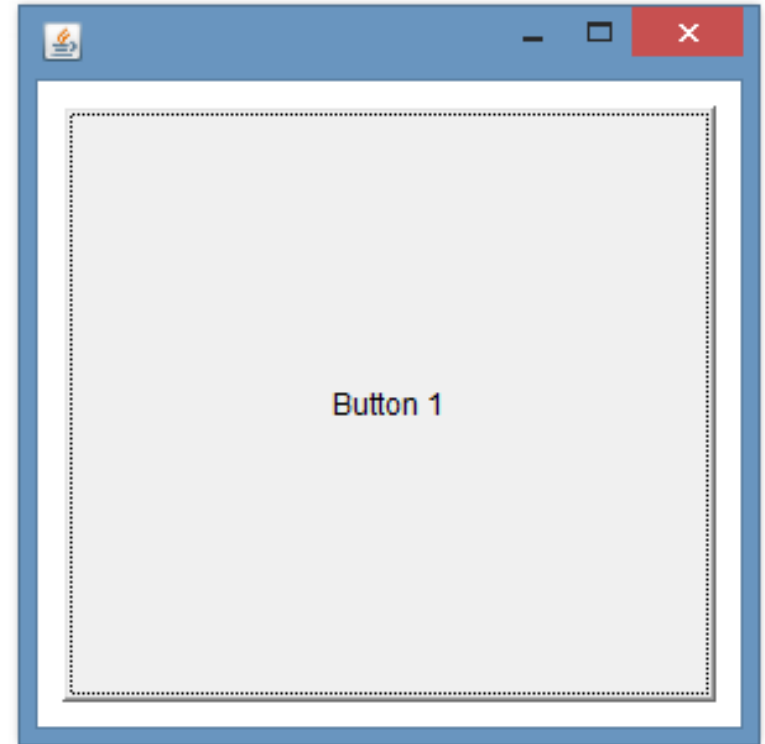
Program - 4. Card Layout

```
import java.awt.*;

public class CardLayoutEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame( );
        CardLayout obj = new CardLayout(10,10 );
        f.setLayout(obj);
        Button b1 = new Button ("Button 1");
        Button b2 = new Button ("Button 2");
        Button b3 = new Button ("Button 3");

        f.add(b1);
        f.add(b2);
        f.add(b3);
        f.setSize(300,300);
        f.setVisible(true);
    }
}
```

Output -



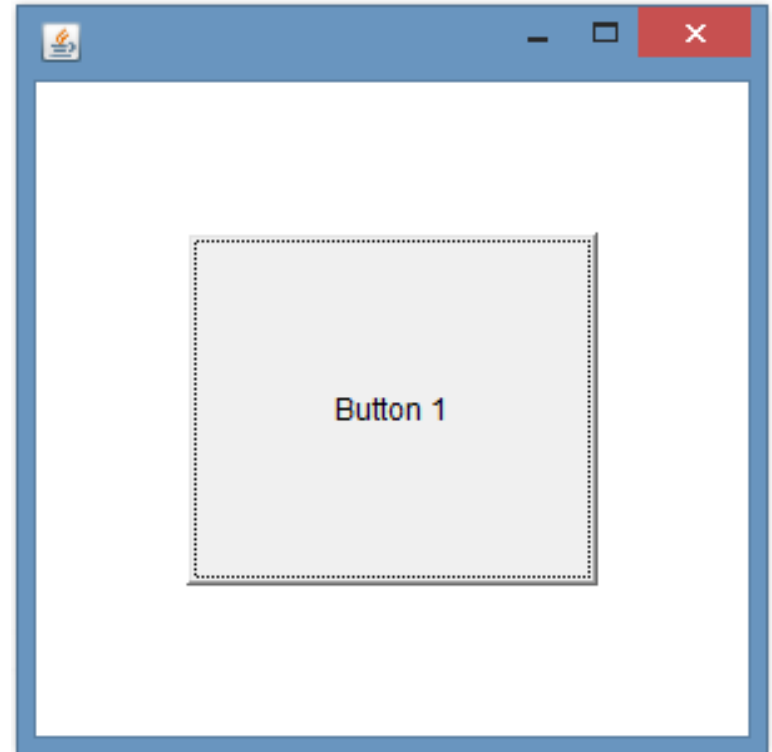
Program - 4. Card Layout

```
import java.awt.*;

public class CardLayoutEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame( );
        CardLayout obj = new CardLayout(60,60 );
        f.setLayout(obj);
        Button b1 = new Button ("Button 1");
        Button b2 = new Button ("Button 2");
        Button b3 = new Button ("Button 3");

        f.add(b1);
        f.add(b2);
        f.add(b3);
        f.setSize(300,300);
        f.setVisible(true);
    }
}
```

Output -



Unit -1 Abstract Windowing Toolkit

1.4 Use of Layout Managers

Type 5. GridBag Layout

5. GridBag Layout

1. GridBagLayout is one of the most flexible and complex layout managers the Java platform provides.
2. A GridBagLayout places components in a grid of rows and columns, allowing specified components to span multiple rows or columns.
3. Not all rows necessarily have the same height. Similarly, not all columns have the same width.
4. Essentially, GridBagLayout places components in rectangles(cells) in a grid, then uses the components preferred sizes to determine how big the cells should be.
5. The resizing behavior is based on weights the program assigns to individual components in the GridBagLayout.

5. GridBag Layout

1. Constructors

Sr. No	Constructors	Description
1	GridBagLayout()	Creates a grid bag layout manager.

5. GridBag Layout

Sr	Variable	Description
1	gridx and gridy	Specify the row and column at the upper left of the component. The leftmost column has address gridx=0 and the top row has address gridy=0.
2	gridwidth, gridheight	Define how many cells will occupy component (height and width).Specify the number of columns(for gridwidth) or rows(for gridheight) in the component's display area. The default value is 1.
3	fill	Used when the component's display area is larger than the component's requested size to determine whether and how to resize the component. Valid value of GridBagConstraints 1.NONE-default, 2.HORIZONTAL -expanded horizontally 3.VERTICAL -expanded vertically 4.BOTH.
4	ipadx, ipady	Specifies the internal padding; how much to add to the size of the component. The default value is 0.
5	weightx, weighty	Weights are used to determine how to distribute space among columns(weightx) and among rows(weighty); this is important for specifying resizing behavior.

5. GridBag Layout

Sr. No	Methods	Description
1	<code>addLayoutComponent(Component comp, Object constraints)</code>	It adds specified component to the layout, using the specified constraints object.
2	<code>removeLayoutComponent(Component cmp)</code>	Removes the specified component from this layout.
3	<code>getLayoutAlignmentX(Container p)</code>	Returns the alignment along the x-axis.
4	<code>getLayoutAlignmentY(Container p)</code>	Returns the alignment along the y-axis.
5	<code>getConstraints(Component cmp)</code>	Gets the constraints for the specified component.

```

import java.awt.*;
public class GridBagLayoutEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame( );
        GridBagLayout obj = new GridBagLayout( );
        GridBagConstraints gbc = new GridBagConstraints( );
        f.setLayout(obj);

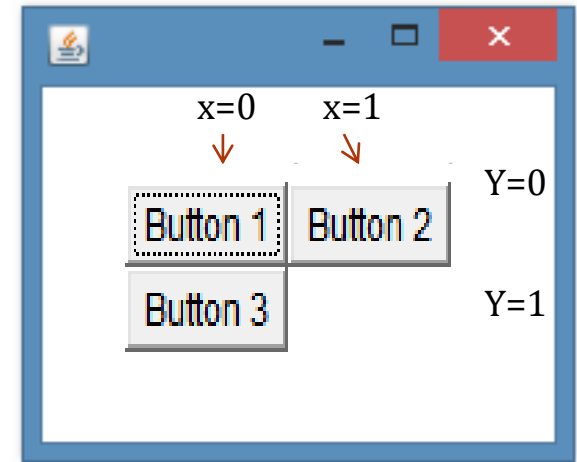
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridx=0;
        gbc.gridy=0;
        Button b1 = new Button ("Button 1");
        f.add(b1,gbc);

        gbc.gridx=1;
        gbc.gridy=0;
        Button b2 = new Button ("Button 2");
        f.add(b2,gbc);

        gbc.gridx=0;
        gbc.gridy=1;
        Button b3 = new Button ("Button 3");
        f.add(b3,gbc);

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

```



```

gbc.fill =
GridBagConstraints.HORIZONTAL;
    gbc.gridx=0;
    gbc.gridy=1;
    Button b3 = new Button ("Button 3");
    f.add(b3,gbc);

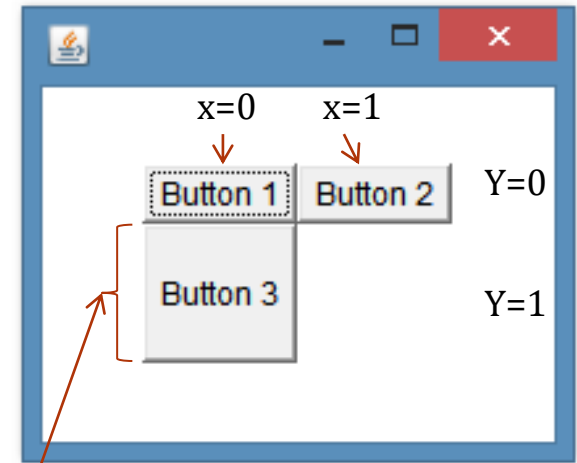
    f.setSize(300,300);
    f.setVisible(true);
}
}

```

```

import java.awt.*;
public class GridBagLayoutEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame();
        GridBagLayout obj = new GridBagLayout();
        GridBagConstraints gbc = new GridBagConstraints();
        f.setLayout(obj);

```



```

        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridx=0;
        gbc.gridy=0;
        Button b1 = new Button ("Button 1");
        f.add(b1,gbc);

        gbc.gridx=1;
        gbc.gridy=0;
        Button b2 = new Button ("Button 2");
        f.add(b2,gbc);

```

```

        gbc.fill =
        GridBagConstraints.HORIZONTAL;
        gbc.ipady=30;
        gbc.gridx=0;
        gbc.gridy=1;
        Button b3 = new Button ("Button 3");
        f.add(b3,gbc);

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

```

```
import java.awt.*;
public class GridBagLayoutEx
{
    public static void main(String args[ ])
    {
```

```
        Frame f = new Frame();
```

```
        GridBagLayout obj = new GridBagLayout( );
```

```
        GridBagConstraints gbc = new GridBagConstraints( );
```

```
        f.setLayout(obj);
```

```
        gbc.fill = GridBagConstraints.HORIZONTAL;
```

```
        gbc.gridx=0;
```

```
        gbc.gridy=0;
```

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

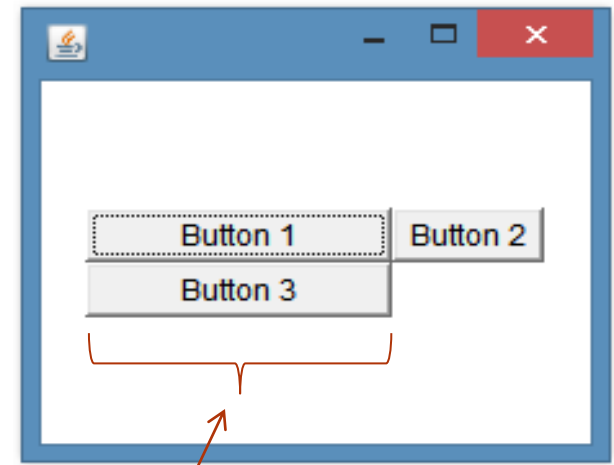
```
        f.add(b1,gbc);
```

```
        gbc.gridx=1;
```

```
        gbc.gridy=0;
```

```
        Button b2 = new Button ("Button 2");
```

```
        f.add(b2,gbc);
```



```
        gbc.fill =
```

```
        GridBagConstraints.HORIZONTAL;
```

```
        gbc.ipadx=60;
```

```
        gbc.gridx=0;
```

```
        gbc.gridy=1;
```

```
        Button b3 = new Button ("Button 3");
```

```
        f.add(b3,gbc);
```

```
        f.setSize(300,300);
```

```
        f.setVisible(true);
```

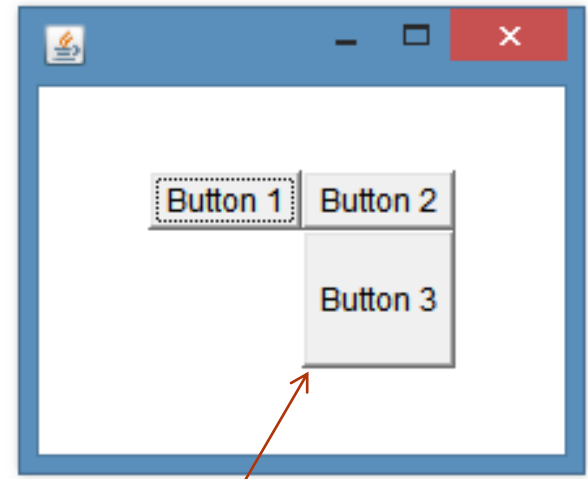
```
    }
```

```
}
```

```

import java.awt.*;
public class GridBagLayoutEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame();
        GridBagLayout obj = new GridBagLayout();
        GridBagConstraints gbc = new GridBagConstraints();
        f.setLayout(obj);

```



```

        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridx=0;
        gbc.gridy=0;
        Button b1 = new Button ("Button 1");
        f.add(b1,gbc);

        gbc.gridx=1;
        gbc.gridy=0;
        Button b2 = new Button ("Button 2");
        f.add(b2,gbc);

```

```

        gbc.fill =
        GridBagConstraints.HORIZONTAL;
        gbc.ipady=30;
        gbc.gridx=1;
        gbc.gridy=1;
        Button b3 = new Button ("Button 3");
        f.add(b3,gbc);

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

```

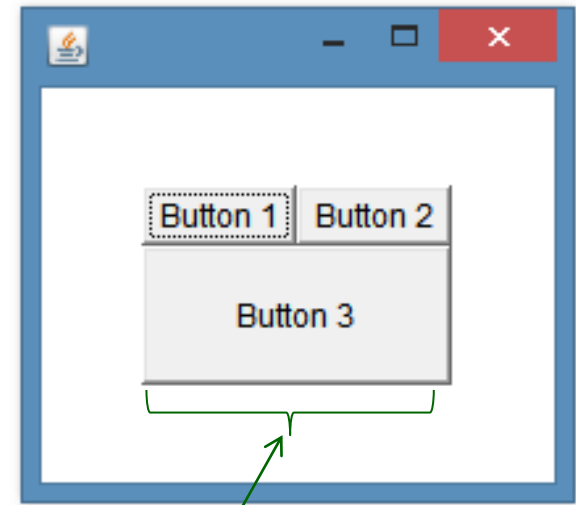
```

import java.awt.*;
public class GridBagLayoutEx
{
    public static void main(String args[ ])
    {
        Frame f = new Frame();
        GridBagLayout obj = new GridBagLayout( );
        GridBagConstraints gbc = new GridBagConstraints( );
        f.setLayout(obj);

        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridx=0;
        gbc.gridy=0;
        Button b1 = new Button ("Button 1");
        f.add(b1,gbc);

        gbc.gridx=1;
        gbc.gridy=0;
        Button b2 = new Button ("Button 2");
        f.add(b2,gbc);
    }
}

```



```

gbc.fill =
GridBagConstraints.HORIZONTAL;
gbc.gridwidth=2;
gbc.ipady=30;
gbc.gridx=0;
gbc.gridy=1;
Button b3 = new Button ("Button 3");
f.add(b3,gbc);

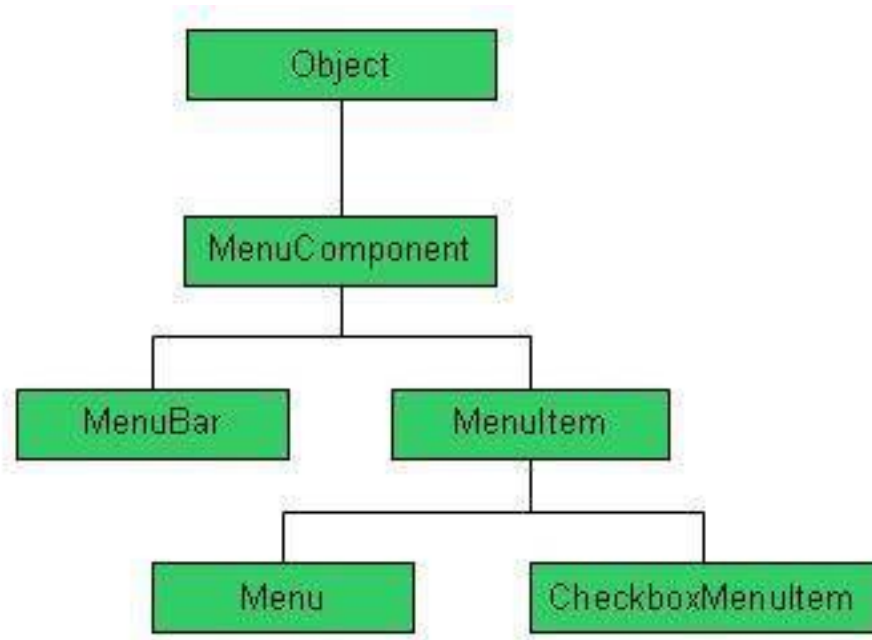
f.setSize(300,300);
f.setVisible(true);
}
}

```

1.4 Menubar and Menu

Menu Bars and Menus

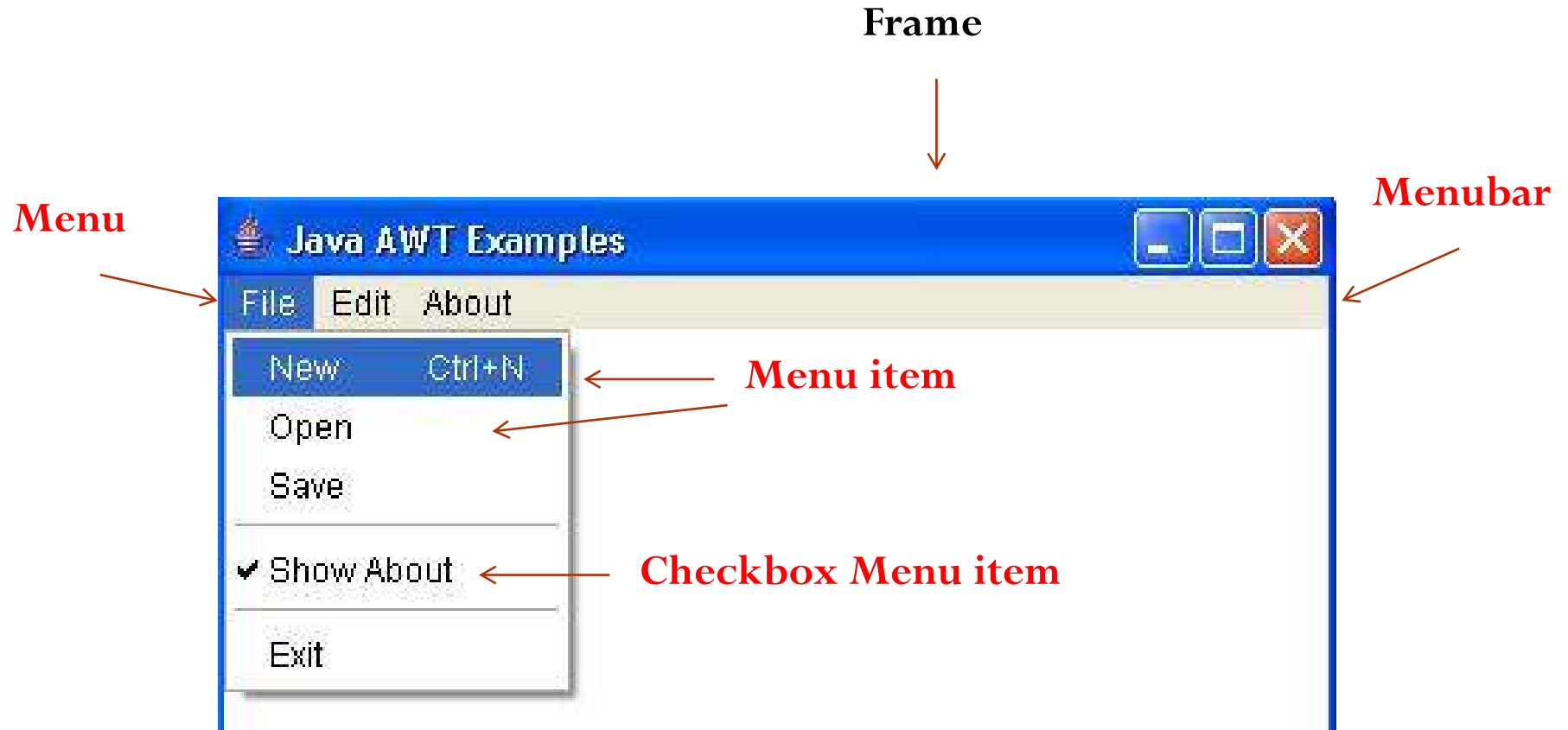
1. A top-level window can have a menu bar associated with it.
2. A menu bar displays a list of top-level menu choices. Each choice is associated with a drop down menu.
3. Fig shows menu hierarchy.



Menu Bars and Menus

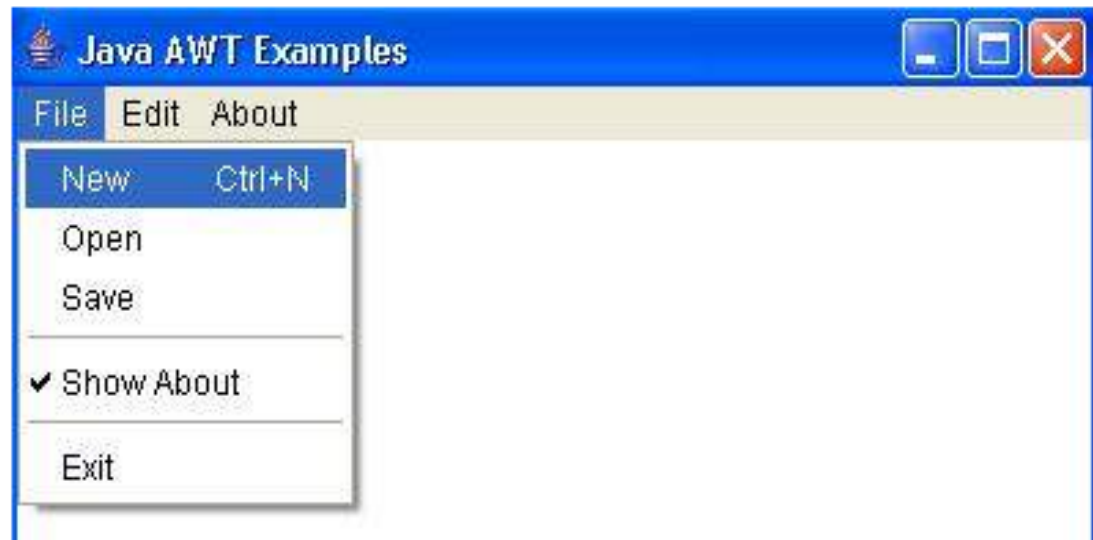
1. To create menus, the java.awt package comes with mainly four classes – MenuBar, Menu, MenuItem and CheckboxMenuItem.
2. All these four classes are not AWT components as they are not subclasses of **java.awt.Component** class.
3. In fact, they are subclasses of **java.awt.MenuComponent**.

Menu Bars and Menus



1. **MenuBar:** MenuBar holds the menus. MenuBar is added to frame with **setMenuBar()** method.
 - Implicitly, the menu bar is added to the **north (top)** of the frame.
 - MenuBar cannot be added to other sides like south , west, etc.

2. **Menu:** Menu holds the menu items. Menu is added to frame with **add()** method.
 - A sub-menu can be added to Menu.



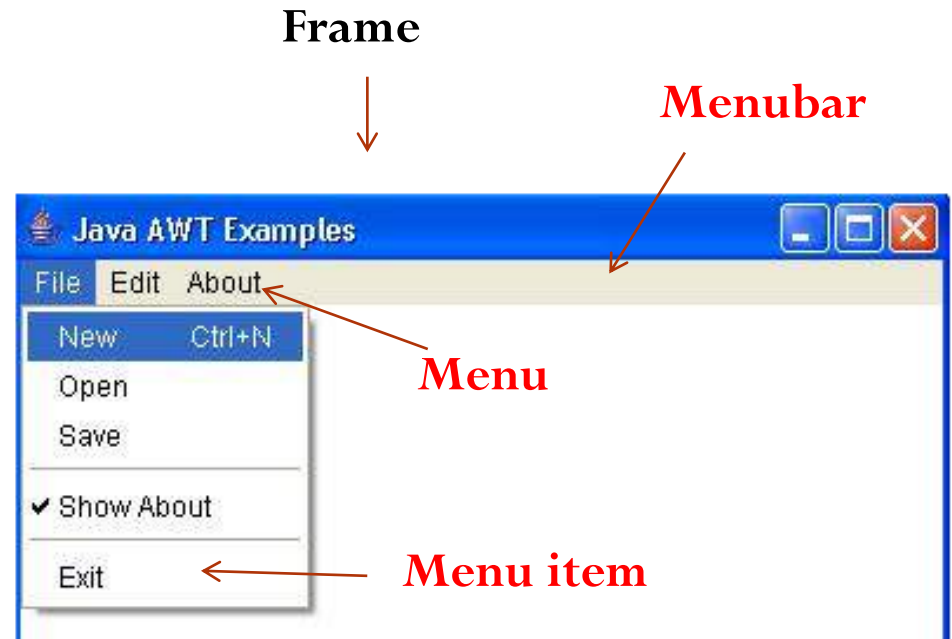
3. **MenuItem:** MenuItem displays the actual option user can select.
 - Menu items are added to menu with method **addMenuItem()**.
 - A dull-colored line can be added in between menu items with **addSeparator()** method.

4. **CheckboxMenuItem:** It differs from MenuItem in that it appears along with a checkbox.
 - The selection can be done with checkbox selected.
 - For that **CheckboxMenuItem(String itemname, Boolean on)** constructor is used.



Steps of Creating Java AWT Menu

1. Create menu bar
2. Add menu bar to the frame
3. Create menus
4. Add menus to menu bar
5. Create menu items
6. Add menu items to menus



Menu Bars

1. Constructors

Sr. No	Constructors	Description
1	MenuBar()	Create menubar
2	Menu()	To create a empty / default menu.
3	Menu(String name)	To create a menu with name.
4	MenuItem()	Create a default menu item.
5	MenuItem(String itemname)	To create a menu item by using name of menu item.
6	MenuItem(String itemname, MenuShortcut shortcut)	To create a menu item by using name of menu item and shortcut.

Program - Menu Bar and Menu

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

```
public class MenuEx extends Frame
```

```
{
```

```
    MenuEx ( )
```

```
{
```

```
    MenuBar mb = new MenuBar( );
```

```
    setMenuBar(mb); // add menu bar to frame
```

```
    Menu a = new Menu("Colors");
```

```
    Menu b = new Menu("Fonts");
```

```
    Menu c = new Menu("Exit");
```

```
    mb.add(a); // add menus to menu bar
```

```
    mb.add(b);
```

```
    mb.add(c);
```

```
    setSize(300, 300);
```

```
    setVisible(true);
```

```
}
```

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

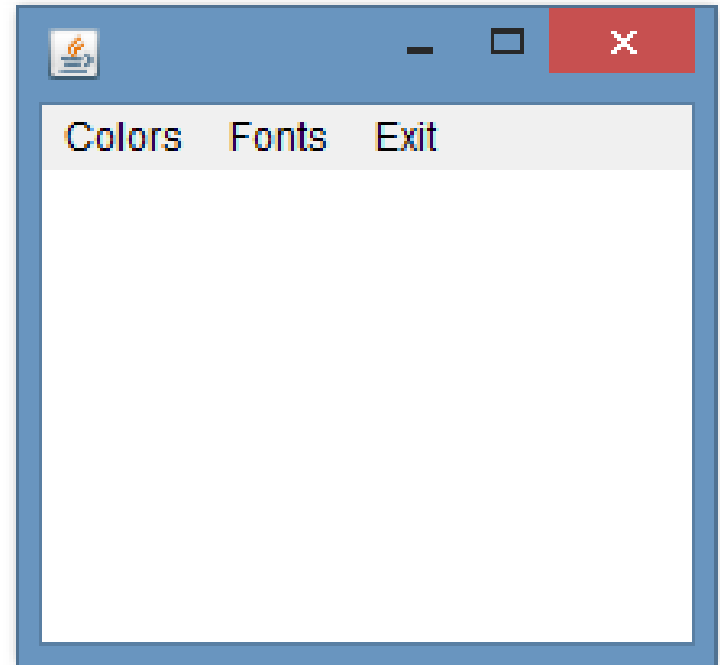
```
{
```

```
        MenuEx obj =new MenuEx( );
```

```
}
```

```
}
```

Output -



Program - MenuItem and CheckboxMenuItem

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

```
public class MenuDemo extends Frame
```

```
{
```

```
    MenuDemo ( )
```

```
    {
```

```
        MenuBar mb = new MenuBar( ); // begin with creating menu bar  
        setMenuBar(mb); // add menu bar to frame
```

```
        Menu a = new Menu("Colors"); // create menus
```

```
        Menu b = new Menu("Fonts");
```

```
        Menu c = new Menu("Exit");
```

```
        mb.add(a); // add menus to menu bar
```

```
        mb.add(b);
```

```
        mb.add(c);
```

```
        MenuItem m1 = new MenuItem("RED"); // create menuItem in menu
```

```
        //MenuItem m2 = new MenuItem("GREEN");
```

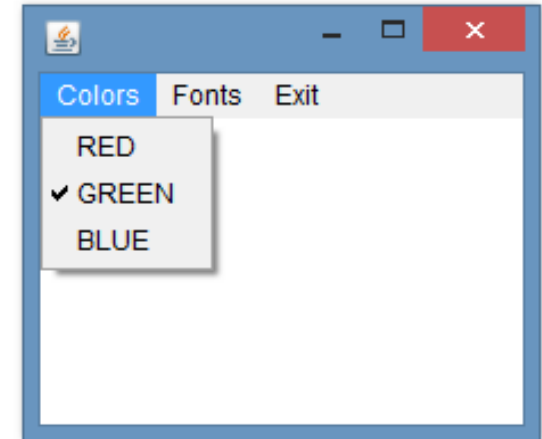
```
        CheckboxMenuItem m2 = new CheckboxMenuItem("GREEN",true);
```

```
        MenuItem m3 = new MenuItem("BLUE");
```

```
        a.add(m1); // add menuItem in menu
```

```
        a.add(m2);
```

```
        a.add(m3);
```



Program to continue..

```
MenuItem m4 = new MenuItem("Times New Roman");
```

```
MenuItem m5 = new MenuItem("Cambria");
```

```
MenuItem m6 = new MenuItem("Arial");
```

```
b.add(m4);
```

```
b.addSeparator( ); // add separator after Times New Roman font
```

```
b.add(m5);
```

```
b.addSeparator( );
```

```
b.add(m6);
```

```
setSize(300, 300);
```

```
setVisible(true);
```

```
}
```

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

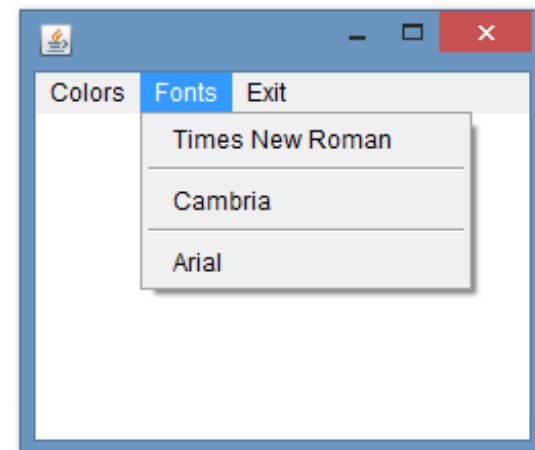
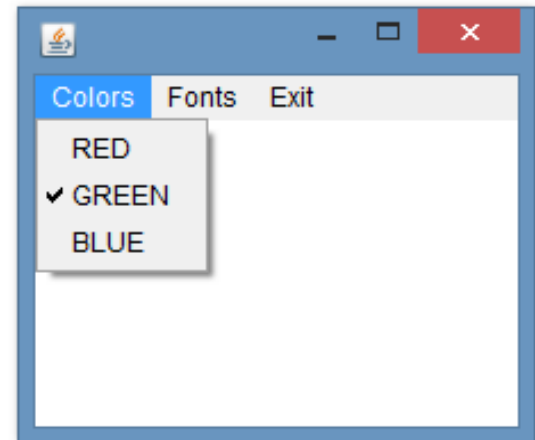
```
{
```

```
    MenuDemo obj =new MenuDemo( );
```

```
}
```

```
}
```

Output -



Unit -1 Abstract Windowing Toolkit

Dialog Box and File Dialog Box

Dialog Boxes

1. A Dialog box is a GUI object in which you can place messages that you want to display on the screen.
2. Dialog boxes are similar to frame windows, except that dialog boxes are always child windows of a Top-level window. It is a child window and must be connected to a main program or to a parent window.
3. Also, Dialog boxes do not have menu bars.
4. A Dialog box works within a main program. It cannot be created as a standalone application.

Dialog Boxes

1. Two types of Dialog boxes exist - **Modal and Modelless.**
2. When a Modal dialog box is Active, we cannot access other parts of our program until we have closed the dialog box. Means Modal dialog box does not allow the user to do any activity without closing it. **For Eg- File Deletion Confirmation dialog box.**
3. When a Modelless dialog box is active, input focus can be directed to another window in our program. Thus , other parts of our program remain active and accessible. Means Modelless dialog box permits the user to do any activity without closing it. **For Eg- Find and Replace dialog box of MS Word.**

Constructors

Sr. No	Constructors	Description
1	Dialog (Frame parent)	Constructors creates an instance of Dialog with no title and with parent as the Frame owning it. It is not modal and is initially resizable.
2	Dialog (Frame parent, String title)	creates an instance of Dialog with parent as the Frame owning it and a window title of title. It is not modal and is initially resizable.
3	Dialog (Frame parent, boolean modal)	Constructors creates an instance of Dialog with no title and with parent as the Frame owning it. It is not modal and is initially resizable. If modal is true, the Dialog grabs all the user input of the program until it is closed. If modal is false, there is no special behavior associated with the Dialog.

Program – Dialog box

```
import java.awt.*;

public class DialogEx extends Dialog
{
    DialogEx(Frame parent, String title)
    {
        super(parent, title, false);
        setLayout(new FlowLayout( ));
        setSize(300,200);
        setBackground(Color.pink);
        Button b1 = new Button ("Okay");
        add(b1);
    }
    public static void main(String args[ ])
    {
        Frame f = new Frame( );
        DialogEx obj = new DialogEx(f, "Hello" );
        obj.setVisible(true);
    }
}
```

Output -



File Dialog

1. Java provides a built-in dialog box that lets user specify a file.
2. To create a file dialog box, instantiate an object of type `FileDialog`.
3. This causes a file dialog box to be displayed.
4. Usually , this is the standard file dialog box provided by the operating system.

Constructors

Sr. No	Constructors	Description
1	FileDialog (Frame parent)	creates a file dialog for loading a file.
2	FileDialog (Frame parent String boxName)	Creates File Dialog box where parent is the owner of the dialog box and boxName is the name displayed in the box's title bar.
3	FileDialog (Frame parent String boxName, int how)	<p>Creates File Dialog box where parent is the owner of the dialog box and boxName is the name displayed in the box's title bar.</p> <p>If how is FileDialog.LOAD. Then the box is selecting a file for reading.</p> <p>If how is FileDialog.SAVE, the box is selecting a file for writing.</p>

Program – File Dialog Box

```
import java.awt.*;

class SampleFrame extends Frame
{
    SampleFrame(String title)
    {
        super(title);
    }
}

public class FileDialogEx
{
    public static void main(String args[ ])
    {
        SampleFrame f = new SampleFrame("File Dialog Demo" );
        f.setVisible(true);
        f.setSize(300,300);
        FileDialog obj = new FileDialog(f, "File Dialog" );
        obj.setVisible(true);
    }
}
```

Program - Output

