UNIT-2

- APPlet is a special type of program that embedded in to web page to generate the dynamic content It runs inside the browser and work at client side.
- APPlet cannot be executed Independently

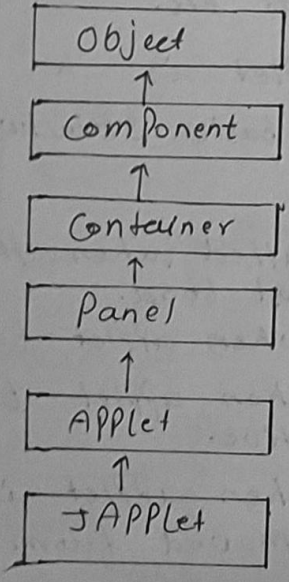
ADVANTAGE

- It works at client side so less response time
- Secure
- It can be executed different types of ~~browser~~ Platform including Linux, windows etc

DRAWBACK

- Plugin is required at client ~~side~~ browser to execute applet

HIERARCHY of APPLET



java.lang.Object :- This is the root of all classes in java. It provide basic methods such as toString(), equals() and hashCode() that every java object inherit.

~~java.lang~~ java.awt.Component :- The component class is the base for all AWT (Abstract Window Toolkit) graphical components
Methods: setSize(), setVisible() & paint()

java.awt.Container :- Container is a subclass of ~~container~~ component that can hold other component (like button, text field etc)

java.awt.Panel :- Panel is concrete subclass of container and also is a generic container for holding component. An Applet is also type of Panel, and thus, can hold various ~~type~~ component such as button, textfield etc.

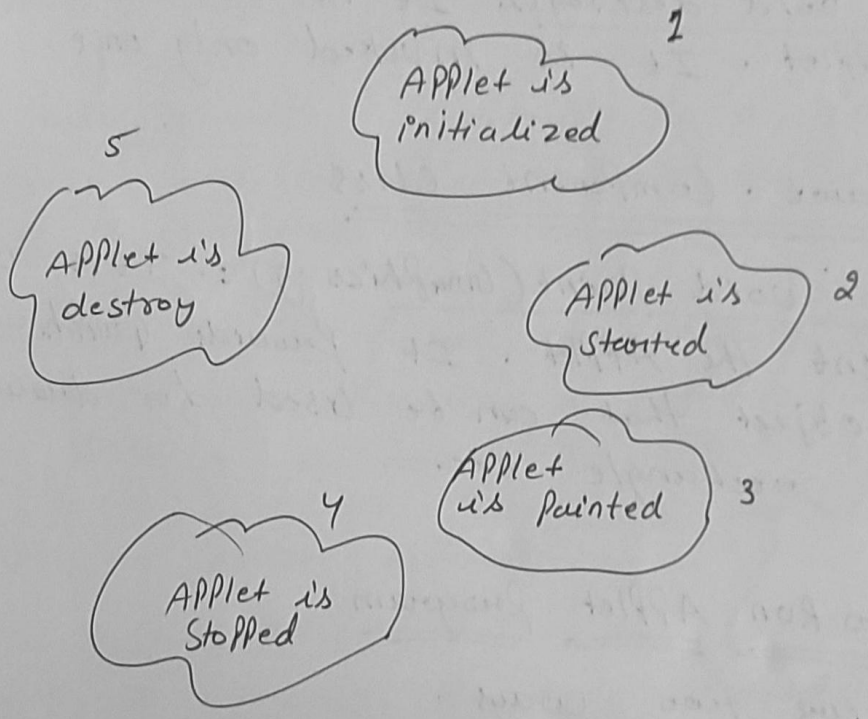
java.applet.Applet :- Applet is a subclass of Panel and provide the basic structure and lifecycle method for applet

Methods: init() → called when the applet is first load.
start() → when applet becomes active
stop() → when applet becomes no longer active
destroy() → when applet is about to be removed from memory.
paint(Graphic g) → Draw output on the screen.

③
javax.swing.JApplet :- (optional for swing based Applet)

- JApplet is a subclass of Applet provide in the swing framework.
- If you are using swing component (JButton, JLabel)
- JApplet provides a better, more flexible GUI component model compare to Applet, but the core lifecycle method remain same.

Applet Life Cycle



→ The java.applet.Applet class 4 life cycle method and java.awt.Component class provide, ~~method~~ lifecycle method for an applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provide 4 life cycle methods.

1) Public void init(): is used to initalized the Applet. It is invoked only once.

2) Public void start(): Is invoked after init() method or browser is maximized. It is used to start the Applet.

3) Public void stop(): It is used to stop the Applet. It is invoked when Applet is stop or browser is ~~max~~ minimized.

4) Public void destroy(): It is used to destroy the Applet. It is invoked only once

java.awt.Component Class

1) Public void Paint(Graphics g) :- It is used to Paint the Applet. It provide Graphics class object that can be used for drawing oval, rectangle etc

Way to Run Applet Program

There are two ways

1) With applet viewer
2) With web browser or By HTML file.

By HTML FILE

To Execute the applet by Html file, create an applet and compile it. After that create an html file and place the applet code in html file.

```
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
  public void Paint(Graphics g)
  {
    g.drawString("welcome", 150, 150);
  }
}
```

Note: Class always public

myapplet.html

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

With AppletViewer; To execute the applet by applet-viewer tool, create an applet that contains applet tag in comment & compile it. After that run it by: appletviewer First.java. Now HTML file is not required but it is for testing purpose only

```
import Java.applet.Applet;
import java.awt.Graphics;
Public Class First extend Applet
{
```

```
Public void Paint (Graphics g)
{
g.drawString ("welcome", 150, 150);
}
}

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

Display Graphics In Applet

It provides many methods for graphics programming.

1> Public abstract void drawString (String str, int x, int y): It is used to draw specified string.

2> Public void drawRect (int x, int y, int width, int height): To draw a rectangle.

3> Public abstract void drawOval (int x, int y, int width, int height): It is used to draw oval.

4> Public abstract void fillOval (int x, int y, int width, int height): It is used fill oval with default color.

5> Public abstract void setColor (Color c):- set the graphics color to specified color.

Example

```
import java.applet.Applet;
import java.awt.*;
public class GraphicsDemo extends Applet
{
  public void paint(Graphics g)
  {
  g.setColor(Color.red);
  g.drawString("Welcome", 150, 150);
  g.drawLine(20, 30, 20, 300);
  g.drawRect(70, 100, 30, 30);
  g.drawOval(70, 200, 30, 30);

  g.set_color(color.Pink);
  g.fillOval(170, 200, 30, 30);

  }
}
```

MyApplet.html

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

    </body>
    </html>
```

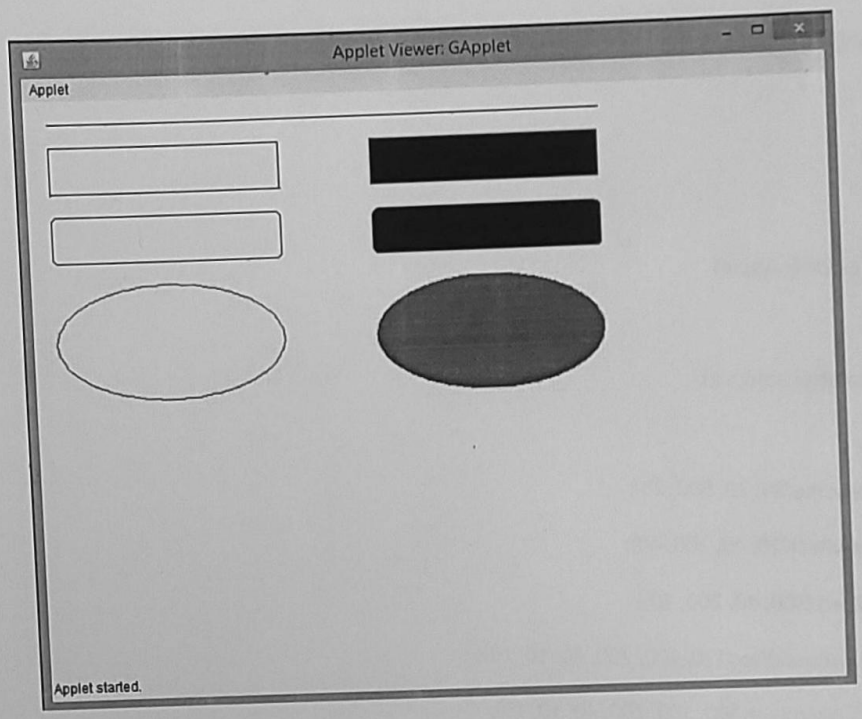
Difference between Java Application and Java Applet

JAVA Application	Java Applet
1) Application are just like a java program that can be executed independently without using web browser	1) Applet are small Java Program that are designed to be include with HTML web document.
2) The Application Program requires a main() methods for its execution.	2) The applet does not require the main() method for its execution instead init() method is required.
3) The javac command is used to compile application program	3) Applet Program are complied with javac command & run using either the appletviewer.
4) Java Application program have full access to the local file system & network	4) Applet do not have local disk & network access
5) Connectivity with other server is possible	5) It is unable to connect to other server
6) It can not run on its own: it needs JRE to execute	6) It cannot start on its own, but it can be executed using a java enabled web browser.

Question:Draw Rectangle,oval,Line in Applet?

```
import java.applet.*;
import java.awt.*;
public class GApplet extends Applet
{
	public void paint(Graphics g)
	{
		g.drawLine(20, 20, 500, 20);
		g.drawRect(20, 40, 200, 40);
		g.fillRect(300, 40, 200, 40);
		g.drawRoundRect(20, 100, 200, 40, 10, 10);
		g.fillRoundRect(300, 100, 200, 40, 10, 10);
		g.setColor(Color.RED);
		g.drawOval(20, 160, 200, 100);
		g.fillOval(300, 160, 200, 100);
	}
}
/*
<applet code="GApplet" height="500" width="700" border="1">
</applet>
*/
```

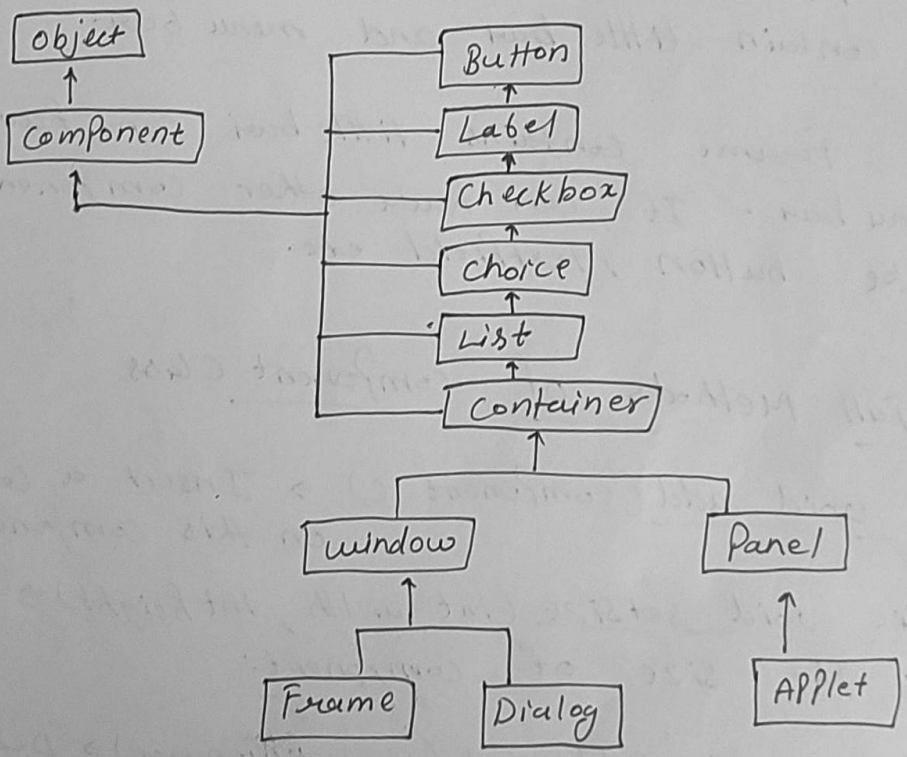
OUTPUT:


Applet Viewer: GApplet
Applet
Applet started.

AWT (Abstract window toolkit) is an API to develop Graphical User Interface (GUI) or window based application in java.

- java AWT components are platform dependent.
- The java.awt packages provide classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox etc.
- It is a heavyweight.

JAVA AWT Hierarchy



Component: All element like the button, textfield, scroll bar etc. are called component.

Container: The container is a component in AWT that can contain another component like buttons, textfield, labels etc.

Type of container : There are 4 types.

1) window
2) Panel
3) Frame
4) Dialog

window :- The window is the containeer that has no border and menu bar. You must use frame, dialog or another window for creating a window.

Panel :- The Panel is the container does not contain title bar and menu bar.

Frame: Frame contains title bar can have menu bar. It can have other component like button, textfield etc.

use_Full Methods of component class

1) Public void add(component c) → Insert a component on this component

2) Public void setSize(int width, int height) → set the size of component.

3) Public void setLayout(LayoutManager) → Defines the layout manager for the component

4) Public void setVisible(boolean Status) → Change the visibility of the component, by default false

There are two ways to create Frame

1> By extending Frame class

2> By creating the object of Frame class.

Example of Extending Frame class

```
import java.awt.*;
public class AWTExample extends Frame
{
  AWTExample()
  {
    // create a button
    Button b = new Button("Click me")
    b.setBounds(30, 100, 80, 30); // set button position
    add(b);          // add button in Frame
    setSize(300, 300); // set size of Frame
    setTitle("This is AWT Example");
    setLayout(null);
    setVisible(true);
  }
  public static void main(String[] args)
  {
    // create instance of frame class.
    AWTExample f = new AWTExample();
  }
}
```

2) Example of creating the object of frame

Here we are creating instance of Frame class. we are creating a TextField, Label & Button component on the Frame.

```
import java.awt.*;
class AWTExample
{
  AWTExample()
{
// creating a Frame
 Frame f= new Frame();
// creating a label
 Label l = new Label("Employee id");
// creating Button
  Button b= new Button("submit");
// creating text field.
 TextField t= new TextField();
// setting positions of above component
 l. setBounds(20, 80, 80, 30);
 t. setBounds(20, 100, 80, 30);
 b. setBounds(100, 100, 80, 30);
// adding component in to frame
```

```
f.add(b);
f.add(l);
f.add(t);
// frame size set
f.setSize(400,300);
// setting tittle of frame
f.setTittle("Employee info");
f.setLayout(null);
f.setVisible(true);
}
public static void main(String[] args)
{
AWTExample aa = new AWTExample();
}}
```

~~EVEN~~ EVENT HANDLING

Changing the state of an object is known as event. For example, click on button, dragging mouse etc.

The java.awt.event package provide many event classes and Listener interface for event handling

There are three component

1) Events:- An event is an changing in the state of an object

2) Event Source:- Event source is an object that generates an event.

3> **Listeners:-** A Listener is an object that listener to the event. A Listner gets notified when an event occur.

Event Classes	Description	Listener Interface
Action Event	generated when button is Pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, move, clicked, Pressed	MouseListener
KeyEvent	generated when input recieved from keyword	KeyListner
TextEvent	generated when value of textarea or textfield is changed	TextListner
AdjustmentEvent	generated when Scroll bar is manipulated	Adjustment Listener
WindowEvent	generated when window is activated, deactivated, deiconified iconified, opened or closed	WindowListener
ContainerEvent	generated when component is added or removed from container	ContainerListener

Implements the ~~described~~ desired listener interface

→ Public class A extends Applet implement ActionListener

* import java.awt.event.*;

* Register listener with desired component
→ addActionListener()

* Implement the listener interface method
Public void action performed (ActionEvent e)

1) ActionListener :-

→ It is an interface which defines actionPerformed()

→ use to handle event caused by sources like Buttons, menu items, Press enter in a text field.

→ Public void action Performed (ActionEvent ae)

2) Mouse MotionListener

→ Defines two methods

mouseDragged()
Void mouseDragged (MouseEvent e)

MouseMoved()
Void mouseMoved (MouseEvent e)

3) ItemListener: used for Radio Buttons, List, choice & Checkbox

→ void itemStateChanged (ItemEvent ie)

Events.

4) KeyListener :

→ Handles event generated from the key of a keyboard

→ Void keyPressed(KeyEvent e)
→ Void keyReleased(KeyEvent e)
→ Void keyTyped(KeyEvent e)

(5) Mouse Listener

Defines 5 methods to recognize when the mouse is clicked, enter a component, exits a component, pressed or released.

Void mouseClicked(MouseEvent e)
Void mouseEntered(MouseEvent e)
Void mouseExited(MouseEvent e)
Void mousePressed(MouseEvent e)
Void mouseReleased(MouseEvent e)

(6) MouseWheelListener

→ Defines ~~MouseWheelListener~~ mouseWheelMoved() method when the mouse wheel is moved.

Void mouseWheelMoved(MouseWheelEvent me)

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

OR

```
import java.awt.*;
import java.awt.event.*;
public class ActionListenerExample implements ActionListener
{ public static void main (String [] args)
{
     Frame f= new Frame ("Example");
     TextField tf = new TextField();
      tf.setBounds (50, 50, 150, 20);
```

```
Button b = new Button ("Click");
b.setBoonds (50, 100, 60, 30);
b.addActionListener (this);
f.add(b); f.add(tf);
f.setSize(400, 400);
f.setLayout (null);
f.setVisible (true);
}
Public void actionPerformed (ActionEvent e)
{ tf.setText(" Welcome to Java");
}}
```

MouseListener

```
import java.awt.*;
import java.awt.event.*;
Public class MouseListener extends Frame implements
MouseListener
{
Label l;
MouseListener()
{
addMouseListener(this);
l = new Label();
l.setBoonds (20, 50, 100, 20);
add(l);
setsize (300, 300);
setLayout(null);
setVisible(true);
}
Public void mouseClicked (MouseEvent e)
{ l.setText(" Mouse Clicked"); }
```

```
Public void mouseEntered (MouseEvent e)
    {
        l. setText ("Mouse Entered");
    }
Public void mouseExited (MouseEvent e)
    {
        l. setText ("Mouse Exit");
    }
public void mousePressed (MouseEvent e)
    {
        l. setText ("Mouse Pressed");
    }
Public void mouseReleased (MouseEvent e)
    {   l. setText ("Mouse Realeased");
    }
public static void main (String [] args)
 {
 MouseListener cf = new MouseListener ();
 }}
```

(23)

Adapter class Example

```
import java.awt.*;
import java.awt.event.*;
Public class AdapterDemo
    {
        Frame f;
        AdapterDemo ()
    {
```

Java adapter classes

Java adapter classes provide the default implementation of listner interface. If you inherit the adapter class, you will not be forced to provide the implementation.

- The adapter classes found java.awt.dnd and javax.swing.event Package.

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
componentAdapter	ComponentListener

Adapter class Example

```
import java.awt.*;
import java.awt.event.*;
Public class AdapterDemo
    {
        Frame f;
        AdapterDemo()
    {
```

```
f=new Frame("Adapter Demo");
f.setVisible(true);
f.setSize(500,500);
f.addWindowListener(new MyAdapter());
}
class MyAdapter extends WindowAdapter
{ public void windowClosing(WindowEvent e)
  {
    f.dispose();
  }
}
Public static void main(String[] args)
{
   new AdapterDemo();
}
}
```

AWT Controls : It may be defined as the allows you to build graphical user Interface These are control Part of java.awt package.

1> **Button** : A button is a clickable that trigger an action when clicked

- Class → java.awt.Button

```
Button button = new Button ("Click me");
add(button);
```

2> **Label**: A label is a non interactive text element used to display information

- Class → java.awt.Label

```
Label label = new Label("This is label");
add(label);
```

3) **Combo box** : It is not a direct class named Combo box, but you can the choice class. which Perovide similiar fonctionality to comboBox.
→ It allow to user to select one item from a dropdown list items.

```
Choice choice = new Choice();
choice.add("Item 1");
Choice.add("Item 2");

add(choice);
```

4) List: A List allows the user to select one or multiple items from a list of items

Class: java.awt.List

```
List list = new List(4, true);
    list.add("Item 1");
    list.add("Item2");
```

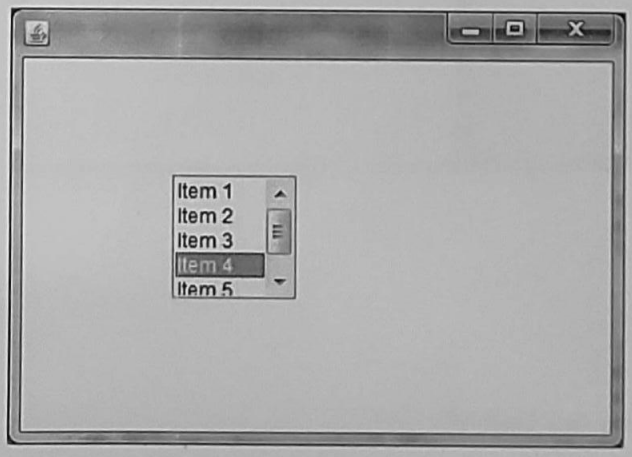
5) Menu bar: The component create menus & Submenus in AWT based Application

```
MenuBar menuBar = new MenuBar();
Menu menu = new Menu("File");
MenuItem menuItem = new MenuItem("Open");
    menu.add(menuItem);
    menuBar.add(menu);
    frame.setMenuBar(menuBar);
```

Create a list in AWT

```
1.  import java.awt.*;
2.
3.  public class ListExample1
4.  {
5.     // class constructor
6.      ListExample1() {
7.      // creating the frame
8.        Frame f = new Frame();
9.       // creating the list of 5 rows
10.       List l1 = new List(5);
11.
12.       // setting the position of list component
13.       l1.setBounds(100, 100, 75, 75);
14.
15.       // adding list items into the list
16.       l1.add("Item 1");
17.       l1.add("Item 2");
18.       l1.add("Item 3");
19.       l1.add("Item 4");
20.       l1.add("Item 5");
21.
22.       // adding the list to frame
23.       f.add(l1);
24.
25.       // setting size, layout and visibility of frame
26.       f.setSize(400, 400);
27.       f.setLayout(null);
28.       f.setVisible(true);
29.    }
30.
31. // main method
32. public static void main(String args[])
33. {
34.    new ListExample1();
35. }
36. }
```

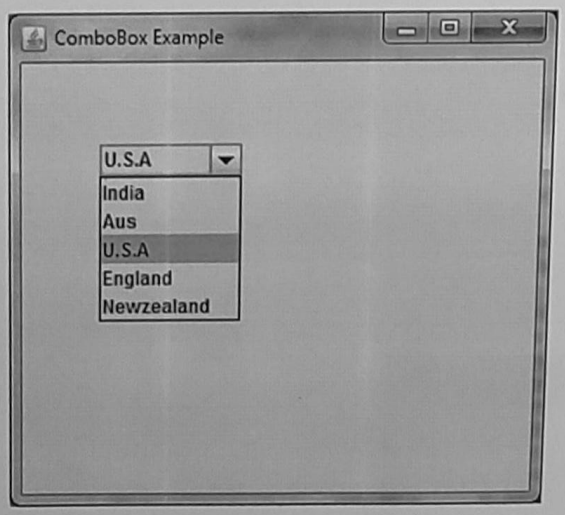
Output:



COMBOBOX

```
1. import javax.swing.*;
2. public class ComboBoxExample {
3. JFrame f;
4. ComboBoxExample(){
5.     f=new JFrame("ComboBox Example");
6.     String country[]={"India","Aus","U.S.A","England","Newzealand"};
7.     JComboBox cb=new JComboBox(country);
8.     cb.setBounds(50, 50,90,20);
9.     f.add(cb);
10.    f.setLayout(null);
11.    f.setSize(400,500);
12.    f.setVisible(true);
13. }
14. public static void main(String[] args) {
15.    new ComboBoxExample();
16. }
17. }
```

Output:

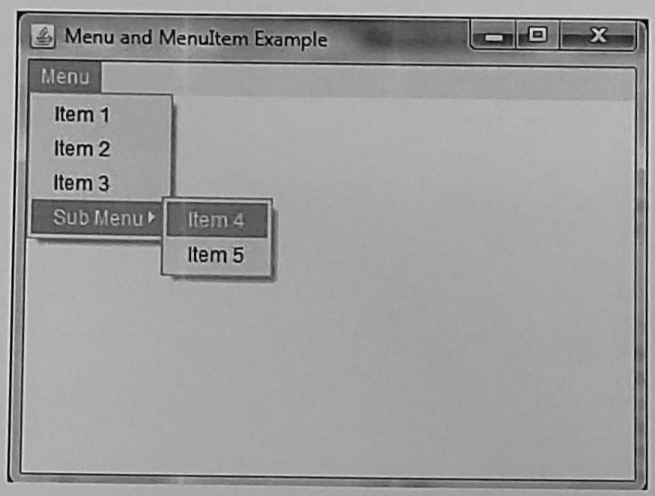


MENUBAR

```
1. import java.awt.*;
2. class MenuExample
3. {
4.     MenuExample(){
5.         Frame f= new Frame("Menu and MenuItem Example");
6.         MenuBar mb=new MenuBar();
7.         Menu menu=new Menu("Menu");
8.         Menu submenu=new Menu("Sub Menu");
```

```
9.          MenuItem i1=new MenuItem("Item 1");
10.         MenuItem i2=new MenuItem("Item 2");
11.         MenuItem i3=new MenuItem("Item 3");
12.         MenuItem i4=new MenuItem("Item 4");
13.         MenuItem i5=new MenuItem("Item 5");
14.         menu.add(i1);
15.         menu.add(i2);
16.         menu.add(i3);
17.         submenu.add(i4);
18.         submenu.add(i5);
19.         menu.add(submenu);
20.         mb.add(menu);
21.         f.setMenuBar(mb);
22.         f.setSize(400,400);
23.         f.setLayout(null);
24.         f.setVisible(true);
25. }
26. public static void main(String args[])
27. {
28. new MenuExample();
29. }
30. }
```

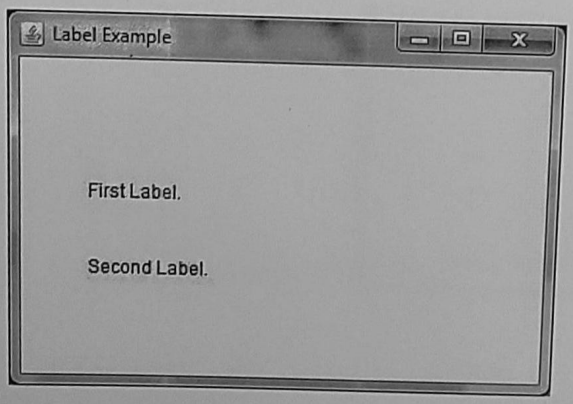
Output:



LABEL

```
1.  import java.awt.*;
2.
3.  public class LabelExample {
4.  public static void main(String args[]){
5.
6.      // creating the object of Frame class and Label class
7.      Frame f = new Frame ("Label example");
8.      Label l1, l2;
9.
10.     // initializing the labels
11.     l1 = new Label ("First Label.");
12.     l2 = new Label ("Second Label.");
13.
14.     // set the location of label
15.     l1.setBounds(50, 100, 100, 30);
16.     l2.setBounds(50, 150, 100, 30);
17.
18.     // adding labels to the frame
19.     f.add(l1);
20.     f.add(l2);
21.
22.     // setting size, layout and visibility of frame
23.     f.setSize(400,400);
24.     f.setLayout(null);
25.     f.setVisible(true);
26. }
27. }
```

Output:



LAYOUT MANAGER

BorderLayout (LayoutManagers)

Java LayoutManagers

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

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

Java BorderLayout

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

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

Constructors of BorderLayout class:

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

Example of BorderLayout class: Using BorderLayout() constructor

FileName: Border.java

```
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.}
```

GRID LAYOUT

(in

Java GridLayout

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

Constructors of GridLayout class

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

zontal

5 unit

nt and

Example of GridLayout class: Using GridLayout() Constructor

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

FileName: GridLayoutExample.java

```
1. // import statements
2. import java.awt.*;
3. import javax.swing.*;
4.
5. public class GridLayoutExample
6. {
7. JFrame frameObj;
8.
9. // constructor
10. GridLayoutExample()
11. {
12. frameObj = new JFrame();
13.
14. // creating 9 buttons
15. JButton btn1 = new JButton("1");
16. JButton btn2 = new JButton("2");
17. JButton btn3 = new JButton("3");
18. JButton btn4 = new JButton("4");
19. JButton btn5 = new JButton("5");
20. JButton btn6 = new JButton("6");
21. JButton btn7 = new JButton("7");
22. JButton btn8 = new JButton("8");
23. JButton btn9 = new JButton("9");
24.
25. // adding buttons to the frame
26. // since, we are using the parameterless constructor, therfore;
27. // the number of columns is equal to the number of buttons we
28. // are adding to the frame. The row count remains one.
29. frameObj.add(btn1); frameObj.add(btn2); frameObj.add(btn3);
30. frameObj.add(btn4); frameObj.add(btn5); frameObj.add(btn6);
31. frameObj.add(btn7); frameObj.add(btn8); frameObj.add(btn9);
32.
33. // setting the grid layout using the parameterless constructor
34. frameObj.setLayout(new GridLayout());
35.
36.
37. frameObj.setSize(300, 300);
38. frameObj.setVisible(true);
39. }
40.
41. // main method
42. public static void main(String argvs[])
43. {
44. new GridLayoutExample();
45. }
46. }
```

FLOW LAYOUT

Java FlowLayout

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

Fields of FlowLayout class

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

Constructors of FlowLayout class

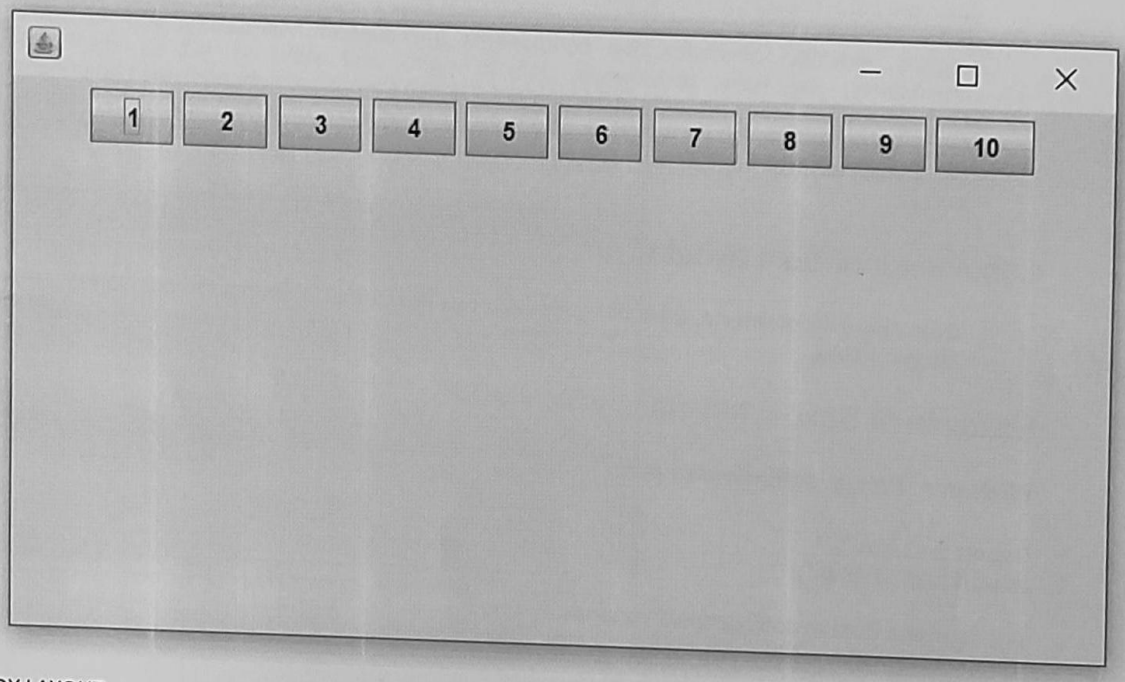
1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates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

Example of FlowLayout class: Using FlowLayout() constructor

FileName: FlowLayoutExample.java

```
1. // import statements
2. import java.awt.*;
3. import javax.swing.*;
4.
5. public class FlowLayoutExample
6. {
7.
8. JFrame frameObj;
9.
10.// constructor
11.FlowLayoutExample()
12.{
13.    // creating a frame object
14.    frameObj = new JFrame();
15.
16.     // creating the buttons
17.    JButton b1 = new JButton("1");
18.    JButton b2 = new JButton("2");
19.    JButton b3 = new JButton("3");
20.    JButton b4 = new JButton("4");
21.    JButton b5 = new JButton("5");
22.    JButton b6 = new JButton("6");
23.    JButton b7 = new JButton("7");
24.    JButton b8 = new JButton("8");
25.    JButton b9 = new JButton("9");
26.    JButton b10 = new JButton("10");
27.
28.
29.    // adding the buttons to frame
30.    frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);
31.    frameObj.add(b5); frameObj.add(b6);  frameObj.add(b7);  frameObj.add(b8);
32.    frameObj.add(b9);  frameObj.add(b10);
33.
34.    // parameter less constructor is used
35.    // therefore, alignment is center
36.    // horizontal as well as the vertical gap is 5 units.
37.    frameObj.setLayout(new FlowLayout());
38.
39.    frameObj.setSize(300, 300);
40.    frameObj.setVisible(true);
41.}
42.
43.// main method
44.public static void main(String argvs[])
```

```
45.{
46.    new FlowLayoutExample();
47.}
48.}
```



BOX LAYOUT

Java BoxLayout

The **Java BoxLayout class** is used to arrange the components either vertically or horizontally. For this purpose, the BoxLayout class provides four constants. They are as follows:

Note: The BoxLayout class is found in javax.swing package.

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.
3. **public static final int LINE_AXIS:** Alignment of the components is similar to the way words are aligned in a line, which is based on the ComponentOrientation property of the container. If the ComponentOrientation property of the container is horizontal, then the components are aligned horizontally; otherwise, the components are aligned vertically. For horizontal

orientations, we have two cases: left to right, and right to left. If the value ComponentOrientation property of the container is from left to right, then the components are rendered from left to right, and for right to left, the rendering of components is also from right to left. In the case of vertical orientations, the components are always rendered from top to bottom.

4. **public static final int PAGE_AXIS:** Alignment of the components is similar to the way text lines are put on a page, which is based on the ComponentOrientation property of the container. If the ComponentOrientation property of the container is horizontal, then components are aligned vertically; otherwise, the components are aligned horizontally. For horizontal orientations, we have two cases: left to right, and right to left. If the value ComponentOrientation property of the container is also from left to right, then the components are rendered from left to right, and for right to left, the rendering of components is from right to left. In the case of vertical orientations, the components are always rendered 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.

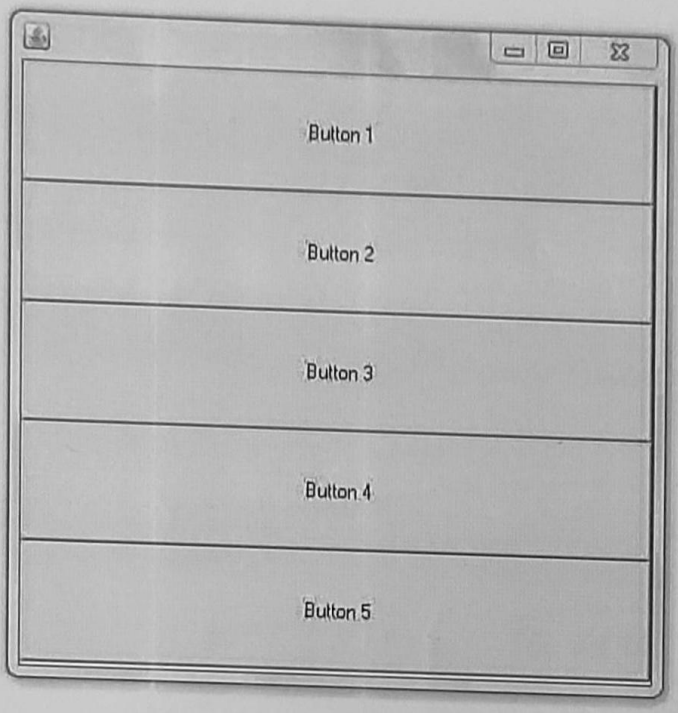
Example of BoxLayout class with Y-AXIS:

FileName: BoxLayoutExample1.java

```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class BoxLayoutExample1 extends Frame {
5.  Button buttons[];
6.
7.  public BoxLayoutExample1 () {
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 BoxLayout (this, BoxLayout.Y_AXIS));
17. setSize(400,400);
18. setVisible(true);
19. }
20.  // main method
21. public static void main(String args[]){
22. BoxLayoutExample1 b=new BoxLayoutExample1();
23. }
24. }
```

download this example

Output:



String handling:- String handling in Java involves performing operations on strings, such as concatenation, comparison, searching, replacing and others.

1. **Concatenation:**

```
String str1 = "Hello";
String str2 = "World";
String str3 = str1 + " " + str2;
System.out.println(str3);
```

output: Hello world

2. **Comparison:** Compare two strings

```
String str1 = "Hello";
String str2 = "Hello";
if (str1.equals(str2))
    {
     System.out.println("String is equal");
    }
else
  {
   System.out.println("String is not equal");
  }
```

Searching:

```
String str = "Hello world";
System.out.println(str.indexOf("world"));
```

output: 6

Replacing

```
String str = "Hello world";
str = str.replace("world", "java");
System.out.println(str);
```

output: Hello java

Important methods

length(): Return the number of character in string

```
String str = "Java Programming";
System.out.println(str.length());
```

output: 16

concat(String str): Concatenates the specified string to the end of this string.

```
String str1 = "Hello";
String str2 = "world";
String result = str1.concat(s2);
System.out.println(result);
```

output: Hello world

equals (object obj)

Check if two String are equal.

```
String s1 = "java";
String s2 = "java";
String s3 = "C";
System.out.println (s1.equals(s2));
System.out.println (s1.equals(s3));
```

output: true
false

indexOf (Char ch)

Return the index of the first occurence of the specified character.

```
String s1 = "java Programming";
System.out.println (s1.indexOf('P'));
```

output : 5

trim() : Remove leading and trailing white-space

```
String str = "   Hello world";
System.out.println (str.trim());
```

output: Hello world.

charAt(int index):

String Returns the character at specified index.

```
String str = "Hello";
System.out.println(str.charAt(1));
```

output: e