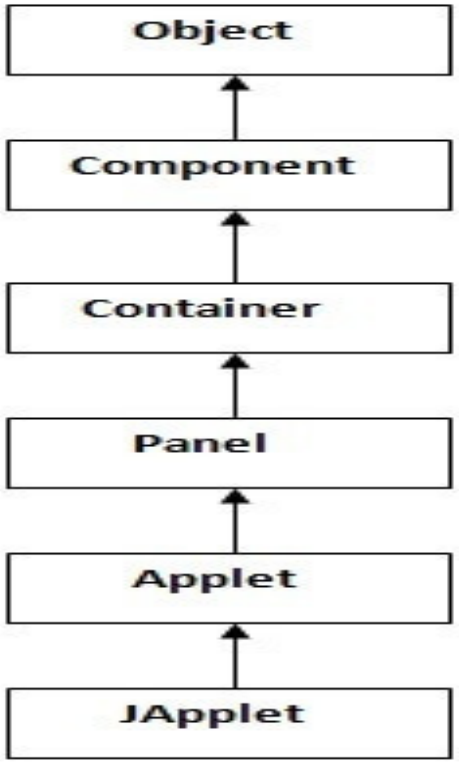
Applet

- Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.
- **Advantage of Applet :**There are many advantages of applet. They are as follows:
- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet :

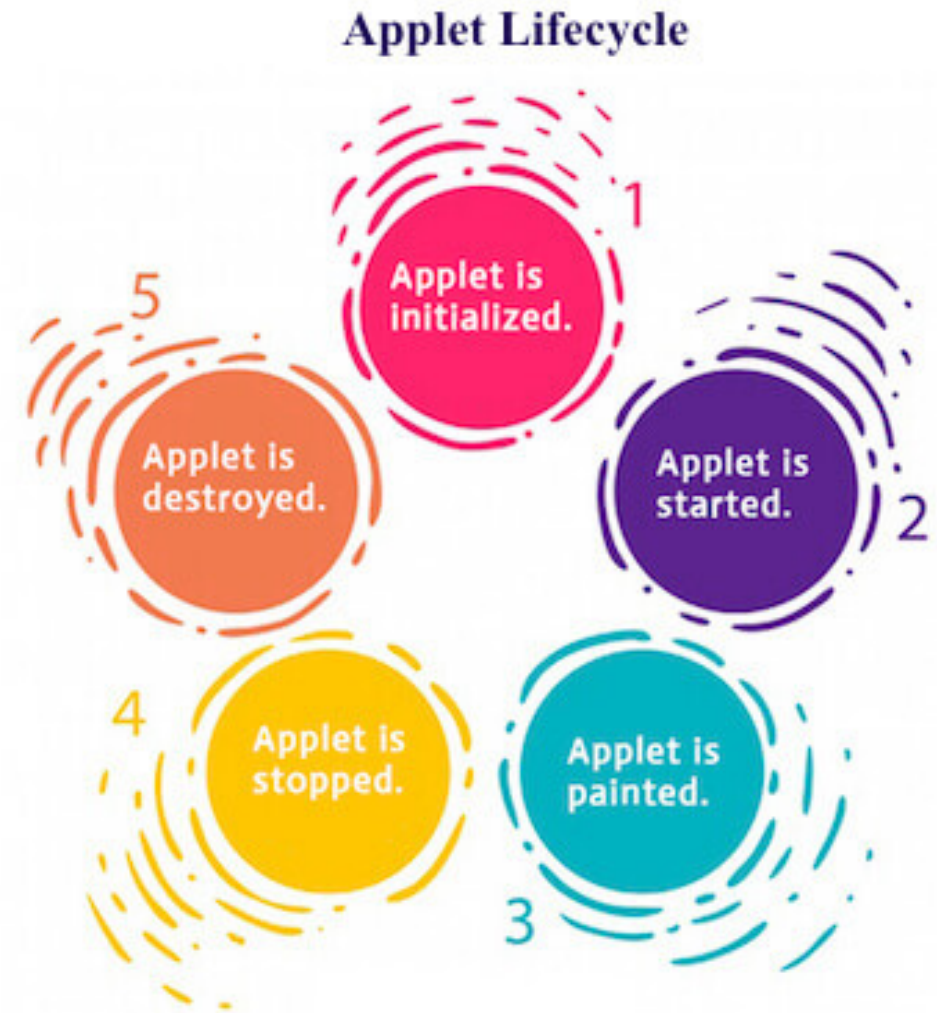
- Plugin is required at client browser to execute applet.

Applet Hierarchy



As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

applet-lifecycle



Lifecycle methods for Applet:

- The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.
- java.applet.Applet class
- For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.
- **public void init():** is used to initialized the Applet. It is invoked only once.
- **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
- **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
- **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class

- The Component class provides 1 life cycle method of applet.
- **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

How to run an Applet?

- There are two ways to run an applet

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

Simple example of Applet by appletviewer tool:

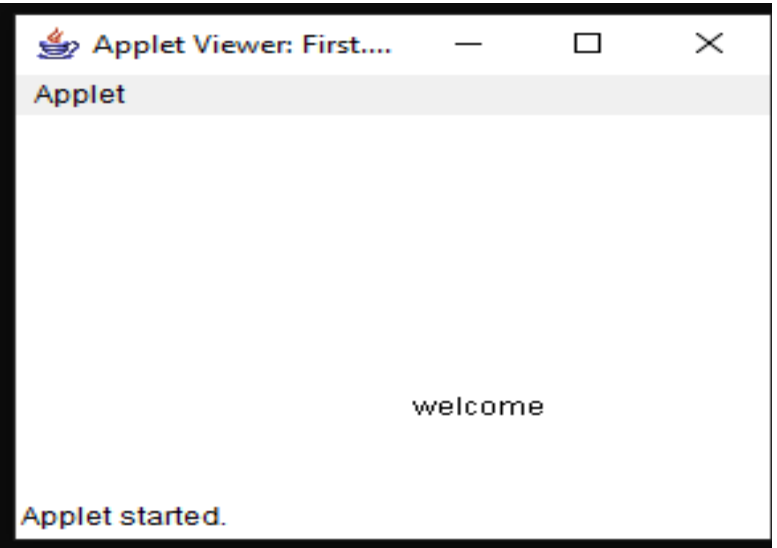
- To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
import java.util.*;
import java.applet.*;
import java.awt.*;
public class First extends Applet
{
	public void paint(Graphics g)
	{
		g.drawString("welcome",150,150);
	}
}
/*
<APPLET code="First.class" width ="200" height="150"></APPLET>
*/
```

```
C:\Users\ps\Desktop\java\unit-4\Applet>javac First.java

C:\Users\ps\Desktop\java\unit-4\Applet>appletviewer First.java

C:\Users\ps\Desktop\java\unit-4\Applet>appletviewer First.java
```



Displaying Graphics in Applet

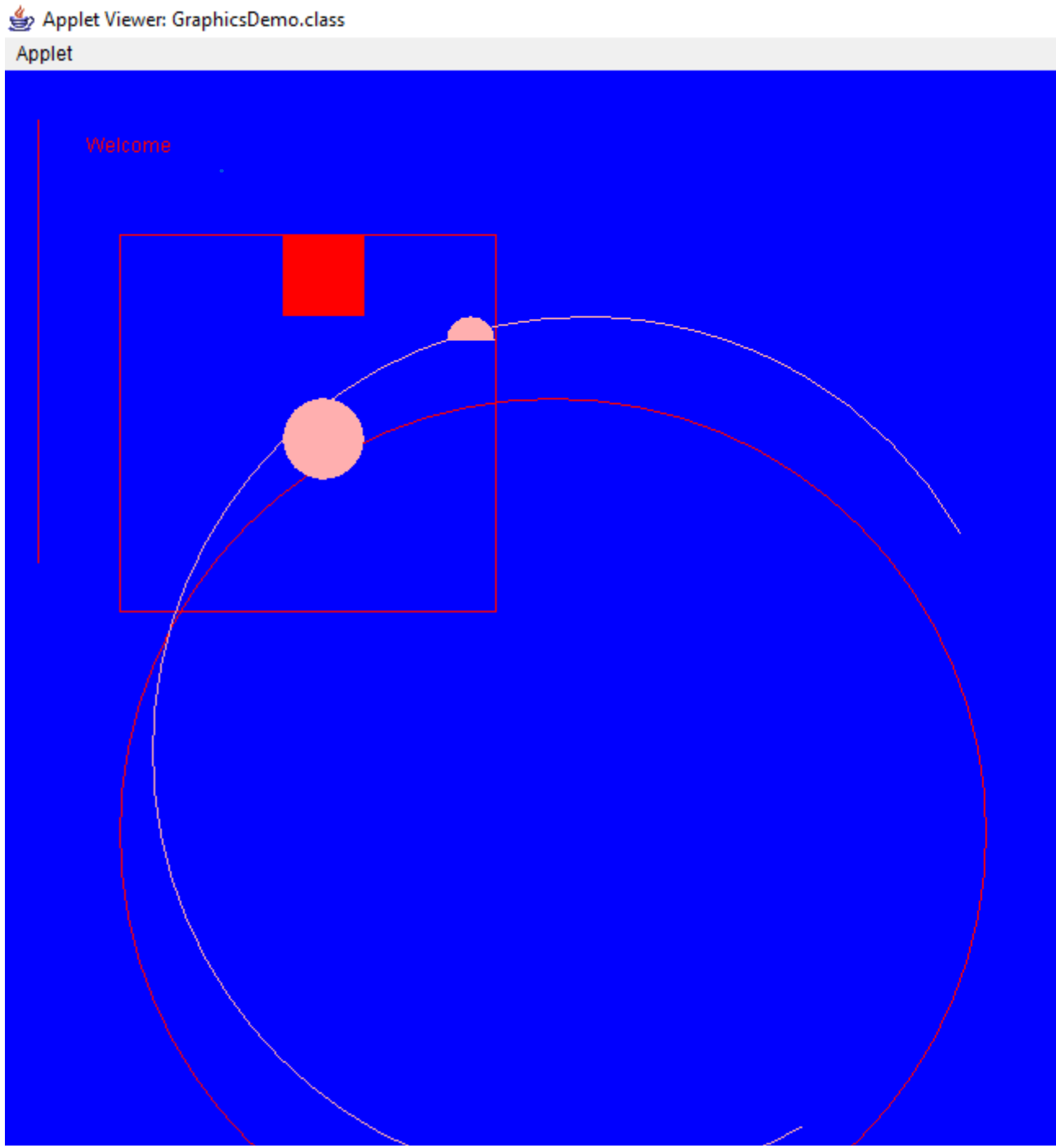
- java.awt.Graphics class provides many methods for graphics programming.

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.

6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

```
import java.applet.Applet;
import java.awt.*;
/*
<applet code="GraphicsDemo.class" width="300"
     height="300">
</applet>
*/
public class GraphicsDemo extends Applet {
   public void paint(Graphics g)
   {
       g.setColor(Color.red);
       g.drawString("Welcome", 50, 50);
        g.drawLine(20, 30, 20, 300);
       g.drawRect(70, 100, 230, 230);
       g.fillRect(170, 100, 50, 50);
       g.drawOval(70, 200, 530, 530);
       g.setColor(Color.pink);
       g.fillOval(170, 200, 50, 50);
       g.drawArc(90, 150, 530, 530, 30, 270);
       g.fillArc(270, 150, 30, 30, 0, 180);
      setBackground(Color.blue);


} }
```



Layout Managers in applet

- 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:

- FlowLayout
- BorderLayout
- CardLayout
- GridLayout
- GridBagLayout with GridBagConstraints
- Intro. to BoxLayout,
- SpringLayout,
- GroupLayout
- Using NO LAYOUT Manager

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

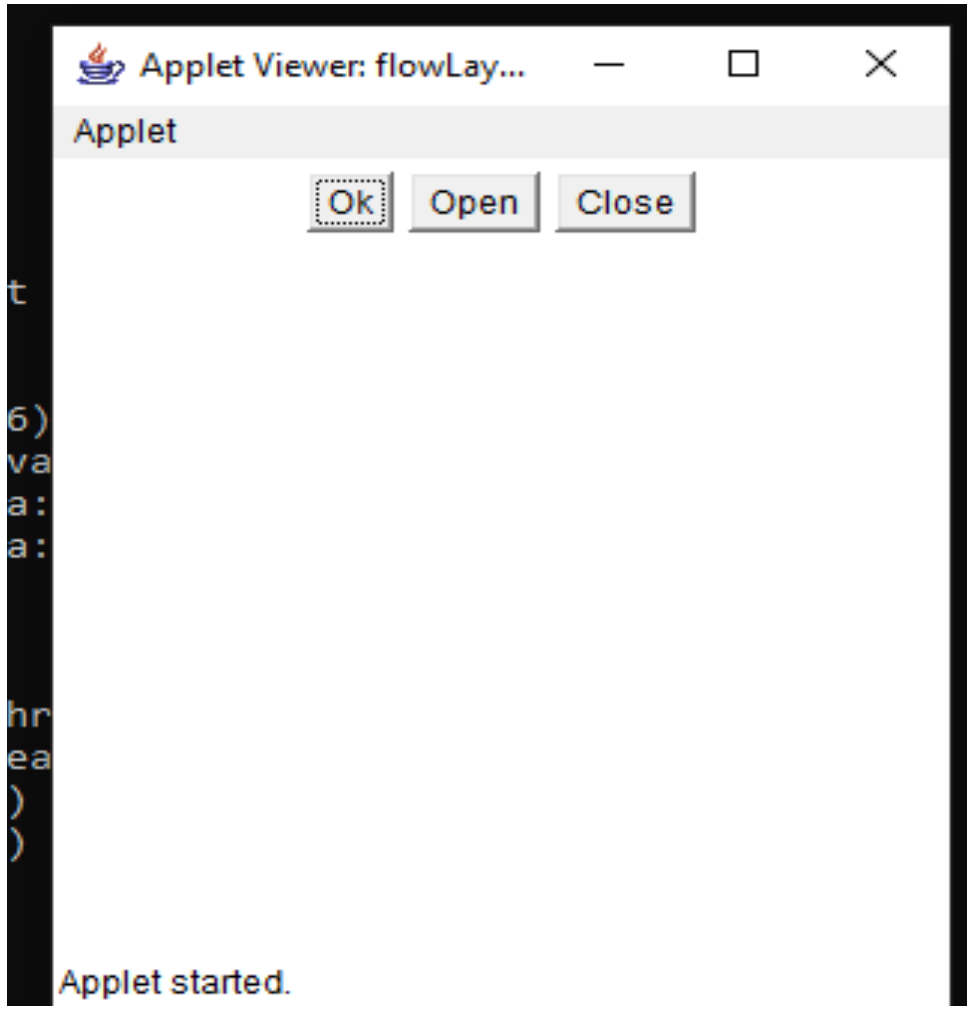
1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

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

```
import java.awt.*;
import java.applet.Applet;
/*
<applet code="flowLayout.class" width="300" height="300">
</applet>
*/

 public class flowLayout extends Applet {
    Button button1, button2, button3;
    public void init() {
       button1 = new Button("Ok");
       button2 = new Button("Open");
       button3 = new Button("Close");
       add(button1);
       add(button2);
       add(button3);
    }
 }
```



Java BorderLayout

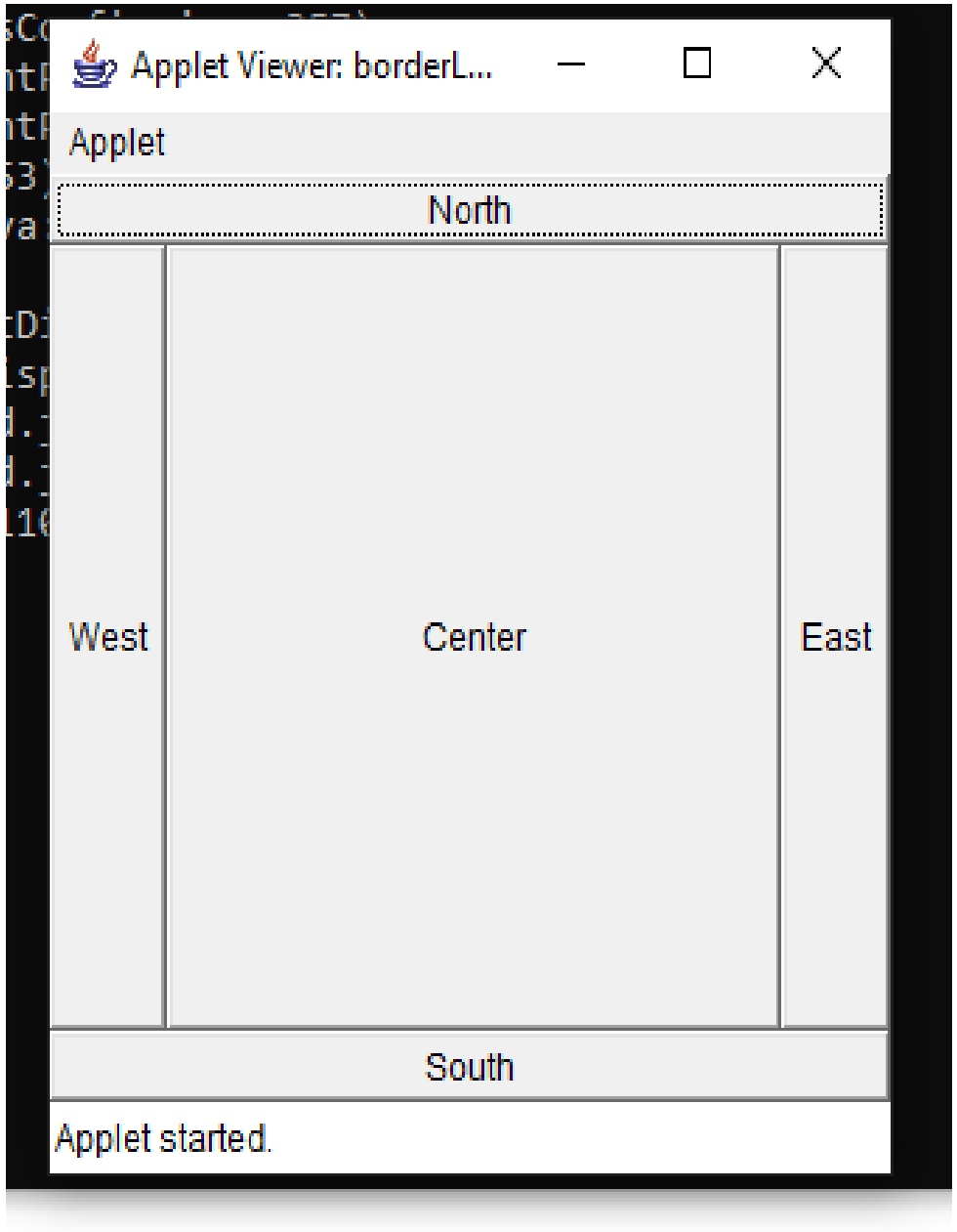
- The BorderLayout is used to arrange the components in five regions: north, south, east, west, and center. Each region (area) may contain one component only. It is the default layout of a frame or window. The Border Layout provides five constants for each region:

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.

```
import java.awt.*;
 import java.applet.Applet;
/*
<applet code="borderLayout.class"
width="300" height="300">
</applet>
*/
 public class borderLayout extends Applet {
   public void init() {
     setLayout(new BorderLayout());
     add(new Button("North"), BorderLayout.
NORTH);
     add(new Button("South"), BorderLayout.
SOUTH);
     add(new Button("East"), BorderLayout.
EAST);
     add(new Button("West"), BorderLayout.
WEST);
     add(new Button("Center"), BorderLayout.
CENTER);
```



Java CardLayout

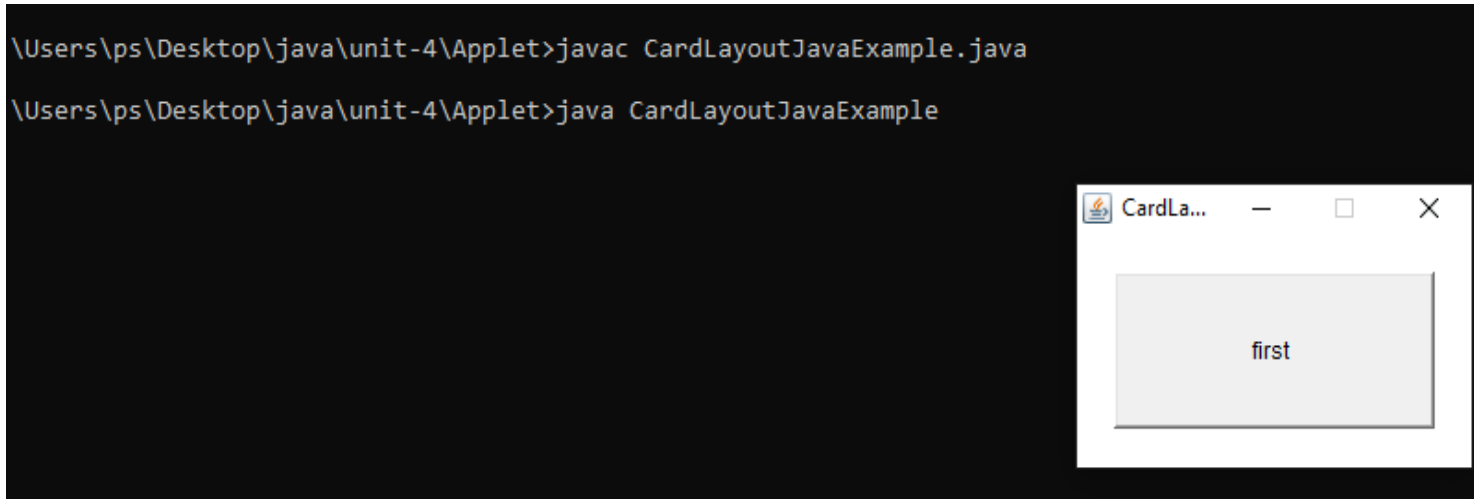
- The **Java CardLayout** class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.
- Constructors of CardLayout Class
- **CardLayout():** creates a card layout with zero horizontal and vertical gap.
- **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Commonly Used Methods of CardLayout Class

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

```
import java.awt.*;
import java.awt.event.*;
class CardLayoutExample extends Frame implements ActionListener
{
    CardLayout card = new CardLayout(20,20);
    CardLayoutExample()
    {
       setLayout(card);
       Button Btnfirst = new Button("first ");
       Button BtnSecond = new Button ("Second");
       Button BtnThird = new Button("Third");
       add(Btnfirst,"Card1");
       add(BtnSecond,"Card2");
       add(BtnThird,"Card3");
       Btnfirst.addActionListener(this);
       BtnSecond.addActionListener (this);
       BtnThird.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e)
    {
      card.next(this);
    }
}
```

```
class CardLayoutJavaExample
  {
      public static void main(String args[])
     {
         CardLayoutExample frame = new
CardLayoutExample();
         frame.setTitle("CardLayout in Java
Example");
         frame.setSize(220,150);
         frame.setResizable(false);
         frame.setVisible(true);
     }
  }
```



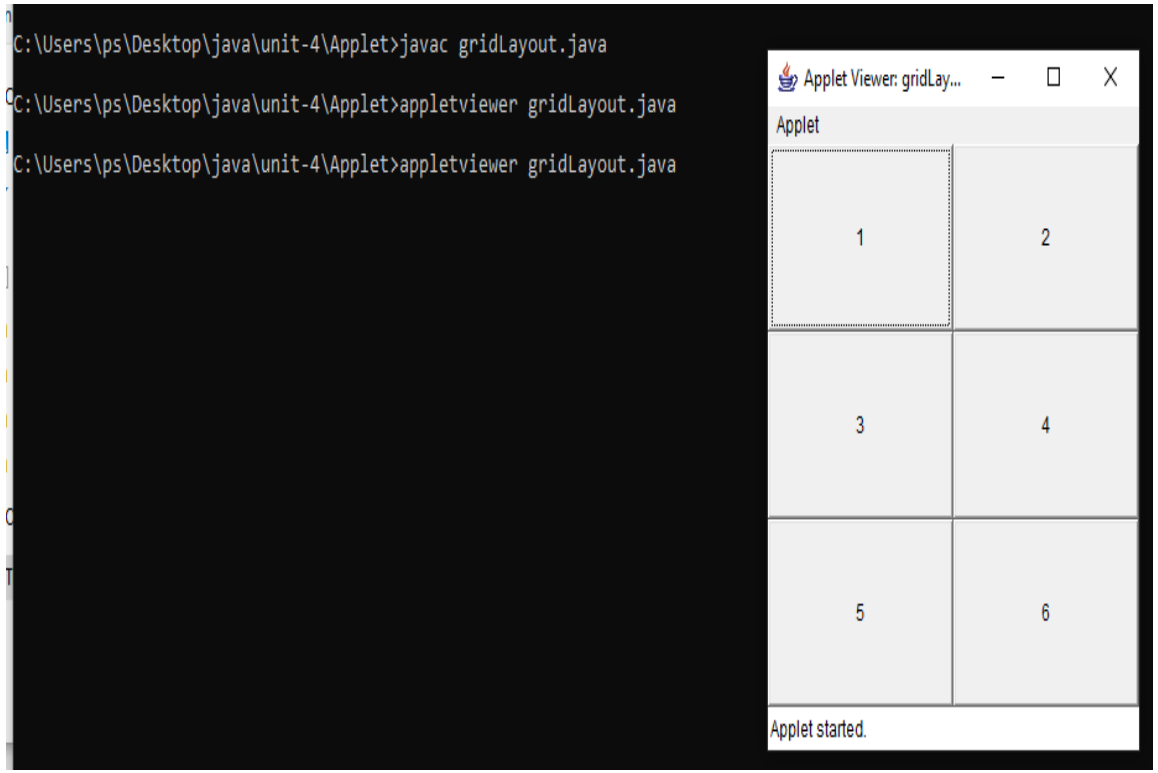
Java GridLayout

- 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.

```
import java.awt.*;
 import java.applet.Applet;
 /*
<applet code="gridLayout.class" width="300"
height="300">
</applet>
*/
 public class gridLayout extends Applet {
    public void init() {
        setLayout(new GridLayout(3,2));
        add(new Button("1"));
        add(new Button("2"));
        add(new Button("3"));
        add(new Button("4"));
        add(new Button("5"));
        add(new Button("6"));
    }
 }
```



GridBagLayout with GridBagConstraints

- The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.
- The components may not be of the same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area.
- Each component associates an instance of GridBagConstraints.
- With the help of the constraints object, we arrange the component's display area on the grid.
- The GridBagLayout manages each component's minimum and preferred sizes in order to determine the component's size.
- GridBagLayout components are also arranged in the rectangular grid but can have many different sizes and can occupy multiple rows or columns.

Constructor

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

```
import java.awt.*;
 class GridBagLayoutExample extends Frame
 {
    GridBagLayoutExample()
    {
       Label lblName = new Label("Name");
       TextField txtName =new TextField(10);
       Label lblcomments = new
    Label("Comments");
       TextArea TAreaComments=new
    TextArea(6,15);
       Button btnSubmit = new
    Button("Submit");
       setLayout(new GridBagLayout());
       GridBagConstraints gc =new
    GridBagConstraints();
       add(lblName,gc,0,0,1,1,0,0);
       add(txtName,gc,1,0,1,1,0,20);
       add(lblcomments,gc,0,1,1,1,0,0);
       add(TAreaComments,gc,1,1,1,1,0,60);
       add(btnSubmit,gc,0,2,2,1,0,20);
    }
```

```
 void add(Component comp,GridBagConstraints gc,int x,
int y,int w,int h,int wx,int wy)
    {
       gc.gridx = x;
       gc.gridy = y;
       gc.gridwidth = w;
       gc.gridheight= h;
       gc.weightx = wx;
       gc.weighty = wy;
       add(comp,gc);
    }
}
 class GridBagLayoutJavaExample
  {
      public static void main(String args[])
     {
         GridBagLayoutExample frame = new
GridBagLayoutExample();
         frame.setTitle("GridBagLayout in Java Example");
         frame.setSize(300,200);
         frame.setVisible(true);
     }
  }
```

BoxLayout

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

Fields of BoxLayout Class

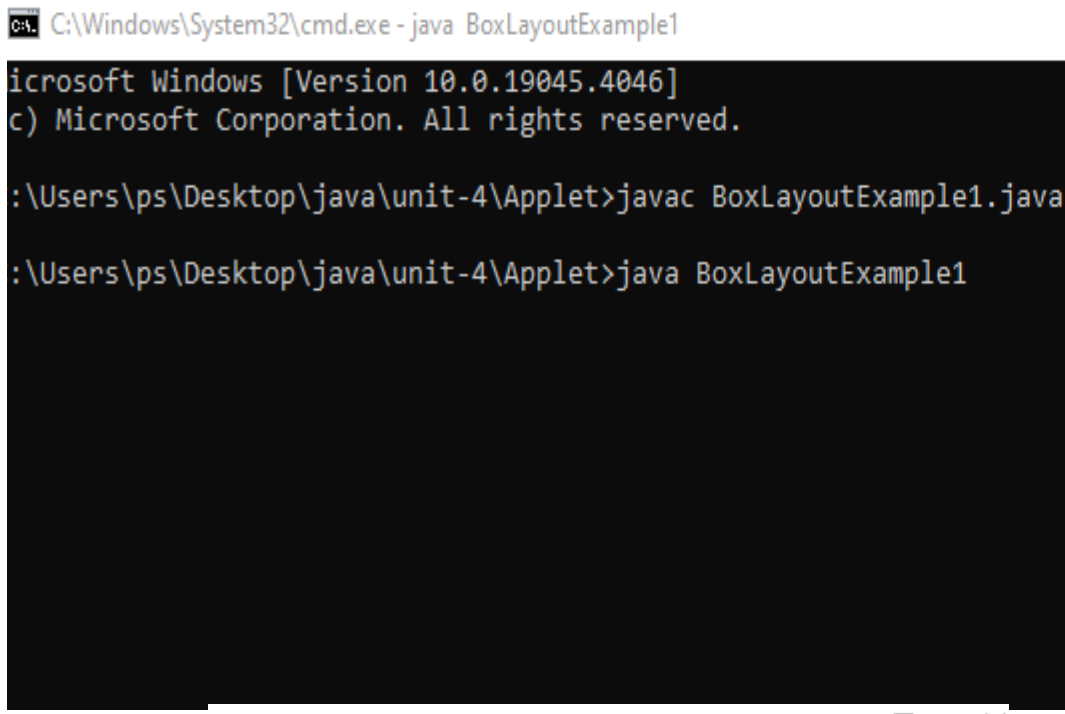
- **public static final int X_AXIS:** Alignment of the components are horizontal from left to right.
- **public static final int Y_AXIS:** Alignment of the components are vertical from top to bottom.
- **public static final int LINE_AXIS:** Alignment of the components is similar to the way words are aligned in a line, which is based on the ComponentOrientation property of the container.
- **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.

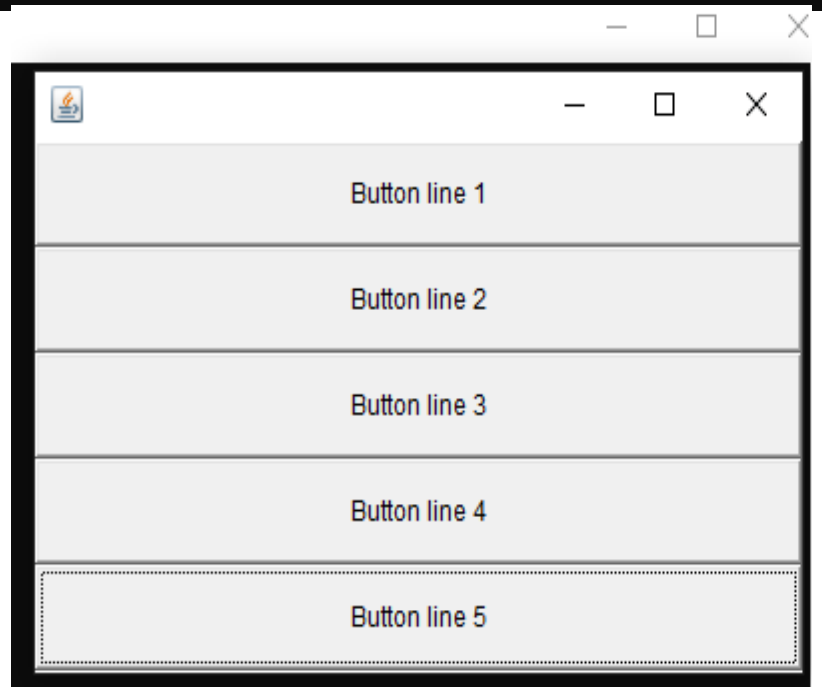
Constructor of BoxLayout class

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

```
import java.awt.*;
import javax.swing.*;
public class BoxLayoutExample1 extends Frame {
 Button buttons[];
 public BoxLayoutExample1 () {
   buttons = new Button [5];    for (int i = 0;i<5;i++) {
     buttons[i] = new Button("Button line " + (i + 1));
     // adding the buttons so that it can be displayed
     add (buttons[i]);
    }

setLayout (new BoxLayout(this, BoxLayout.PAGE_AXIS));
setSize(400,400);
setVisible(true);
}
  public static void main(String args[]){
BoxLayoutExample1 b=new BoxLayoutExample1();
}
}
```





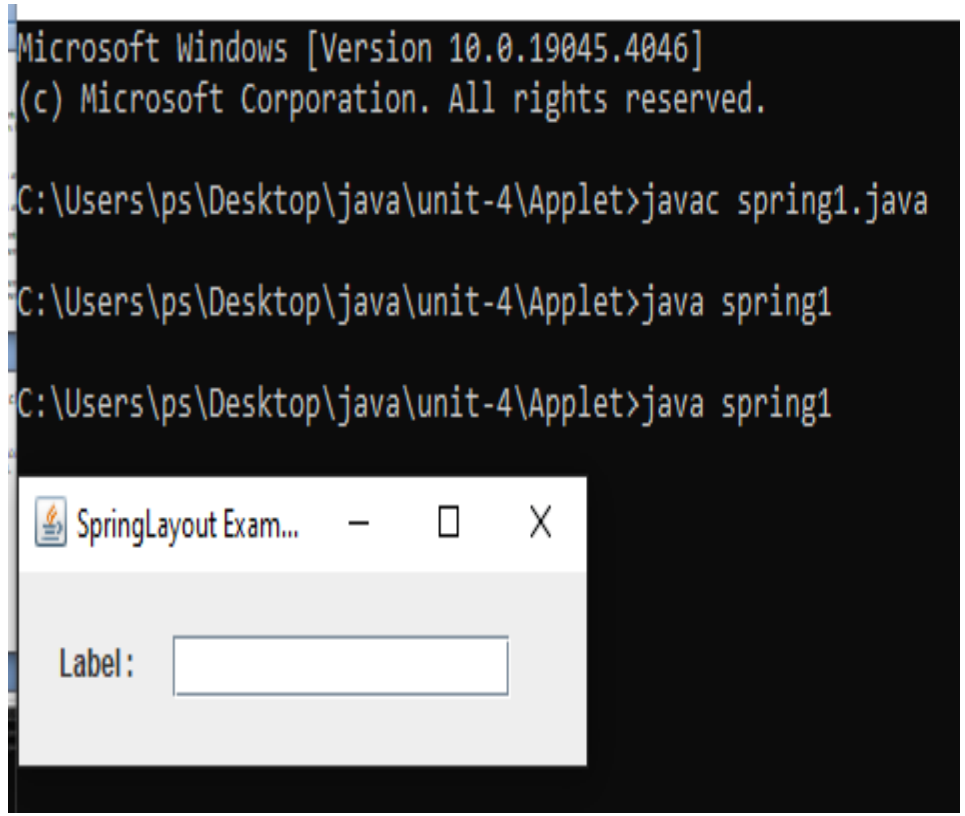
SpringLayout

- A SpringLayout arranges the children of its associated container according to a set of constraints. Constraints are nothing but horizontal and vertical distance between two-component edges. Every constraint is represented by a SpringLayout.Constraint object.
- Each child of a SpringLayout container, as well as the container itself, has exactly one set of constraints associated with them.
- Each edge position is dependent on the position of the other edge. If a constraint is added to create a new edge, than the previous binding is discarded. SpringLayout doesn't automatically set the location of the components it manages.

Constructor

- **SpringLayout():** The default constructor of the class is used to instantiate the SpringLayout class.

```
import java.awt.*;
import javax.swing.*;
public class spring1
{
  public static void main(String args[])
  {
    JFrame f = new JFrame("SpringLayout Example");
    Container content = f.getContentPane();
    SpringLayout layout = new SpringLayout();
    content.setLayout(layout);
    Component lab = new JLabel("Label :");
    Component text = new JTextField(15);
    content.add(lab);
    content.add(text);
    layout.putConstraint(SpringLayout.WEST, lab, 20, SpringLayout.WEST,
      content);
    layout.putConstraint(SpringLayout.NORTH, lab, 20, SpringLayout.NORTH,
      content);
    layout.putConstraint(SpringLayout.NORTH, text, 20, SpringLayout.NORTH,
      content);
    layout.putConstraint(SpringLayout.WEST, text, 20, SpringLayout.EAST,
      lab);
    f.setSize(300, 100);
    f.setVisible(true);
```



GroupLayout

- **GroupLayout** *groups its components and places them in a Container hierarchically.* The grouping is done by instances of the Group class.
- Group is an abstract class, and two concrete classes which implement this Group class are SequentialGroup and ParallelGroup.
- SequentialGroup positions its child sequentially one after another whereas ParallelGroup aligns its child on top of each other.
- The GroupLayout class provides methods such as createParallelGroup() and createSequentialGroup() to create groups.
- GroupLayout treats each axis independently. That is, there is a group representing the horizontal axis, and a group representing the vertical axis.

Nested Classes

Modifier and Type	Class	Description
static class	GroupLayout.Alignment	Enumeration of the possible ways ParallelGroup can align its children.
class	GroupLayout.Group	Group provides the basis for the two types of operations supported by GroupLayout: laying out components one after another (SequentialGroup) or aligned (ParallelGroup).
class	GroupLayout.ParallelGroup	It is a Group that aligns and sizes it's children.
class	GroupLayout.SequentialGroup	It is a Group that positions and sizes its elements sequentially, one after another.

Fields

Modifier and Type	Field	Description
static int	DEFAULT_SIZE	It indicates the size from the component or gap should be used for a particular range value.
static int	PREFERRED_SIZE	It indicates the preferred size from the component or gap should be used for a particular range value.

Constructors

GroupLayout(Container host)	It creates a GroupLayout for the specified Container.

null Layout

- null layout is not a real layout manager. It means that no layout manager is assigned and the components can be put at specific x,y coordinates.
- The "null layout manager" is also known as absolute positioning or NO Layout.
- When you are using a null layout manager, make sure that your component's size is big enough to be displayed properly on all platforms.