UNIT IV – AWT CONTROLS

AWT classes – Windows fundamentals – Working with frame windows - Control fundamentals - AWT containers and components - Layout managers – Menu bars and menus- Handling events by extending AWT Components.

LAYOUT MANAGERS

- Layout refers to the arrangement of components within the container. In another way, it could be said that layout is placing the components at a particular position within the container.
- The task of laying out the controls is done automatically by the Layout Manager.
- Even if you do not use the layout manager, the components are still positioned by the default layout manager.
- It is possible to lay out the controls by hand, however, it becomes very difficult because of the following two reasons.
- It is very tedious to handle a large number of controls within the container.
- Usually, the width and height information of a component is not given when we need to arrange them.
- Properties like size, shape, and arrangement varies from one layout manager to the Other. When the size of the applet or the application window changes, the size, shape, and arrangement of the components also changes in response, i.e. the layout managers adapt to the dimensions of the appletviewer or the application window.

Sr.No.	LayoutManager & Description
1	**BorderLayout** The borderlayout arranges the components to fit in the five regions: east, west, north, south, and center.
2	**CardLayout** The CardLayout object treats each component in the container as a card. Only one card is visible at a time.
3	**FlowLayout** The FlowLayout is the default layout. It layout the components in a directional flow.
4	**GridLayout** The GridLayout manages the components in the form of a rectangular grid.

5	**GridBagLayout** This is the most flexible layout manager class. The object of GridBagLayout aligns the component vertically, horizontally, or along their baseline without requiring the components of the same size.
6	**GroupLayout** The GroupLayout hierarchically groups the components in order to position them in a Container.
7	**SpringLayout** A SpringLayout positions the children of its associated container according to a set of constraints.

1) BorderLayout

The class BorderLayout arranges the components to fit in the five regions: east, west, north, south, and center. Package used is java.awt.BorderLayout.

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.
- JBorderLayout(int hgap, int vgap): creates a border layout with the given horizontal and vertical gaps between the components.

Program:

```
import java.awt.*;
import javax.swing.*;
public class Border
{
        JFrame f;
        Border()
        {
                f=new JFrame();
                JButton b1=new JButton("NORTH");;
                JButton b2=new JButton("SOUTH");;
                JButton b3=new JButton("EAST");;
                JButton b4=new JButton("WEST");;
                JButton b5=new JButton("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);
        }
        public static void main(String[] args)
        {
            new Border();
        }
}
```

Output:

2) GridLayout

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

Constructors of GridLayout class

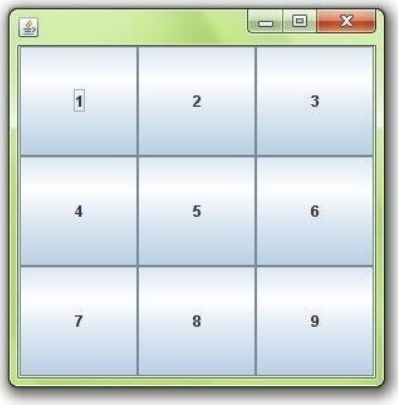
- GridLayout(): creates a grid layout with one column per component in a row.
- GridLayout(int rows, int columns): creates a grid layout with the given rows and
- columns but no gaps between the components.
- GridLayout(int rows, int columns, int hgap, int vgap): creates a grid layout with the given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

Program:

```
import java.awt.*;
import javax.swing.*;
public class MyGridLayout
```

```
{
	JFrame f;
	MyGridLayout()
	{
		f=new JFrame();
		JButton b1=new JButton("1");
		JButton b2=new JButton("2");
		JButton b3=new JButton("3");
		JButton b4=new JButton("4");
		JButton b5=new JButton("5");
		JButton b6=new JButton("6");
		JButton b7=new JButton("7");
		JButton b8=new JButton("8");
		JButton b9=new JButton("9");
		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(new GridLayout(3,3));
		f.setSize(300,300);
		f.setVisible(true);
	}
	public static void main(String[] args)
	{
		new MyGridLayout();
	}
}
```

Output:



3) FlowLayout

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

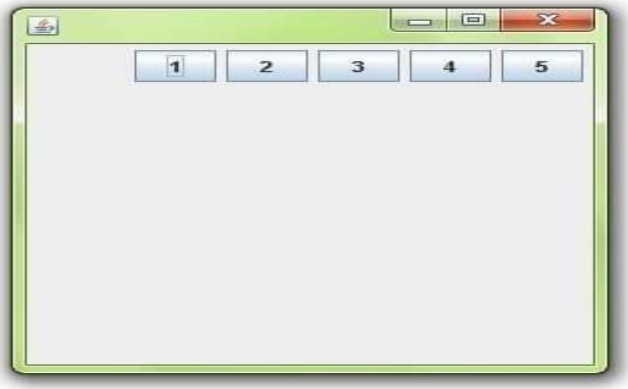
- 1. FlowLayout(): creates a flow layout with centered alignment and a default 5 Unit horizontal and vertical gap.
- 2. FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
- 3. FlowLayout(int align, int hgap, int vgap): creates a flow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

Program:

```
import java.awt.*;
import javax.swing.*;
public class MyFlowLayout
{
	JFrame f;
	MyFlowLayout()
	{
		f=new JFrame();
		JButton b1=new JButton("1");
		JButton b2=new JButton("2");
		JButton b3=new JButton("3");
		JButton b4=new JButton("4");
		JButton b5=new JButton("5");
		f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
		f.setLayout(new FlowLayout(FlowLayout.RIGHT));
		f.setSize(300,300);
		f.setVisible(true);
	}
	public static void main(String[] args)
	{
		new MyFlowLayout();
```

```
        }
}
```

Output:



4) BoxLayout

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

Fields of BoxLayout class

public static final int X_AXIS
public static final int Y_AXIS
public static final int LINE_AXIS
public static final int PAGE_AXIS

Constructor of BoxLayout class

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

Program:

```
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 " + (i + 1)); add (buttons[i]);
                }
                setLayout (new BoxLayout (this, BoxLayout.Y_AXIS)); setSize(400,400);
```

```
            setVisible(true);
        }
        public static void main(String args[])
        {
            BoxLayoutExample1 b=new BoxLayoutExample1();
        }
}
```

Output:

5) CardLayout

The 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

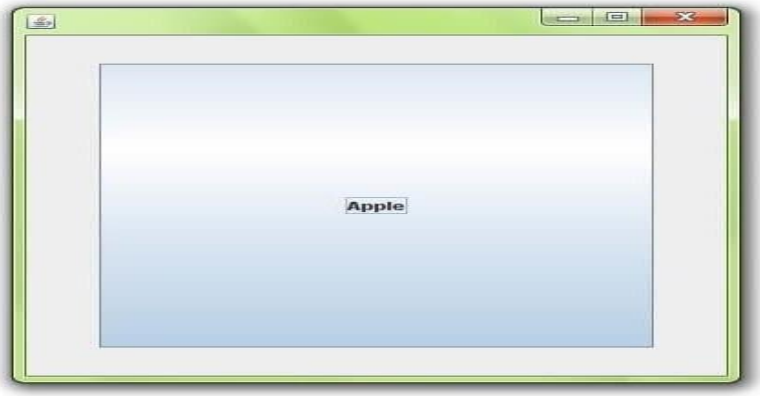
public void next(Container parent): is used to flip to the next card of the given container.
public void previous(Container parent): is used to flip to the previous card of the given container.
public void first(Container parent): is used to flip to the first card of the given container.
public void 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.

Program:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CardLayoutExample extends JFrame implements ActionListener
{
```

```
    CardLayout card;
    JButton b1,b2,b3;
    Container c;
    CardLayoutExample()
    {
        c=getContentPane();
        card=new CardLayout(40,30);
        c.setLayout(card);
        b1=new JButton("Apple");
        b2=new JButton("Boy");
        b3=new JButton("Cat");
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
        c.add("a",b1);c.add("b",b2);c.add("c",b3);
    }
    public void actionPerformed(ActionEvent e)
    {
        card.next(c);
    }
    public static void main(String[] args)
    {
        CardLayoutExample cl=new CardLayoutExample();
        cl.setSize(400,400);
        cl.setVisible(true);
        cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

Output:



6) ScrollPaneLayout

The layout manager used by JScrollPane. JScrollPaneLayout is responsible for nine components: a viewport, two scrollbars, a row header, a column header, and four "corner" components.

Fields

Modifier and Type	Field	Description
protected JViewport	colHead	It is column header child.
protected JScrollBar	hsb	It is scrollpane's horizontal scrollbar child.
protected int	hsbPolicy	It displays a policy for the horizontal scrollbar.
protected Component	lowerLeft	This displays the lower left corner.
protected Component	lowerRight	This displays in the lower right corner.
protected JViewport	rowHead	It is a row header child.
protected Component	upperLeft	This component displays in the upper left corner.
protected Component	upperRight	This component displays in the upper right corner.
protected JViewport	viewport	It is scrollpane's viewport child.
protected JScrollBar	vsb	It is scrollpane's vertical scrollbar child.
protected int	vsbPolicy	It is the display policy for the vertical scrollbar.

Useful methods

Modifier and Type	Method	Description

void	addLayoutComponent(String s, Component c)	It adds the specified component to the layout.
protected Component	addSingletonComponent(Component oldC, Component newC)	It removes an existing component.
JViewport	getColumnHeader()	It returns the JViewport object that is the column header.
Component	getCorner(String key)	It returns the Component at the specified corner.
JScrollBar	getHorizontalScrollBar()	It returns the JScrollBar object that handles horizontal scrolling.
int	getHorizontalScrollBarPolicy()	It returns the horizontal scrollbar-display policy.
JViewport	getRowHeader()	It returns the JViewport object that is the row header.
JScrollBar	getVerticalScrollBar()	It returns the JScrollBar object that handles vertical scrolling.
int	getVerticalScrollBarPolicy()	It returns the vertical scrollbar- display policy.
JViewport	getViewport()	It returns the JViewport object that displays the scrollable contents.

Program:

```
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
public class ScrollPaneDemo extends JFrame
{
        public ScrollPaneDemo()
        {
                super("ScrollPane Demo");
                ImageIcon img = new ImageIcon("child.png");
                JScrollPane png = new JScrollPane(new JLabel(img));
                getContentPane().add(png);
```

```
            setSize(300,250); setVisible(true);
        }
        public static void main(String[] args)
        {
            new ScrollPaneDemo();
        }
}
```

Output:

MENU BAR

JMenuBar, JMenu and JMenuItems are a part of Java Swing package. JMenuBar is an implementation of menu bar . the JMenuBar contains one or more JMenu objects, when the JMenu objects are selected they display a popup showing one or more JMenuItems . JMenu basically represents a menu . It contains several JMenuItem Object . It may also contain JMenu Objects (or submenu).

JMenuBar class declaration

public class JMenuBar extends JComponent implements MenuElement, Accessible

JMenu class declaration

public class JMenu extends JMenuItem implements MenuElement, Accessible

Constructors :

- JMenuBar() : Creates a new MenuBar.
- JMenu() : Creates a new Menu with no text.
- JMenu(String name) : Creates a new Menu with a specified name.
- JMenu(String name, boolean b) : Creates a new Menu with a specified name and boolean value specifies it as a tear-off menu or not. A tear-off menu can be opened and dragged away from its parent menu bar or menu.

Commonly used methods:

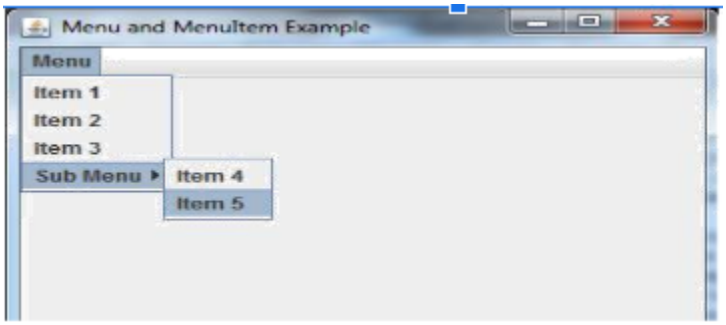
- add(JMenu c) : Adds menu to the menu bar. Adds JMenu object to the Menu bar.
- add(Component c) : Add component to the end of JMenu
- add(Component c, int index) : Add component to the specified index of JMenu
- add(JMenuItem menuItem) : Adds menu item to the end of the menu.
- add(String s) : Creates a menu item with specified string and appends it to the end of menu.
- getItem(int index) : Returns the specified menuitem at the given index

Program:

```
import javax.swing.*;
class MenuExample
{
	JMenu menu, submenu;
	JMenuItem i1, i2, i3, i4, i5;
	MenuExample()
	{
		JFrame f= new JFrame("Menu and MenuItemExample");
		JMenuBar mb=new JMenuBar();
		menu=new JMenu("Menu");
		submenu=new JMenu("Sub Menu");
		i1=new JMenuItem("Item 1");
		i2=new JMenuItem("Item 2");
		i3=new JMenuItem("Item 3");
		i4=new JMenuItem("Item 4");
		i5=new JMenuItem("Item 5");
		menu.add(i1); menu.add(i2); menu.add(i3);
		submenu.add(i4); submenu.add(i5);
		menu.add(submenu);
		mb.add(menu);
		f.setJMenuBar(mb);
		f.setSize(400,400);
		f.setLayout(null);
		f.setVisible(true);
	}
	public static void main(String args[])
	{
		new MenuExample();
	}
```

}

Output:

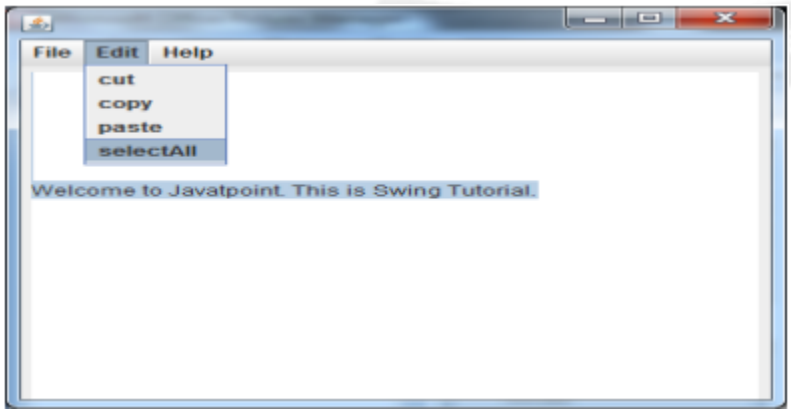


Example for creating Notepad:

```
import javax.swing.*;
import java.awt.event.*;
public class MenuExample implements ActionListener
{
        JFrame f;
        JMenuBar mb;
        JMenu file,edit,help;
        JMenuItem cut,copy,paste,selectAll;
        JTextArea ta;
        MenuExample()
        {
                f=new JFrame();
                cut=new JMenuItem("cut");
                copy=new JMenuItem("copy");
                paste=new JMenuItem("paste");
                selectAll=new JMenuItem("selectAll");
                cut.addActionListener(this);
                copy.addActionListener(this);
                paste.addActionListener(this);
                selectAll.addActionListener(this);
                mb=new JMenuBar();
                file=new JMenu("File");
                edit=new JMenu("Edit");
                help=new JMenu("Help");
                edit.add(cut);edit.add(copy);edit.add(paste);edit.add(selectAll);
                mb.add(file);mb.add(edit);mb.add(help);
                ta=new JTextArea();
```

```
            ta.setBounds(5,5,360,320);
            f.add(mb);f.add(ta);
            f.setJMenuBar(mb);
            f.setLayout(null);
            f.setSize(400,400);
            f.setVisible(true);
        }
        public void actionPerformed(ActionEvent e)
        {
            if(e.getSource()==cut)
                ta.cut();
            if(e.getSource()==paste)
                ta.paste();
            if(e.getSource()==copy)
                ta.copy();
            if(e.getSource()==selectAll)
                ta.selectAll();
        }
        public static void main(String[] args)
        {
            new MenuExample();
        }
}
```

Output:



ROHINI COLLEGE OF ENGINEERING AND TECHNOLOGY
PALKULAM, KANYAKUMARI
OBSERVE OPTIMIZE OUTSPREAD