Chapter 3 (AWT and Swing)

(Note: This document is dedicated for BE Computer 7th Sem. or BCA 5th Sem., so it is assumed that the already have knowledge what is GUI Programming. Further discussion on the example will be in class)

- AWT and Swing are two different classed used for develop GUI interfaces.
- All the part of GUI window are objects
- Every GUI is composed of three parts:
  - The Interface (actual windows)
  - The Engine (the underlying functionality of an application)
  - The Listener (the connection between the interface and the engine.)

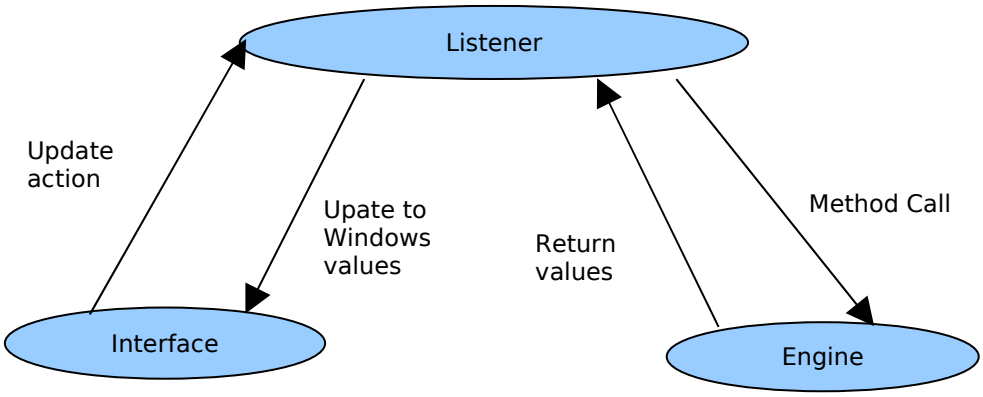


Fig: Working Model of GUI programming in Java

AWT

AWT stands for "Abstract Windowing Toolkit". It's main purpose is to provide the Java programmer the means needed to generate the UI (User Interface) for their application which will communicate with the user. The Abstract Window toolkit (AWT) contains numerous classes and methods that allow us to create and manage windows. It is used to support Applet window as well as GUI environment such as window. The AWT must reach into the GUI components of the native OS, which means that it performs a task that an applet cannot otherwise accomplish. An untrusted applet cannot make any direct calls into an OS because otherwise it could do bad things to the user's machine. The only way an untrusted applet can access important functionality such as "draw a window on the screen" is through calls in the standard Java library that's been specially ported and safety checked for that machine.

Swing

Swing is a set of classes that provides more powerful and flexible components that are possible with the AWT.

Benefits of Swing

- Swing components are Beans (and thus use the Java 1.1 event model), so they can be used in any development environment that supports Beans.
- Swing provides a full set of UI components.
- For speed, all the components are lightweight (no "peer" components are used),

- Swing is written entirely in Java for portability.
- Keyboard navigation is automatic – you can use a Swing application without the mouse, but you don't have to do any extra programming (the old AWT required some ugly code to achieve keyboard navigation).
- Scrolling support is effortless – you simply wrap your component in a **JScrollPane** as you add it to your form.
- Other features such as tool tips typically require a single line of code to implement.
- Swing also supports something called "pluggable look and feel," which means that the appearance of the UI can be dynamically changed to suit the expectations of users working under different platforms and operating systems. It's even possible to invent your own look and feel.
- If you've struggled long and hard to build your UI using Java 1.1, you don't want to throw it away to convert to Swing. Fortunately, the library is designed to allow easy conversion – in many cases you can simply put a 'J' in front of the class names of each of your old AWT components.

Difference between AWT and Swing

AWT	Swing
• It have to use OS peer for each components	• We would have only one peer, the operating system's window object. All of the buttons, entry fields, etc. are drawn by the Swing package on the drawing surface provided by the window object.
• It consist of less code but programmer have to code extensively	• Swing has more code and program don't have to code extensively
• AWT is a thin layer of code on top of the OS, whereas Swing is much larger	• Swing is much larger
• It has less functionalities	• Swing also has very much richer functionality
• It is called Heavy weight programming because all the components have their peer in OS.	• Swing components are called "lightweight" because they do not require a native OS object to implement their functionality. • JDialog and JFrame are heavyweight, because they do have a peer. So components like JButton, JTextArea, etc., are lightweight because they do not have an OS peer.
• For GUI-intensive work, AWT feels very primitive to work with compared to Swing	• Because Swing implements GUI functionality itself rather than relying on the host OS, it can offer a richer environment on all platforms Java runs on.
• AWT is more limited in supplying the same functionality on all platforms because not all platforms implement the same-looking controls in the same ways.	• Swing us platform independent so it in not limited in supplying the same functionality on all platforms.
• It has their own viewport which sends the output to the screen	• It does not write itself to the screen but redirect it to the component it builds on

• Its components also have their own z-ordering.	• Its components don't have z-ordering
• AWT and Swing in the same container. If you do, AWT will always be drawn on top of the Swing components.	• AWT and Swing in the same container. If you do, Swing will always be drawn on Below the Swing components.
• Its components are dependent to OS, so its appearance changes in different platform	• Its components are not dependent to OS, so its appearance remains same in all the platforms
• AWT supports Ctrl-C Ctrl-V copy/paste without any coding.	• In Swing, it requires page and pages of bubblegum. Further you can't do it at all in unsigned Swing JApplets.
• AWT is supported in older version of JVMs also.	• Swing is bigger and more complicated, and is not supported by earlier JVMs.
• With AWT you must not deal with the irritatingly pointless complexity of ContentPanes.	• With Swing you must deal with the irritatingly pointless complexity of ContentPanes.
• They are faster in operation because it uses OS components	• They are slower because they have to draw all components them self
• There are no equivalents in AWT.	• Swing has JSpinner, JSlider and JColorChooser.

Similarities between AWT and Swing

- There are no big architectural differences; the class hierarchy is almost the same. The reason is that
  - Swing is built upon AWT.
- Both AWT and Swing are used for GUI and window programming.

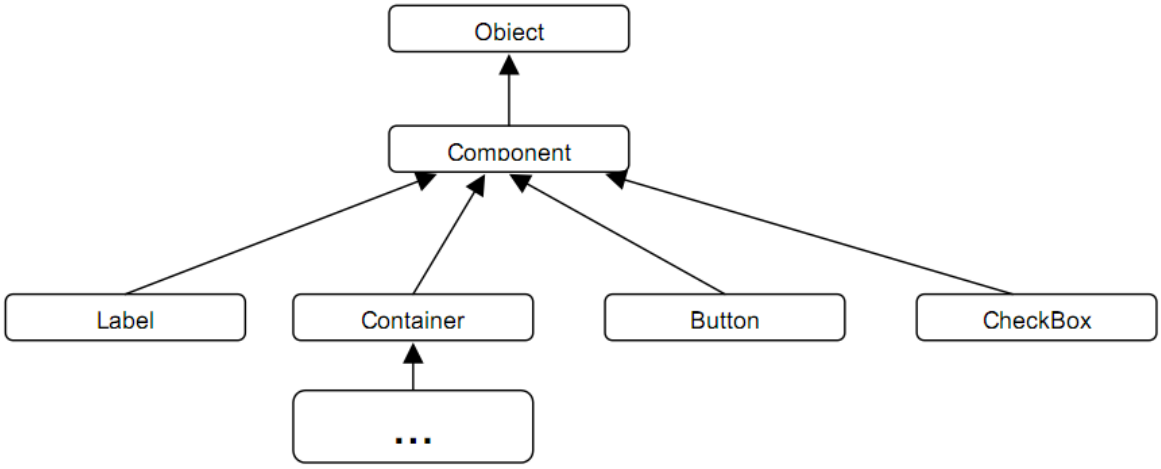


Fig: AWT Class Hierarchy

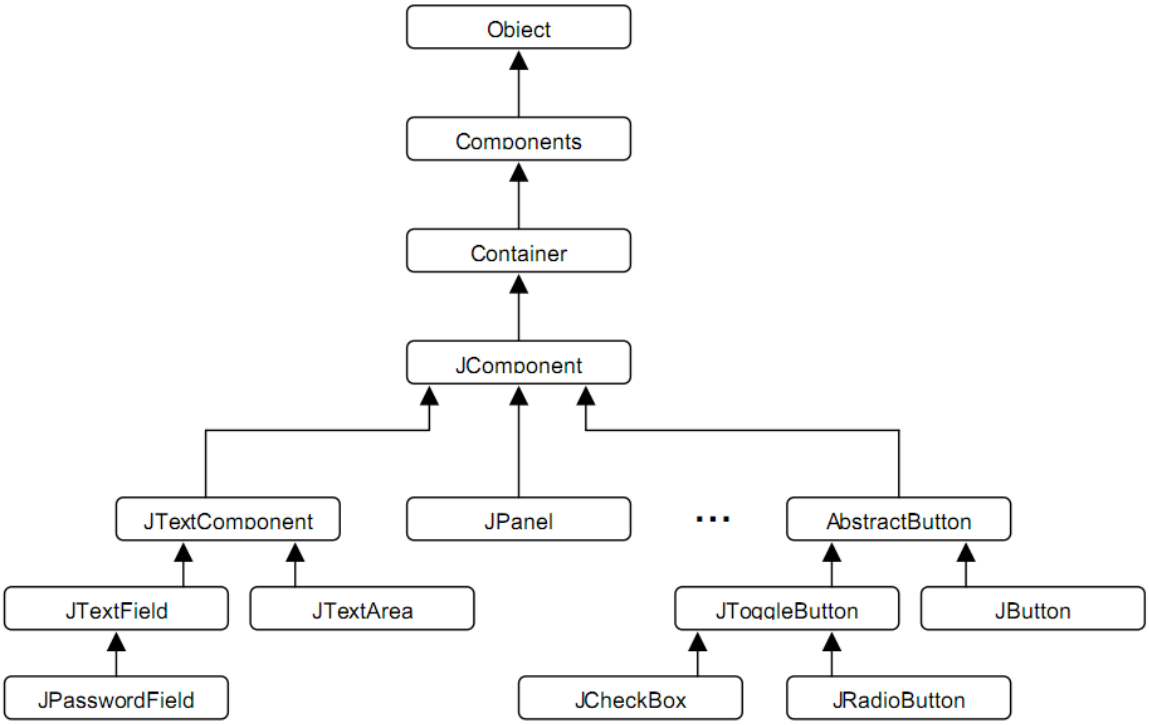


Fig: Swing Class Hierarchy

GUI Design Strategy
1. Identified Needed Components
2. Isolate regions of behavior
3. Sketch the GUI
4. Choose Layout Manager

Common Methods
- getSize() and setSize()
- getLocation() and setLocation()
- setVisible()
- setEnabled()
- setFont()

Basic swing Components
- Swing component classes can be found in javax.swing package.
- The component class name all begins with J.
- AWT and Swing components should not be combined.
- Non component object of AWT can be combined with Swing.
- Swing components are sub classes of java.awt.Container.

Type of Components
- Container Components (Applet, JFrame, JDialog)
- Ordinary Components or Atomic Components (JButton, JLabel etc.)
- Menu Components (Not included in course)

Container Component
- It can contain other components including other containers

- It uses layout managers for determining the size and position of child components.

JFrame

- Independent movable window.
- An application can contain 1 or more.
- Sample

```
public static void main(String[] args)
{
        JFrame f= new JFrame("Demo");
        f.setSize(300,400);
        f.setVisible(true);
}
```

Note: pack() method resizes the frame to fit it's content and choose appropriate Layout manager.

Example

```
public static void main(String args[])
{
        JFrame f = new JFrame ("demo");
        f.setSize(300,250);
        Container c = f.getContentPane();
        c.setLayout(new FlowLayout());
        c.add(new JButton("#1"));
        c.add(new JButton("#2"));
        f.setVisible(true);
}
```

JPanel

- In above example the Layout manager manages layout of container instead of JFrame.
- Blank rectangular component that can contain other components.
- Each panel uses a separate layout manager.
- By default JPanel used FlowLayout manager.
- Example

```
public static void main(String args[])
{
        JFrame f = new JFrame("Cantent pane");
        f.setSize(350,250);
        Container c=f.getContentPane();
        c.setLayout(new GridLayout(2,1));
        JPanel p= new JPanel();
        p.setBackground(Color.LightGray);
        p.add(new JButton("#1"));
        p.add(new JButton("#2"));
        c.add(p);
        p.setBackground(Color.LightGray);
        p.add(new JButton("#1"));
        p.add(new JButton("#2"));
        c.add(p);
        f.setVisible(true);
}
```

Ordinary (Atomic) Components

- JLabel
- JButton
- JCheckBox
- JRadioButton

- JTextField
- JTextArea
- JComboBox
- JScrollBar

JLabel

- Simplest component type which will display text and/or image.
- Do not respond to user imput and do not emit events.

```
public static void main(String args[])
{
        JFrame f = new JFrame("JLabel Demo");
        Container c = f.getContentPane();
        c.setLayout(new FlowLayout());
        c.add(new JLabel("This is JLabel");
        f.setVisible(true);
}
```

JButton

- Implements simple push button.
- Can display text/icon or both.
- On user click, action events are sent to all registered Action listeners.

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

public class ButtonDemo extends JFrame
{
        private JButton bntExit;
        public static void main(String args[])
{
        (new ButtonDemo()).setVisible(true);
}

ButtonDemo()
{
        super("Button Demo");
        setSize(350,250);
        Container c=getContentPane();
        c.setLayout(new FlowLayout());
        btnExit=new JButten("Exit");
        c.add(btnExit);

        ButtonListener listener = new ButtonListener();
        btnExit.addActionListener(listener);
}
class ButtonListener implements ActionListener
{
        public void actionPerformed(ActionEvent e)
        {
                if(e.getSource() == bntExit)
                        System.out.println("Hello");
        }
}
}
```

JCheckbox

- Implements a check box that can be selected or deselected.

```
import java.awt.*;
import javax.swing.*;
public class chkbxDemo{
        public static void main(String[] args){
                JFrame frame = new JFrame(“CheckBoxDemo”);
                frame.setSize(300,200);
                Container cont = frame.getContentPane();
                cont.setLayout(new FlowLayout());
                cont.add(new JCheckBox(“Apple”);
                cont.add(new JCheckBox9(“Banana”);
                cont.add(new Button(“Submit”);
                frame.setVisible(true);
        }
}
```

JRadioButton

- Implements a check box that can be selected or deselected.

```
import java.awt.*;
import javax.swing.*;
public class RadioDemo{
   public static void main(String[] args)    {
      JFrame frame = new JFrame("Radio Demo");
      frame.setSize(350,250);
      Container cont = frame.getContentPane();
      cont.setLayout(new FlowLayout());
      ButtonGroup btnGroup = new ButtonGroup();

     JRadioButton rbtnP = new                    JRadioButton("Programmer",true);
    JRadioButton rbtnD = new JRadioButton("Designer");
                btnGroup.add(rbtnP);
     btnGroup.add(rbtnD);

     cont.add(rbtnP);
     cont.add(rbtnD);

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

JScrollBar

- Lets the user enter an adjustable pseudo-analog value.
- Can be organized horizontally or vertically.

o JScrollBar.HORIZONTAL
o JScrollBar.VERTICAL

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

public class ScrollBarDemo extends JFrame{
   public static void main(String args[]){
      (new ScrollBarDemo()).setVisible(true);
```

```
}
ScrollBarDemo(){
   super("Scroll Bar Demo");
   setSize(350,250);
   Container cont = getContentPane();
   JScrollBar sbar = new JScrollBar(JScrollBar.HORIZONTAL);
   cont.add(sbar,BorderLayout.NORTH);

   BarListener listener = new BarListener();
   sbar.addAdjustmentListener(listener);
}
class BarListener implements AdjustmentListener{
   public void adjustmentValueChanged(AdjustmentEvent e){
      System.out.println("Val = " + e.getValue());
   }
}
}
```

JTextField and JTextArea

- Single line and multi-line text entry.
- Both are inherited javax.swing.text.JTextComponent.
- Both sends Key events when they receive keyboard input.
- In addition Action events occurs while pressed Enter.

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

public class TextDemo extends JFrame implements ActionListener, KeyListener{
   private JTextField field;
   private JTextArea area;
public static void main(String args[]){
      (new TextDemo()).setVisible(true);
   }

   TextDemo(){
      super("TextDemo");
      setSize(350,250);
      Container cont = getContentPane();

      field = new JTextField("Type here");
      field.addKeyListener(this);
      field.addActionListener(this);
      cont.add(field,BorderLayout.NORTH);

      area=new JTextArea();
      cont.add(area,BorderLayout.CENTER);
   }
public void keyPressed(KeyEvent e){
      System.out.println(e);
   }

   public void keyReleased(KeyEvent e){}

   public void keyTyped(KeyEvent e){
      area.append("Key: " + e.getKeyChar() + "'");
   }
```

```
   public void actionPerformed(ActionEvent e){
      area.append("Action: " + field.getText() + "'");

   }
}
```

JComboBox

```
ComboDemo()
   {
      super("ComboDemo");
      setSize(350,250);
      Container cont = getContentPane();
      cont.setLayout(new FlowLayout());

      String[] initialVals = {"BCA","BE Comp","BIT"};
      JComboBox combo = new JComboBox(initialVals);
      combo.setEditable(true);
      cont.add(combo);
   }
}
```

Layout Management

- A layout manager determines the location and size of components placed into a container.
- Different layout manager classes use different algorithms for determining size and location.
- The only layout manager methods that are normally used are constructors.
- In Java we normally do not manually manage layout, because there are two main reasons
  - It is very tedious to manually layout a large number of components.
  - Sometimes the width and height information is not yet available when we need to arrange some control, because it is pretty confusing to figure out when it is okay to use the size of a given component to position it relative to another.
- A layout manager is an instance of any class that implements the **LayoutManager** interface.
- If no call to **setLayout( )** is made, then the default layout manager is used.
- Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it.

The **setLayout( )** method has the following general form:

Void setLayout(LayoutManager layoutObj)

- Here, layoutObj is a reference to the desired layout manager.
- If we wish to disable the layout manager and position components manually, pass null for layoutObj.
- In this case, we have to determine the shape and size of each component manually, using the **setBounds( )**.

- Each component that is being managed by a layout manager contains the **getPreferredSize**( ) and **getMinimumSize**( ) methods.

- These return the preferred and minimum size required to display each component.

FlowLayout Class

- It places components in rows.
- Each component is given its preferred size. When a row has as many components as can fit, a new row is started.
- This is the default layout manager for JPanel and Applet.
- Components are laid out from the upper-left corner, left to right and top to bottom.
- It uses the constructor to tune the behavior.

There are three useful constructors.

- **new FlowLayout( )**
- **new FlowLayout(int algn)**
- **new FlowLayout(int algn, int hgap, int vgap)**

- The first form creates the default layout, which centers components and leaves five pixels of space between each component.
- The second form lets us specify how each line is aligned.

FlowLayout.LEFT
FlowLayout.RIGHT
FlowLayout.CENTER

- The third form allows us to specify the horizontal and vertical space left between components in hgap and vgap

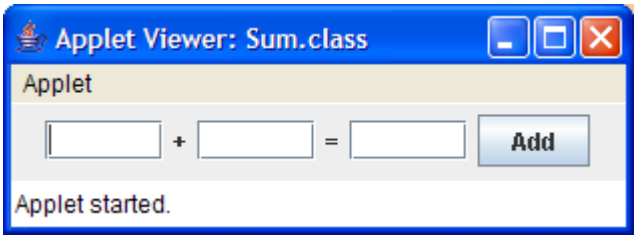


Fig: Initial State of Applet.

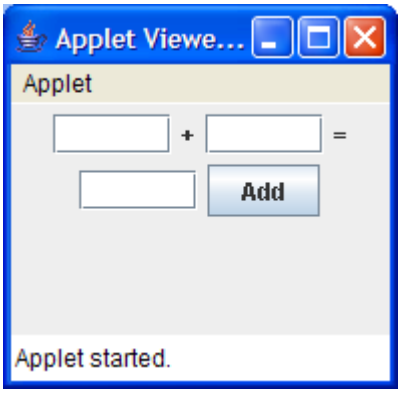


Fig: After resize (decrease the width and increase the height)

BorderLayout Class

- It divides its container into three horizontal stripes:
  - top (north),
  - middle, and
  - bottom (south).
- At most one component can be placed into the north or south.
- These components get their preferred height, but they take up the full width of the container.
- The middle stripe gets whatever height is available after deducting the heights of the north and south components.
- It can hold three components:
  - one on the left (west),
  - one in the middle (center), and
  - one on the right (east).
- The east and west components get their preferred widths, but they take up the full height of the middle stripe.
- This is the default layout manager for content panes (JFrame,JDialog,Window).
- There are two useful constructors.

  - **new BorderLayout( )**
  - **new BorderLayout(int hgap, int vgap)**

- When components are added to a container that uses a BorderLayout, they should be added with the following kind of statement.

container.add(component, *whr*);

- The parameter *whr* should be one of the following.
  - **BorderLayout.NORTH**
  - **BorderLayout.EAST**
  - **BorderLayout.SOUTH**
  - **BorderLayout.WEST**
  - **BorderLayout.CENTER**

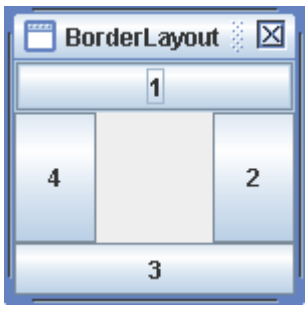


Fig: Initial State

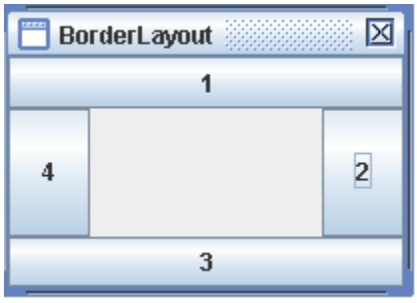


Fig: resizing horizontally

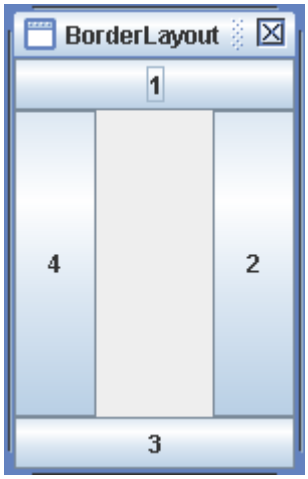
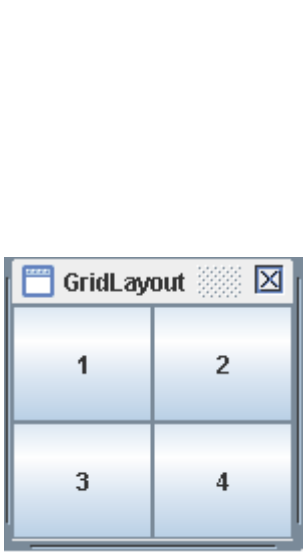


Fig: resizing vertically

GridLayout Class

- It places components in a rectangular grid whose cells all have the same size. Each component is sized to fill the cell.
- There are two useful GridLayout constructors.
  - **new GridLayout(int rows, int cols)**
  - **new GridLayout(int rows, int cols, int hgap, int vgap)**



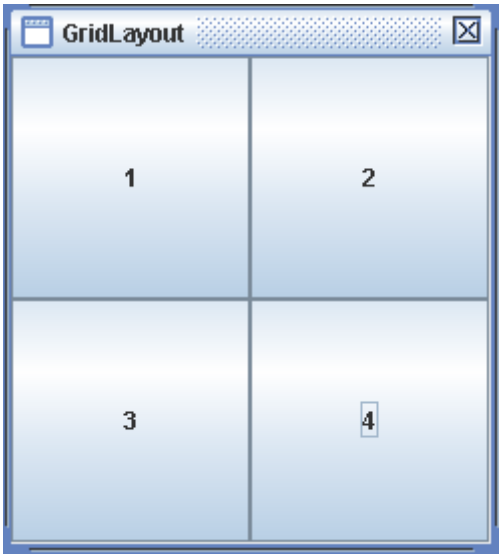
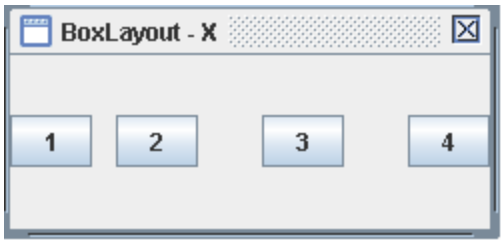


Fig:Initial State Fig: after resizing (in diagnol increase height and width both)

(Example already discussed above)

BoxLayout Class

- It places components into a single row or column, as specified by the second constructor parameter.
- For a row (parameter value BoxLayout.X_AXIS), each component get its preferred height.
- The widths are adjusted according to a formula that incorporates both preferred and maximum widths.
- For a column (parameter value BoxLayout.Y_AXIS), each component gets its preferred width.
- The heights are adjusted according to a formula that incorporates both preferred and maximum heights.
  - **new BoxLayout(Component c, BoxLayout.X_AXIS)**
  - **new BoxLayout(Component c, BoxLayout.Y_AXIS)**



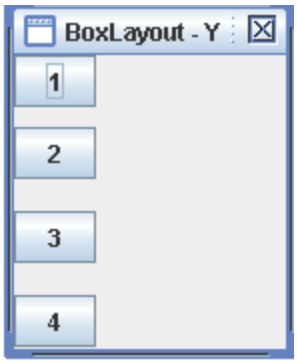


Fig: XboxLayout Fig: yBoxlayout

CardLayout

- It is unique among the other layout managers in that it stores several different layouts.
- Each layout can be thought of as being on a separate index card in a deck that can be shuffled so that any card is on top at a given time.
- This can be useful for user interfaces with optional components that can be dynamically enabled and disabled upon user input.
- We can prepare the other layouts and have them hidden, ready to be activated when needed.

o CardLayout provides these two constructors:

CardLayout( )
CardLayout(int horz, int vert)

- The first form creates a default card layout.
- The second form allows us to specify the horizontal and vertical space left between components in horz and vert, respectively.
- Use of a card layout requires a bit more work than the other layouts.
- The cards are typically held in an object of type Panel.
- This panel must have CardLayout selected as its layout manager.
- The cards that form the deck are also typically objects of type Pane.
- Thus we must create a panel that contains the deck and a panel for each chard in the deck.
- Next, we add to the appropriate panel the components that form each card.
- We then add panel to the main applet panel.
- Once these steps are complete, we must provide some way for the user to select between cards.
- Once these steps are complete, we must provide some button for each card in the deck.
- When card panels are added to a panel, they are usually given a name.

GridBagLayout

- GridBagLayout is the most sophisticated, flexible layout manager the Java platform provides.

- Grid bag layouts are the most complex and most flexible of the AWT layouts, letting you specify more exactly than any of the other AWT layout managers where you want your components to go.

Assignment:

Write programs to demonstrate GridBagLayout and CardLayout.

Hints:
Students should be familier with AWT, Swing, JFrame, JPanel, JTextField, JLabel, JCheckBox, JRadioButton and JTextArea along with event handling.

In Layout Management students should practice on FlowLayout, GridLayout, BorderLayout and combination

Extra Stuff (not in Syllabus)

JMenu
- Allow the programmer to organize Swing components in menus.
- JMenuBar component implements a menu bar that occupies the top portion of a JFrame and can contain drop-down menus.
- JFrame calls setJMenuBar(theMenuBar) to insert menu.
- to populate a menu bar, construct a number of instances of JMenu and install them in the menu bar by calling the theMenuBar.add(theMenu).
- The JMenu constructor takes as an argument the string that will appear on the menu bar.
- Example

```
//MenuDemo.java
// Title : MenuDemo.java - Simple demo of building menus.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class MenuDemo {
   public static void main(String[] args) {
      JFrame win = new MenuDemoGUI();
      win.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      win.setVisible(true);
   }
}
//MenuDemoGUI
class MenuDemoGUI extends JFrame {
   JTextArea m_editArea = new JTextArea(20, 50);

         // declare and create new menu
   JMenu     m_fileMenu = new JMenu("File");
         // create new menu item
         JMenuItem m_openItem = new JMenuItem("Open");
         //another menu item
         JMenuItem m_quitItem = new JMenuItem("Quit");
   JMenu     m_editMenu = new JMenu("Edit");
   JMenuItem m_copyItem = new JMenuItem("Copy");
   JMenuItem m_pasteItem= new JMenuItem("Paste");
```

```
//constructor
    public MenuDemoGUI() {
       //... Add listeners to menu items
       m_openItem.addActionListener(new OpenAction());
       m_quitItem.addActionListener(new QuitAction());

       JMenuBar menubar = new JMenuBar();  // declare and create new menu bar
     menubar.add(m_fileMenu);
          m_fileMenu.add(m_openItem);
          m_fileMenu.addSeparator();  // add separator line to menu
          m_fileMenu.add(m_quitItem);
          menubar.add(m_editMenu);
     m_editMenu.add(m_copyItem);
     m_editMenu.add(m_pasteItem);
//... Content pane: create, layout, add components
       JPanel content = new JPanel();
       content.setLayout(new BorderLayout());
       content.add(m_editArea, BorderLayout.CENTER);

       //... Set JFrame's menubar, content pane, and title.
       this.setContentPane(content);       // Set windows content pane..
       this.setJMenuBar(menubar);          // Set windows menubar.
       this.pack();
       this.setTitle("MenuDemo");
    }//endconstructor

//// OpenAction
    class OpenAction implements ActionListener {
       public void actionPerformed(ActionEvent e) {
          JOptionPane.showMessageDialog(null, "Sorry, can't open anything");
       }
    }

/// QuitAction
    class QuitAction implements ActionListener {
       public void actionPerformed(ActionEvent e) {
          System.exit(0);  // terminate this program
       }
    }
}
```

Example of uing JTable

```
import java.awt.event.WindowAdapter;
```

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

public class jTableEx extends JFrame
{
   String data[][]={{"Shiba","Bagbazar","BCA Honors"},{"Roshan","BE Elec and
e","Lazimpat"}};

   String fields[]={"Name","Address","Education"};

   public static void main( String[] args)
   {
      jTableEx myEx = new jTableEx("JTAble Example");
   }

   public jTableEx( String title )
   {
      super ( title );
      setSize(150,150);
      addWindowListener ( new WindowAdapter()
       { public void windowClosing( WindowEvent we ){
         dispose();
         System.exit( 0 );
       }
      });
      init();
      pack();
      setVisible (true);

   }
   private void init()
   {
      JTable jt = new JTable (data,fields);
      JScrollPane pane = new JScrollPane ( jt );
      getContentPane().add( pane);
   }
}
```

Example of JList

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

public class JListEx extends JFrame
{
   private DefaultListModel nameList;
   private JList list;

   public JListEx()
   {
      nameList = new DefaultListModel();
      nameList.addElement("Ramesh");
      nameList.addElement("Santosh");
      nameList.addElement("Prakash");

      list = new JList ( nameList);
```

```
list.setSelectionMode( ListSelectionModel.SINGLE_SELECTION);

//create JButton ( "Add Name");
JButton addButton = new JButton ("Add Name");
addButton.addActionListener(
   new ActionListener()
   {
      public void actionPerformed(ActionEvent event)
      {
         //prompt user for name
         String name = JOptionPane.showInputDialog(JListEx.this,"Enter Name");

         //add new Name to model
         nameList.addElement(name);
      }
   });

   //create JButton for removing selected name
   JButton removeButton = new JButton("Remove Selected Name");

   removeButton.addActionListener(
      new ActionListener(){
         public void actionPerformed (ActionEvent event )
         {
            nameList.removeElement( list.getSelectedValue());
         }
      }
      );


      //lay out GUI Components
      JPanel inputPanel = new JPanel();
      inputPanel.add( addButton );
      inputPanel.add( removeButton );

      Container container = getContentPane();
      container.add(list,BorderLayout.CENTER);
      container.add( inputPanel,BorderLayout.NORTH);

      setDefaultCloseOperation(EXIT_ON_CLOSE);
      setSize(400,300);
      setVisible(true);
   } //end of JListEx constructor

   public static void main(String args[])
   {
      new JListEx();
   }
}
```