
Java Programming
TEK RAJ CHHETRI

Course Objective

- Explain the Java programming environment
- Describe the concepts of programming elements using Java and object-oriented
- programming concepts
- Apply the exception handling and input/output in Java programming
- Apply the event handling, GUI programming using swing, and Java database connectivity

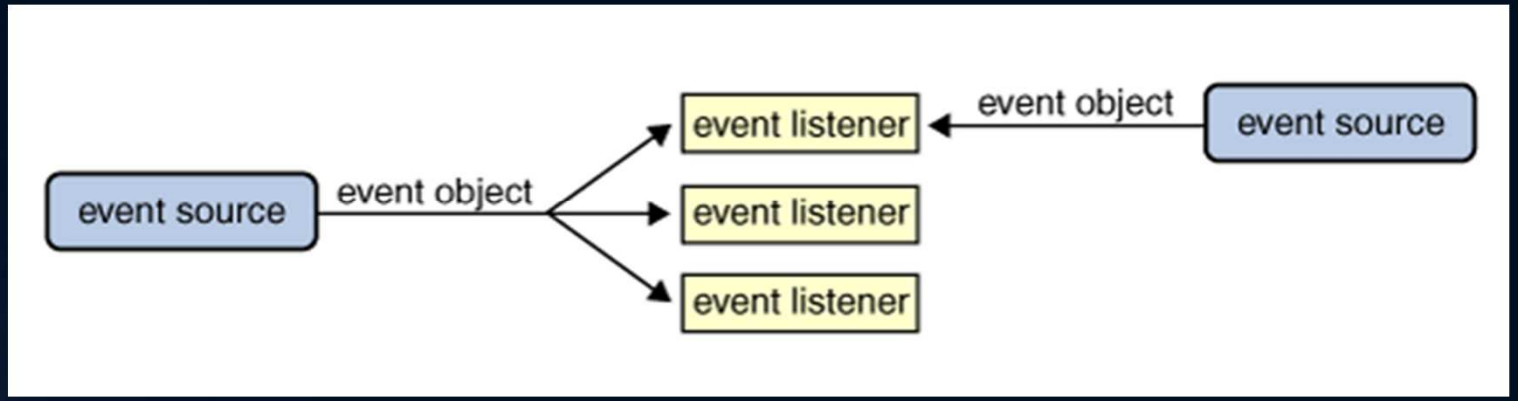
Unit 7: Event Handling and User Interface Components with Swing

- Basics of Event Handling
- Event Classes
- Event Listeners and Adapter Classes
- Swing and the MVC Design Pattern
- Layout Management
- Basic Swing Components

Learning Outcome (Unit 7)

- Develop understanding about event handling and MVC design pattern.
- Ability to apply event handling
- Develop GUI application using Swing

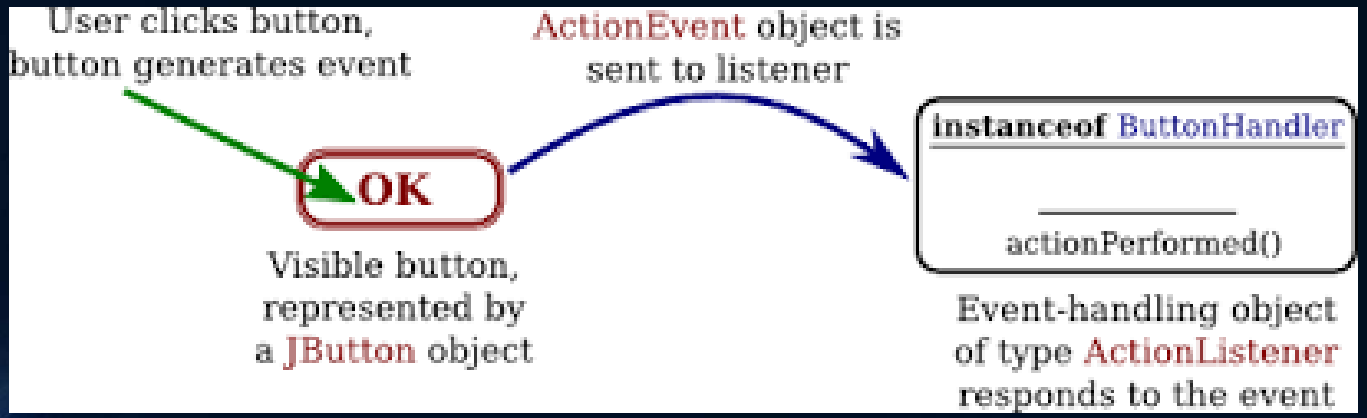
Basics of Event Handling


event source
event object
event listener
event listener
event listener
event object
event source

- Event:
  - In delegation model, an event is an object that describes state change in a source.
  - Eg:
  - Key event, mouse event → mouse pressed, dragged etc

- Event Source:
  - A source is an object that generates an event.
  - A source must register listener.
  - Each type of event has its own listener.
  - Eg:
    - Button, Checkbox, Choice,List, Menu Item, Scrollbar, Text Components, Window
  - **General form**:
    - public void addTypeListener (TypeListener el)
  - When an event occurs, all registered listeners are notified and receive a copy of the event object.
  - It is called Multicasting the event.
  - Some sources may allow only one listener to register.
  - **General form**:
    - public void addTypeListener (TypeListener el) throws java.util.TooManyListenersException
  - **To remove Listener**
    - Public void removeTypeListener (TypeListener el)

- Event Listener:
  - Listener is an event that is notified when event occurs.
  - Has two requirements:
    1. Must be registered with one or more sources.
    2. Must implement methods to receive and process these notifications.



- **ActionListener**
  - void actionPerformed (ActionEvent ae)
- **AdjustmentListener**
  - void adjustmentValueChanged (AdjustmentEvent ae)
- **ComponentListener**
  - void componentResized (ComponentEvent ce)
  - void componentMoved (ComponentEvent ce)
  - void componentShown (ComponentEvent ce)
  - void componentHidden (ComponentEvent ce)

- **ContainerListener**
  - void componentAdded (ContainerEvent ce)
  - void componentRemoved (ContainerEvent ce)
- **FocusListener**
  - void focusGained (FocusEvent fe)
  - void focusLost (FocusEvent fe)
- **ItemListener**
  - void itemStateChanged (ItemEvent ie)

- **KeyListener**
  - void keyPressed (KeyEvent ke)
  - void keyReleased (KeyEvent ke)
  - void keyTyped (KeyEvent ke)
- MouseListener
  - void mouseClicked (MouseEvent me)
  - void mouseEntered (MouseEvent me)
  - void mouseExited (MouseEvent me)
  - void mousePressed (MouseEvent me)
  - void mouseReleased (MouseEvent me)

- **MouseMotionListener**
  - void mouseDragged (MouseEvent me)
  - void mouseMoved (MouseEvent me)
- **TextListener**
  - void textChanged (TextEvent te)
- **WindowListener**
  - void windowActivated (WindowEvent we)
  - void windowClosed (WindowEvent we)
  - void windowClosing (WindowEvent we)
  - void windowDeactivated (WindowEvent we)
  - void windowDeiconified (WindowEvent we)
  - void windowIconified (WindowEvent we)
  - void windowOpened (WindowEvent we)

KeyBoard Event

```
1.  import java.applet.Applet;
2.  import java.awt.*;
3.  import java.awt.event.KeyEvent;
4.  import java.awt.event.KeyListener;
5.  /**
6.  * Java program to demonstrate key event handlers
7.   * To run this program in commandline compile this file as normal java file
8.   * then you could follow one of the two method to run
9.     * Put this code <applet code="KeyBoardEvent" width=300 height=300></applet> in html file and save it. And use appletviewer
    filename.html for output
10.    * Or use appletviewer classname.java
11.  */
12. /*
13.    <applet code="KeyBoardEvent" width=300 height=300></applet>
14. */
15. public class KeyBoardEvent extends Applet implements KeyListener{
16.    String message = "";
17.    int x = 10, y = 20;
```

```
18.     public void  init(){
19.       addKeyListener( this );
20.     }
21.     public void keyPressed(KeyEvent ke){
22.       showStatus( "Key Down" );
23.     }
24.     public void  keyReleased(KeyEvent ke){
25.       showStatus( "Key Released" );
26.     }
27.     public void keyTyped(KeyEvent ke){
28.       message += ke.getKeyChar();
29.       repaint();
30.     }
31.     //display key stroke
32.     public void paint(Graphics graphics){
33.       graphics.drawString(message,x,y);
34.     }
35.   }
```

Mouse Event Example

```
1.  import java.applet.Applet;
2.  import java.awt.*;
3.  import java.awt.event.*;
4.  /**
5.   * java program to demonstrate mouse event
6.   <applet code="MouseEvents" width=100 height=100></applet>
7.   */
8.  public class MouseEvents extends Applet implements MouseListener, MouseMotionListener {
9.     String message = "";
10.    int xPosition = 0, yPosition = 0;

11.    public void init(){
12.       addMouseListener( this );
13.       addMouseMotionListener( this );
14.    }
15.    //mouse clicked
16.    public void mouseClicked(MouseEvent mouseEvent){
```

```
17.         //update co-ordinate
18.         xPosition = 10;
19.         yPosition = 20;
20.         message = "Mouse Clicked";
21.         repaint();
22.       }
23.       //mouse entered
24.       public void mouseEntered(MouseEvent mouseEvent){
25.         //update co-ordinate
26.         xPosition = 10;
27.         yPosition = 20;
28.         message = "Mouse Entered";
29.         repaint();
30.       }
31.       //mouse exited
32.       public void mouseExited(java.awt.event.MouseEvent mouseEvent){
33.         //update co-ordinate
34.         xPosition = 10;
35.         yPosition = 20;
36.         message = "Mouse exited";
```

```
37.         repaint();
38.       }
39.       //mouse  pressed
40.       public void  mousePressed(MouseEvent mouseEvent){
41.         //update co-ordinate
42.         xPosition = mouseEvent.getX();
43.         yPosition = mouseEvent.getY();
44.         message = "Mouse Down";
45.         repaint();
46.       }
47.       //mouse button released
48.       public void mouseReleased(MouseEvent mouseEvent){
49.         //update co-ordinate
50.         xPosition = mouseEvent.getX();
51.         yPosition = mouseEvent.getY();
52.         message = "Mouse Released";
53.         repaint();
54.       }
55.       //mouse dragged
56.       public void mouseDragged(MouseEvent mouseEvent){
```

```
57.         //update co-ordinate
58.         xPosition = mouseEvent.getX();
59.         yPosition = mouseEvent.getY();
60.         showStatus( "Dragging mouse at "+xPosition+" , "+yPosition );
61.         repaint();
62.       }
63.       //mouse moved
64.       public void mouseMoved(MouseEvent mouseEvent){
65.         //update co-ordinate
66.         xPosition = mouseEvent.getX();
67.         yPosition = mouseEvent.getY();
68.         showStatus( "Moving mouse at "+xPosition+" , "+yPosition );
69.         repaint();
70.       }
71.       public void paint(Graphics graphics){
72.         graphics.drawString( message, xPosition, yPosition );
73.       }
74.   }
```

Event Class

Main Event Classes in java.awt.event

Event Class	Description
ActionEvent	Generated when a button is pressed, a list is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract super class for all component input event classes.
ItemEvent	Generated when a check box or a list item is clicked; also occurs when a choice selection is made or a checkable menu is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
TextEvent	Generated when the value of a textarea or textfield is changed.
WindowEvent	Generated when a window os activated, closed, deactivated, deiconified, iconified, opened, or quit.

Main Event Classes in java.awt.event

Event Class	Description
ActionEvent	Generated when a button is pressed, a list is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract super class for all component input event classes.
ItemEvent	Generated when a check box or a list item is clicked; also occurs when a choice selection is made or a checkable menu is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
TextEvent	Generated when the value of a textarea or textfield is changed.
WindowEvent	Generated when a window os activated, closed, deactivated, deiconified, iconified, opened, or quit.

Adapter Class

- We must implement all the methods define by interface even though we don’t require.
- This can be troublesome and time consuming as we implement those methods that are not require.
- But java provides solution to this using Adapter class.
- An adapter class provides empty implementation of all the methods.
- The adapter class are different for different listener.

Adapter Class	Listener Interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  import javax.swing.*;
4.  public class AdapterDemo extends MouseMotionAdapter{
5.     JFrame frame;
6.     AdapterDemo(){
7.        frame=new JFrame("Mouse Motion Adapter");
8.        frame.addMouseMotionListener(this);
9.        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10.
11.       frame.setSize(300,300);
```

```
12.         frame.setLayout(null);
13.         frame.setVisible(true);
14.     }
15.  public void mouseDragged(MouseEvent e) {
16.     Graphics g=frame.getGraphics();
17.     g.setColor(Color.RED);
18.     g.fillOval(e.getX(),e.getY(),20,20);
19.  }
20.  public static void main(String[] args) {
21.     new AdapterDemo();
22.  }
23.  }
```

Swing and the MVC Design Pattern

Swing:

- Part of the Java Foundation Classes (JFC)
- Provides a rich set of GUI components
- Used to create a Java program with a graphical user interface (GUI)
  - table controls, list controls, tree controls, buttons, and labels, and so on…
- The Swing API provides many different classes for creating various types of user interface elements
- Three classes: JFrame, JPanel, and JLabel are part of a larger collection of classes related through inheritance.

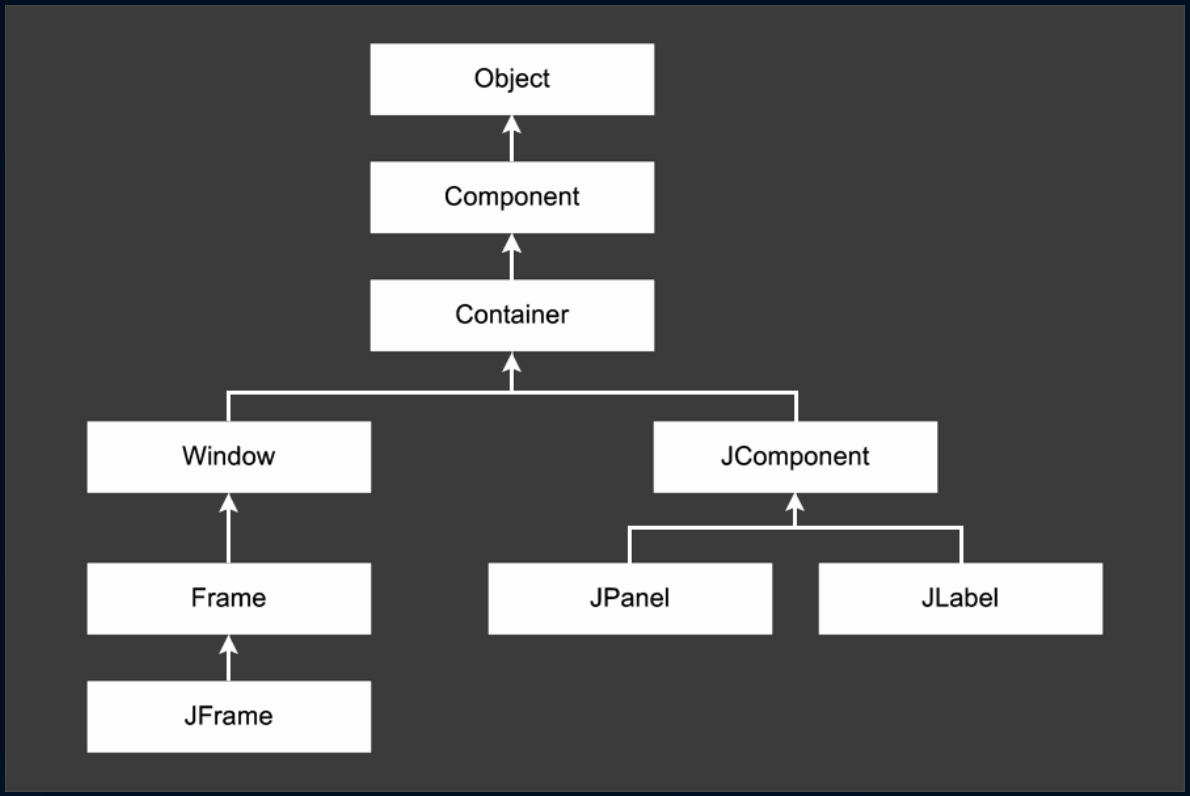


Fig: Swing Class Hierarchy.

- **Object**:
  - The top of the tree.

- **Component**:
  - Represents an object that has a visual representation that can be shown on-screen and that can interact with users.
  - This class defines some basic methods that are available to all Swing classes.

- **Container**:
  - Builds on the basic visual capabilities of the Component class by adding the ability to hold other containers.

- **Window**:
  - A specialized type of container object that has a border, a title bar, buttons that minimize, maximize, and close the window, and that can be repositioned and possibly even resized by the user.

- **Frame**:
  - A type of Window that serves as the basis for Java GUI applications. Frame is an AWT class that has been improved upon by the JFrame class.

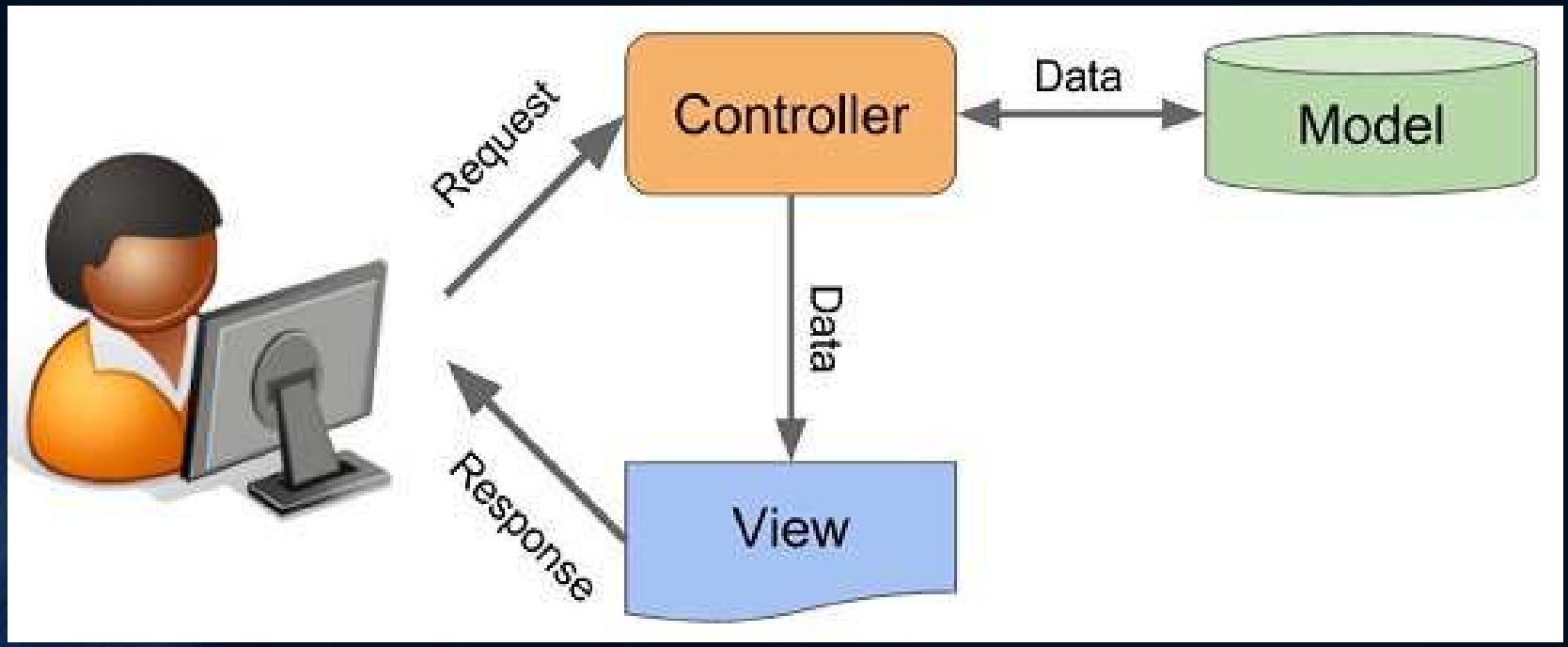
- **JFrame**:
  - The Swing version of the older Frame class.

- **JComponent**:
  - Is the basis for all other Swing components except for frames.

- **JPanel**:
  - Used to organize and control the layout of other components such as labels, buttons, text fields, etc.

- **JLabel**:
  - Creates a label that displays a simple text value.

MVC – Model View Controller:



```java
/*
    *This is model which handles the data
*/
public class Model{

    private String greetTo=" ";

    public Model(String greetValue){
        this.greetTo  = greetValue;
    }

    public String getGreetText(){
        return "Welcome to "+greetTo;
    }
}
```

```java
//This is view used for displaying UI components
import javax.swing.*;
import java.awt.*;
public class View {
    private JFrame frame;
    private JLabel label;
    private JButton button;
    public View(String text){
        frame = new JFrame("MVC Programming using Swing");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(500,500);
        frame.setVisible(true);
        label = new JLabel(text);
        frame.add(label, BorderLayout.CENTER);
        button = new JButton("Click Me");
        frame.add(button, BorderLayout.SOUTH);
    }
    public JButton getButton(){
        return button;
    }
    public void setText(String text){
        label.setText(text);
    }
}
```

```java
//This is controller which controls program flow
import java.awt.event.*;
public class Controller {
    private Model model;
    private View view;
    private ActionListener actionListener;
    public Controller(Model model, View view){
        this.model = model;
        this.view = view;
    }
    public void contol(){
        actionListener = new ActionListener() {
            public void actionPerformed(ActionEvent actionEvent) {
                showLabel();
            }
        };
        view.getButton().addActionListener(actionListener);
    }
    private void showLabel(){
        view.setText((model.getGreetText()));
    }
}
```

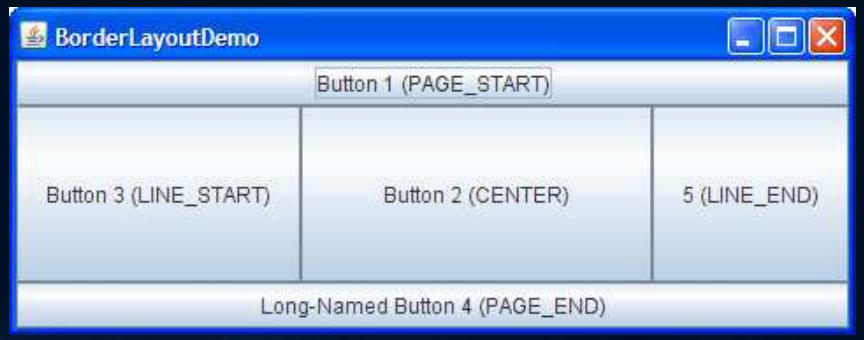
```java
import javax.swing.*;
public class StartProgram
{
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                Model model = new Model("Java Programming");
                View view = new View("This label will change after button click");
                Controller controller = new Controller(model,view);
                controller.contol();
            }
        });
    }
}
```

Layout Management

Several AWT and Swing classes provide layout managers for general use:

1. BorderLayout
2. BoxLayout
3. CardLayout
4. FlowLayout
5. GridBagLayout
6. GridLayout
7. GroupLayout
8. SpringLayout

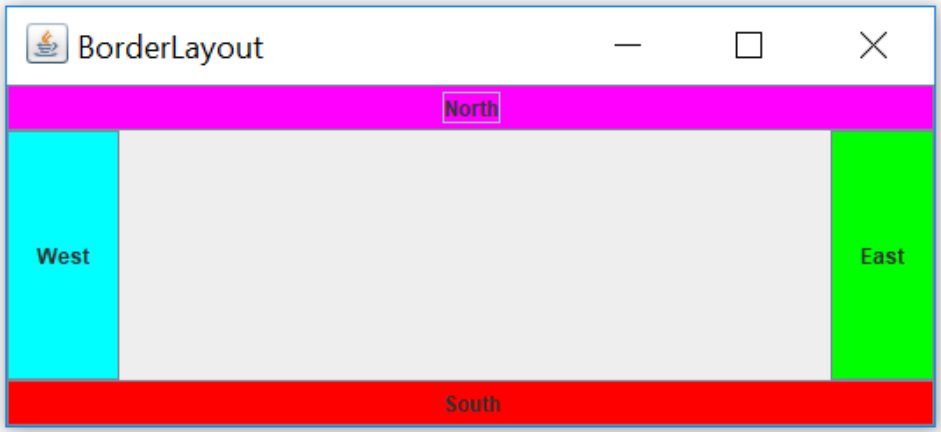
- **BorderLayout:**
  - BorderLayout places components in up to five areas: top, bottom, left, right, and center.
  - All extra space is placed in the center area.
  - Constructor/s:
    - BorderLayout(): creates a border layout but with no gaps between the components.
    - JBorderLayout(int hgap, int vgap): creates a border layout with the given horizontal and vertical gaps between the components.



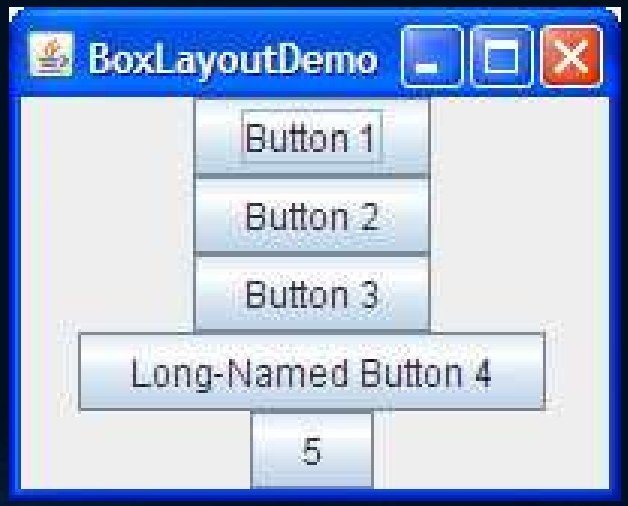
```
1.  import javax.swing.*;
2.  import java.awt.*;
3.  /**
4.   * Created by Tek Raj Chhetri on 7/31/2018.
5.   */
6.  public class FlowLayoutExample {
7.     static JFrame frame;
8.     private static void initGUI(){
9.        frame = new JFrame( "Flow Layout" );
10.       JButton one = new JButton( "1" );
11.       JButton two = new JButton( "2" );
12.       JButton three = new JButton( "3" );
13.       JButton four = new JButton( "4" );
14.       JButton five = new JButton( "5" );
15.       one.setBackground( Color.RED );
16.       one.setForeground( Color.white ); //text color
17.       two.setBackground( Color.magenta );
18.       two.setForeground( Color.white );
19.       three.setBackground( Color.cyan );
20.       three.setForeground( Color.white );
21.       four.setBackground( Color.green );
22.       four.setForeground( Color.white );
```

```
23.         five.setBackground( Color.black );
24.         five.setForeground( Color.white );
25.         //add button to frame
26.         frame.add( one );
27.         frame.add( two );
28.         frame.add( three );
29.         frame.add( four );
30.         frame.add( five);
31.         frame.setVisible( true );
32.         frame.setSize( 500,400 );
33.         frame.setLayout( new FlowLayout( FlowLayout.CENTER ) );
34.         frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
35.       }
36.       public static void main(String[] args) {
37.         SwingUtilities.invokeLater( new Runnable() {
38.           @Override
39.           public void run() {
40.             initGUI();
41.           }
42.         } );
43.       }
44.     }
```

```
C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>java BorderLayoutExample
```



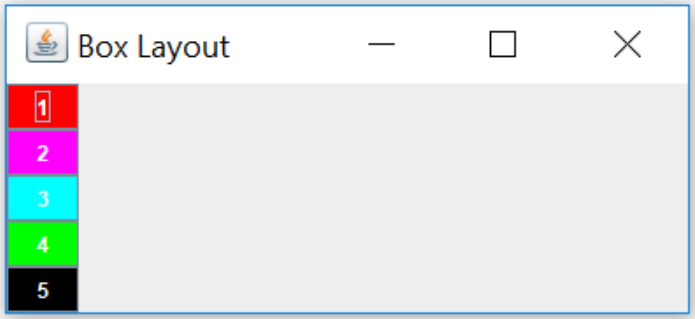
- **BoxLayout:**
  - The Box Layout class puts components in a single row or column.
  - Constructor:
    - BoxLayout(Container c, int axis): creates a box layout that arranges the components with the given axis.



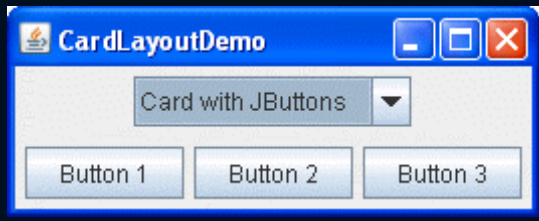
```
1.  import javax.swing.*;
2.  import java.awt.*;
3.  /**
4.   * Created by Tek Raj Chhetri on 7/31/2018.
5.   */
6.  public class BoxLayoutExample {
7.     static JFrame frame;
8.     private static void initGUI(){
9.        frame = new JFrame( "Box Layout" );
10.       JButton one = new JButton( "1" );
11.       JButton two = new JButton( "2" );
12.       JButton three = new JButton( "3" );
13.       JButton four = new JButton( "4" );
14.       JButton five = new JButton( "5" );
15.       one.setBackground( Color.RED );
16.       one.setForeground( Color.white ); //text color
17.       two.setBackground( Color.magenta );
18.       two.setForeground( Color.white );
19.       three.setBackground( Color.cyan );
20.       three.setForeground( Color.white );
21.       four.setBackground( Color.green );
22.       four.setForeground( Color.white );
```

```
23.             five.setBackground( Color.black );
24.             five.setForeground( Color.white );
25.             //add button to frame
26.             frame.add( one );
27.             frame.add( two );
28.             frame.add( three );
29.             frame.add( four );
30.             frame.add( five);
31.             frame.setVisible( true );
32.             frame.setLayout(  new BoxLayout( frame.getContentPane(), BoxLayout.Y_AXIS ));
33.             frame.pack();
34.             frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
35.         }
36.         public static void main(String[] args) {
37.             SwingUtilities.invokeLater( new Runnable() {
38.                 @Override
39.                 public void run() {
40.                   initGUI();
41.                 }
42.             } );
43.         }
44.     }
```

```
C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>java BoxLayoutExample
```

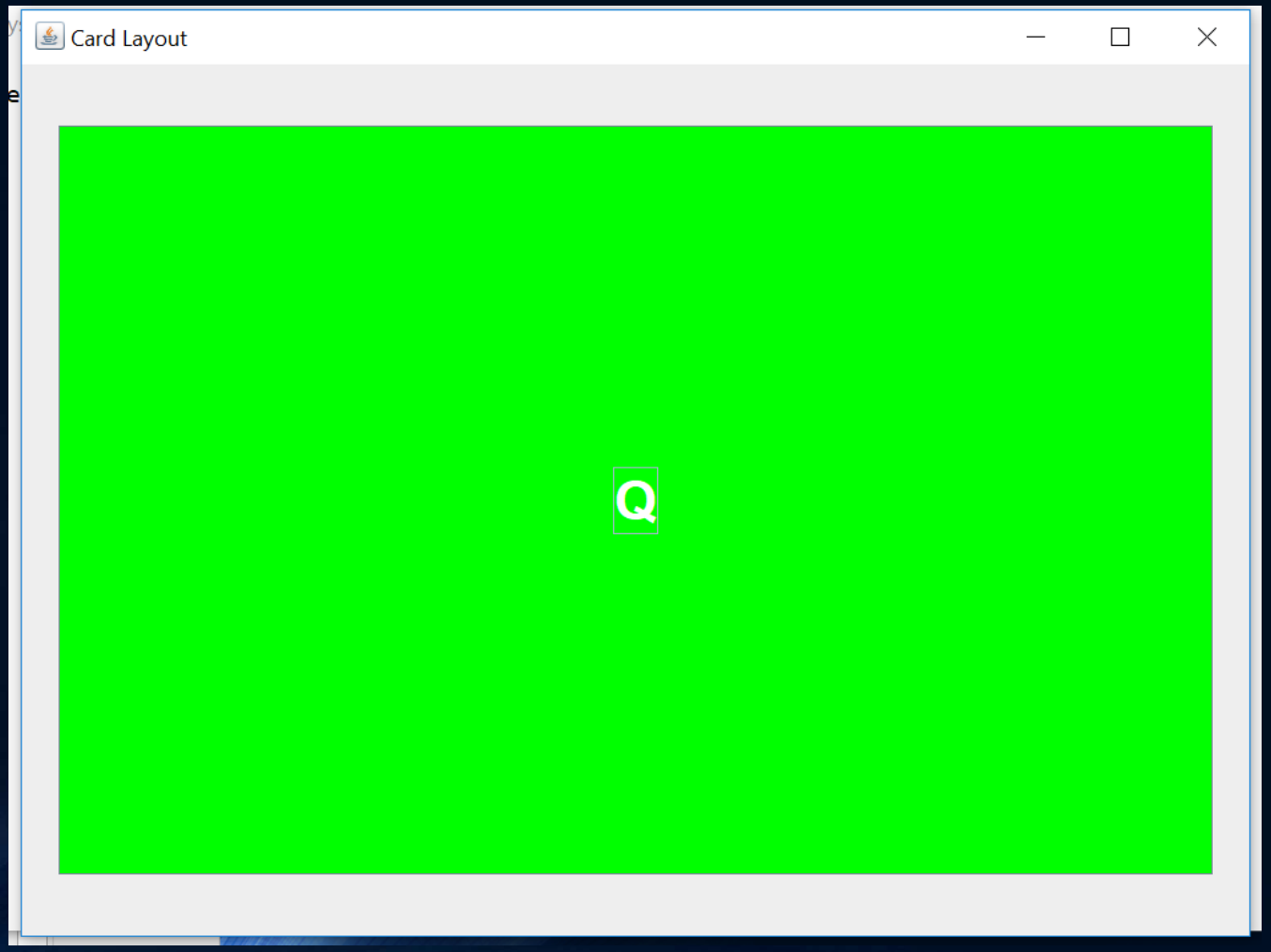


- **CardLayout:**
  - The CardLayout class lets you implement an area that contains different components at different times.
  - A CardLayout is often controlled by a combo box, with the state of the combo box determining which panel (group of components) the CardLayout displays.
  - An alternative to using CardLayout is using a tabbed pane, which provides similar functionality but with a pre-defined GUI
  - Constructors:
    - 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:
    - 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.

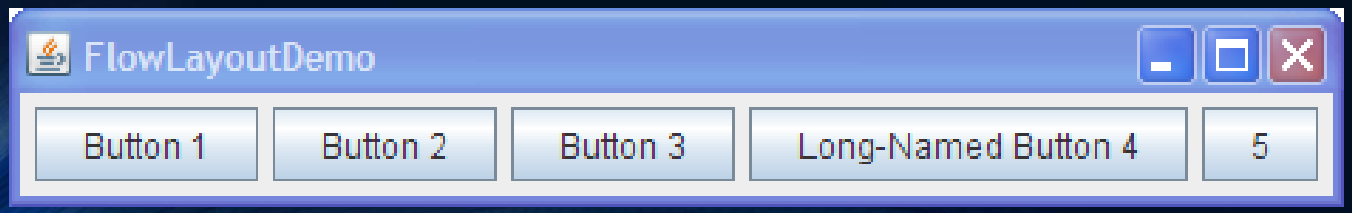


```
1.  import javax.swing.*;
2.  import java.awt.*;

3.  /**
4.   * Created by Tek Raj Chhetri on 7/31/2018.
5.   */

6.  public class CardLayoutExample {

7.      static JFrame frame;
8.      private  static void initGUI(){
9.          frame = new JFrame( "Card Layout" );
10.         JButton queen = new JButton( "Q" );
11.         queen.setFont(new Font( "Arial",Font.BOLD,45 ));
12.         queen.setBackground( Color.green );
13.         queen.setForeground( Color.WHITE );
```

```
14.         //add to frame
15.         frame.add( queen );
16.         frame.setVisible( true );
17.         frame.setSize( 1025,768 );
18.         frame.getContentPane().setLayout( new CardLayout( 30,50 ) );
19.         frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );

20.     }

21.     public static void main(String[] args) {
22.         SwingUtilities.invokeLater(new Runnable(){
23.             public void run(){
24.                 initGUI();
25.             }
26.         });
27.     }
28. }
```


Card Layout
Q

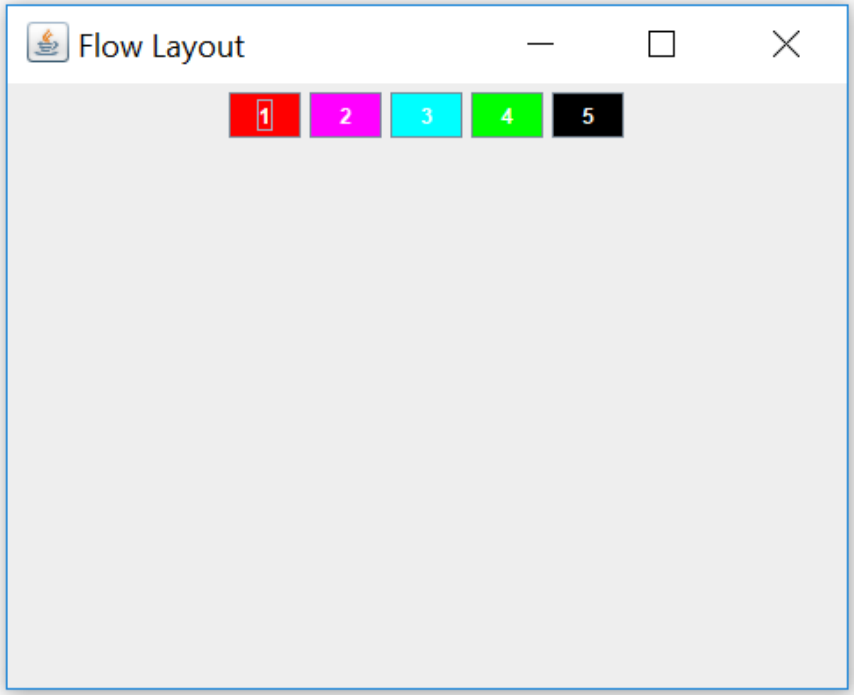
- **FlowLayout:**
  - FlowLayout is the default layout manager for every JPanel.
  - It simply lays out components in a single row, starting a new row if its container is not sufficiently wide.
  - Constructors:
    - FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal 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.
    - Alignment: LEFT, RIGHT, CENTER, LEADING, TRAILING



```
1.  import javax.swing.*;
2.  import java.awt.*;
3.  /**
4.   * Created by Tek Raj Chhetri on 7/31/2018.
5.   */
6.  public class FlowLayoutExample {
7.     static JFrame frame;
8.     private static void initGUI(){
9.        frame = new JFrame( "Flow Layout" );
10.       JButton one = new JButton( "1" );
11.       JButton two = new JButton( "2" );
12.       JButton three = new JButton( "3" );
13.       JButton four = new JButton( "4" );
14.       JButton five = new JButton( "5" );
15.       one.setBackground( Color.RED );
16.       one.setForeground( Color.white ); //text color
17.       two.setBackground( Color.magenta );
18.       two.setForeground( Color.white );
19.       three.setBackground( Color.cyan );
20.       three.setForeground( Color.white );
21.       four.setBackground( Color.green );
22.       four.setForeground( Color.white );
```

```
23.         five.setBackground( Color.black );
24.         five.setForeground( Color.white );
25.         //add button to frame
26.         frame.add( one );
27.         frame.add( two );
28.         frame.add( three );
29.         frame.add( four );
30.         frame.add( five);
31.         frame.setVisible( true );
32.         frame.setSize( 500,400 );
33.         frame.setLayout( new FlowLayout( FlowLayout.CENTER ) );
34.         frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
35.       }
36.       public static void main(String[] args) {
37.         SwingUtilities.invokeLater( new Runnable() {
38.           @Override
39.           public void run() {
40.             initGUI();
41.           }
42.         } );
43.       }
44.     }
```

```
C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>javac FlowLayoutExample.java

C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>java FlowLayoutExample
```



- **GridBagLayout:**
  - GridBagLayout is a sophisticated, flexible layout manager.
  - It aligns components by placing them within a grid of cells, allowing components to span more than one cell.
  - The rows in the grid can have different heights, and grid columns can have different widths.

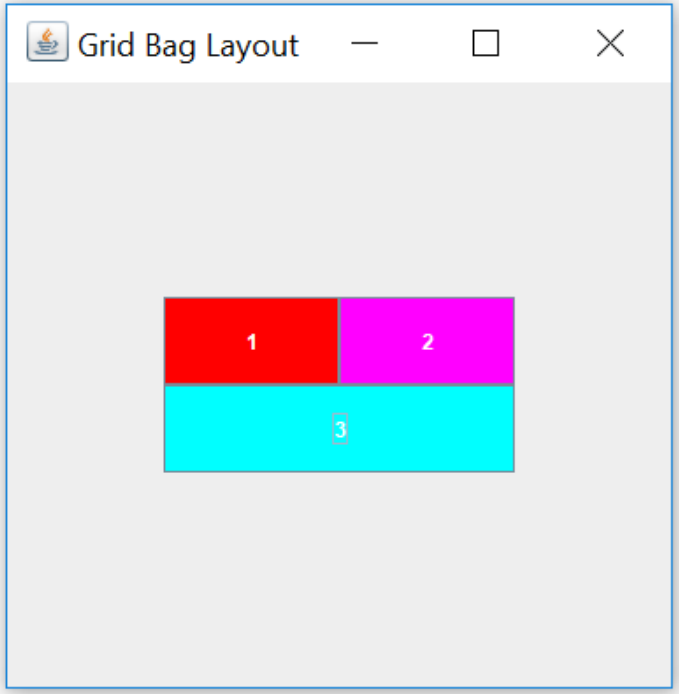
```
1.  import javax.swing.*;
2.  import java.awt.*;
3.  /**
4.   * Created by Tek Raj Chhetri on 7/31/2018.
5.   */
6.  public class GridBagLayoutExample {
7.     static JFrame frame;
8.     private static void initGUI(){
9.        frame = new JFrame( "Grid Bag Layout" );
10.       JButton one = new JButton( "1" );
11.       JButton two = new JButton( "2" );
12.       JButton three = new JButton( "3" );
13.       one.setSize( 100,50 );
```

```
14.        one.setPreferredSize( one.getSize(  ) );
15.        two.setSize( 100,50 );
16.        two.setPreferredSize( one.getSize(  ) );
17.        three.setSize( 200,50 );
18.       three.setPreferredSize( three.getSize()  );
19.        one.setBackground( Color.RED );
20.        one.setForeground( Color.white ); //text color
21.        two.setBackground( Color.magenta );
22.        two.setForeground( Color.white );
23.        three.setBackground( Color.cyan );
24.        three.setForeground( Color.white );
25.        frame.setVisible( true );
26.        frame.setLayout( new GridBagLayout() );
```

```
27.     frame.setSize( 400,400 );
28.     frame.setPreferredSize( frame.getSize() );
29.     frame.pack();
30.     frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
31.     GridBagConstraints gridBagConstraints = new GridBagConstraints(  );
32.     gridBagConstraints.fill = GridBagConstraints.HORIZONTAL;
33.     gridBagConstraints.gridwidth = 2;
34.     gridBagConstraints.gridx = 0;
35.     gridBagConstraints.gridy = 1;
36.     frame.add( three,gridBagConstraints );
37.     gridBagConstraints.fill = GridBagConstraints.HORIZONTAL;
38.     gridBagConstraints.gridwidth = 1;
39.     gridBagConstraints.gridx = 0;
40.     gridBagConstraints.gridy = 0;
```

```
41.         frame.add( one,gridBagConstraints );
42.         gridBagConstraints.gridx = 1;
43.         gridBagConstraints.gridy = 0;
44.         frame.add( two,gridBagConstraints );
45.     }
46.     public static void main(String[] args) {
47.         SwingUtilities.invokeLater( new Runnable() {
48.             @Override
49.             public void run() {
50.                 initGUI();
51.             }
52.         } );
53.     }
54. }
```

```
C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>javac GridBagLayoutExample.java

C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>java GridBagLayoutExample
```



- **GridLayout:**
  - GridLayout simply makes a bunch of components equal in size and displays them in the requested number of rows and columns.
  - Constructors:
    - 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.

```
1.  import javax.swing.*;
2.  import java.awt.*;
3.  import java.awt.event.ActionEvent;
4.  import java.awt.event.ActionListener;

5.  /**
6.   * Created by Tek Raj Chhetri on 7/31/2018.
7.   * Program to demonstrate Grid Layout
8.   */
9.  public class GridLayoutExample {
10.     static JFrame frame;
11.    private static void initGUI(){
12.       frame = new JFrame( "Grid Layout" );

13.       JButton one = new JButton( "1" );
14.       JButton two = new JButton( "2" );
```

```
15.     JButton three = new JButton( "3" );
16.     JButton four = new JButton( "4" );
17.     JButton five = new JButton( "5" );
18.     JButton six = new JButton( "6" );
19.     JButton seven = new JButton( "7" );
20.     JButton eight = new JButton( "8" );
21.     JButton nine = new JButton( "9" );
22.     //set color to button

23.     one.setBackground( Color.RED );
24.     one.setForeground( Color.white ); //text color
25.     two.setBackground( Color.magenta );
26.     two.setForeground( Color.white );
27.     three.setBackground( Color.cyan );
28.     three.setForeground( Color.white );
29.     four.setBackground( Color.green );
```

```
30.     four.setForeground( Color.white );
31.     five.setBackground( Color.black );
32.     five.setForeground( Color.white );
33.     six.setBackground( Color.blue );
34.     six.setForeground( Color.white );
35.     seven.setBackground( Color.magenta );
36.     seven.setForeground( Color.white );
37.     eight.setBackground( Color.orange );
38.     eight.setForeground( Color.white );
39.     nine.setBackground( Color.MAGENTA );
40.     nine.setForeground( Color.white );
41.     //add buttons to frame
42.     frame.add( one );
43.     frame.add( two );
44.     frame.add( three );
45.     frame.add( four );
```

```
46.         frame.add( five);
47.         frame.add( six );
48.         frame.add( seven );
49.         frame.add( eight );
50.         frame.add( nine );

51.         frame.setVisible( true );
52.         frame.setSize( 500,400 );
53.         //set gridlayout
54.         frame.setLayout( new GridLayout(3,3) );
55.         frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
56.         //add listener
57.         one.addActionListener( new ActionListener() {
58.             @Override
59.             public void actionPerformed(ActionEvent e) {
60.                 showDialog( 1 );
```

```
61.         }
62.       } );
63.       two.addActionListener( new ActionListener() {
64.         @Override
65.         public void actionPerformed(ActionEvent e) {
66.           showDialog( 2 );
67.         }
68.       } );
69.       three.addActionListener( new ActionListener() {
70.         @Override
71.         public void actionPerformed(ActionEvent e) {
72.           showDialog( 3 );
73.         }
74.       } );
75.       four.addActionListener( new ActionListener() {
76.         @Override
```

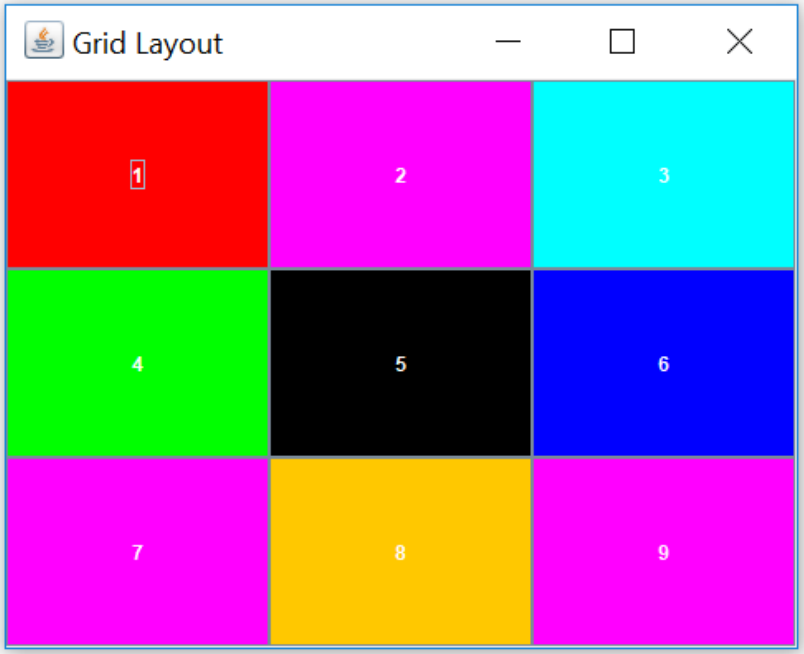
```
77.         public void actionPerformed(ActionEvent e) {
78.            showDialog( 4 );
79.         }
80.      } );
81.      five.addActionListener( new ActionListener() {
82.         @Override
83.         public void actionPerformed(ActionEvent e) {
84.            showDialog( 5 );
85.         }
86.      } );
87.      six.addActionListener( new ActionListener() {
88.         @Override
89.         public void actionPerformed(ActionEvent e) {
90.            showDialog( 6 );
91.         }
92.      } );
```

```
93.     seven.addActionListener( new ActionListener() {
94.       @Override
95.       public void actionPerformed(ActionEvent e) {
96.         showDialog( 7 );
97.       }
98.     } );
99.     eight.addActionListener( new ActionListener() {
100.      @Override
101.      public void actionPerformed(ActionEvent e) {
102.        showDialog( 8 );
103.      }
104.    } );
105.    nine.addActionListener( new ActionListener() {
106.      @Override
107.      public void actionPerformed(ActionEvent e) {
108.        showDialog( 9 );
```

```
109.         }
110.       } );
111.    }

112.    private static void showDialog(int number){
113.       JOptionPane.showMessageDialog( frame,number+" is clicked");
114.    }

115.    public static void main(String[] args) {
116.       SwingUtilities.invokeLater( new Runnable() {
117.          @Override
118.          public void run() {
119.             initGUI();
120.          }
121.       } );
122.    }
123. }
```

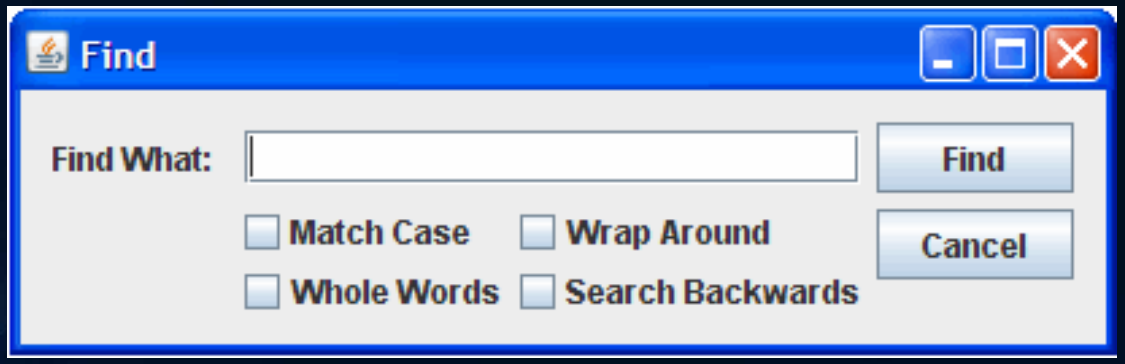
C:\WINDOWS\system32\cmd.exe - java GridLayoutExample

```
C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>javac GridLayoutExample.java

C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>java GridLayoutExample
```



```
C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>javac GridLayoutExample.java

C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>java GridLayoutExample
```

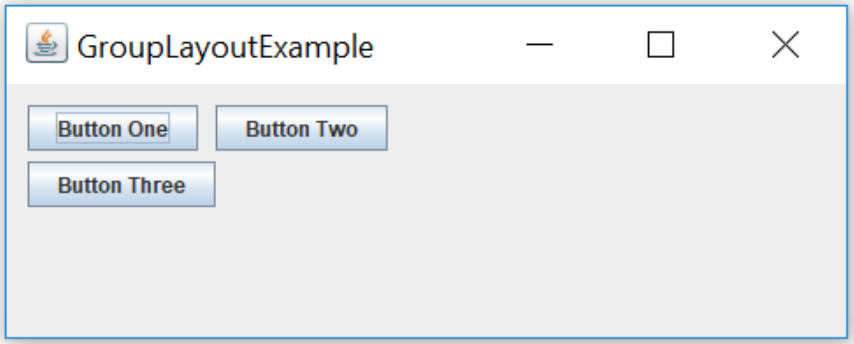
- **GroupLayout:**
  - GroupLayout is a layout manager that was developed for use by GUI builder tools, but it can also be used manually.
  - GroupLayout works with the horizontal and vertical layouts separately.
  - The layout is defined for each dimension independently. Consequently, however, each component needs to be defined twice in the layout.



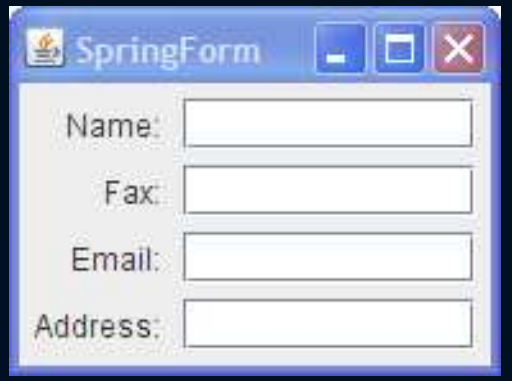
```
1.  //@source Javatpoint.com
2.  import javax.swing.*;
3.  import java.awt.*;
4.  import static javax.swing.GroupLayout.Alignment.*;
5.  public class GroupLayoutExample {
6.     public static void main(String[] args) {
7.        JFrame frame = new JFrame("GroupLayoutExample");
8.        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9.        Container myPanel = frame.getContentPane();
10.       frame.setSize( 500,200 );
11.       frame.setPreferredSize( frame.getSize() );
12.       GroupLayout groupLayout = new GroupLayout(myPanel);
13.       groupLayout.setAutoCreateGaps(true);
14.       groupLayout.setAutoCreateContainerGaps(true);
```

```
15.         myPanel.setLayout(groupLayout);
16.         JButton b1 = new JButton("Button One");
17.         JButton b2 = new JButton("Button Two");
18.         JButton b3 = new JButton("Button Three");
19.         groupLayout.setHorizontalGroup(groupLayout.createSequentialGroup()
20.
    .addGroup(groupLayout.createParallelGroup(LEADING).addComponent(b1).addComponent(
    b3))
21.                 .addGroup(groupLayout.createParallelGroup(TRAILING).addComponent(b2)));
22.         groupLayout.setVerticalGroup(groupLayout.createSequentialGroup()
23.
    .addGroup(groupLayout.createParallelGroup(BASELINE).addComponent(b1).addComponent
    (b2))
24.                 .addGroup(groupLayout.createParallelGroup(BASELINE).addComponent(b3)));
25.         frame.pack();
26.         frame.setVisible(true);
27.     }
28. }
```

```
C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>java GroupLayoutExample
```



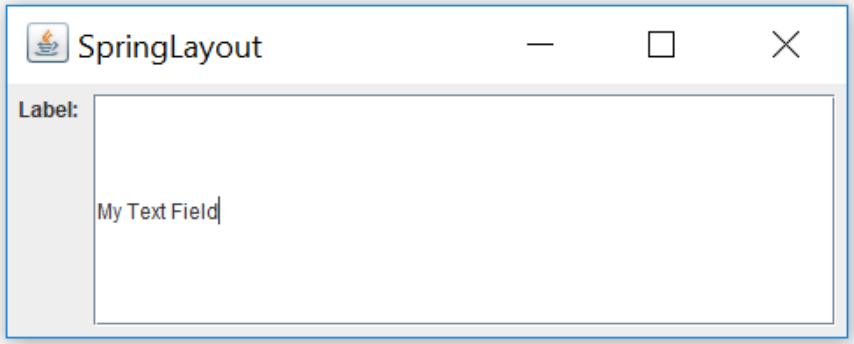
- **SpringLayout:**
  - SpringLayout is a flexible layout manager designed for use by GUI builders.
  - It lets you specify precise relationships between the edges of components under its control.
  - Eg:
    - Placing left edge of one component is a certain distance (which can be dynamically calculated) from the right edge of a second component.



```
1.  //@source javatpoint.com
2.  import javax.swing.*;
3.  import java.awt.*;

4.  public class SpringLayoutExample {
5.     private static void createAndShowGUI() {
6.        JFrame frame = new JFrame("SpringLayout");
7.        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8.        frame.setSize( 500,200 );
9.        frame.setPreferredSize( frame.getSize() );

10.       Container contentPane = frame.getContentPane();
11.       SpringLayout layout = new SpringLayout();
12.       contentPane.setLayout(layout);

13.       JLabel label = new JLabel("Label: ");
14.       JTextField textField = new JTextField("My Text Field", 15);
15.       contentPane.add(label);
16.       contentPane.add(textField);
```

```
17.         layout.putConstraint(SpringLayout.WEST, label,6,SpringLayout.WEST, contentPane);
18.         layout.putConstraint(SpringLayout.NORTH, label,6,SpringLayout.NORTH, contentPane);
19.         layout.putConstraint(SpringLayout.WEST, textField,6,SpringLayout.EAST, label);
20.         layout.putConstraint(SpringLayout.NORTH, textField,6,SpringLayout.NORTH, contentPane);
21.         layout.putConstraint(SpringLayout.EAST, contentPane,6,SpringLayout.EAST, textField);
22.         layout.putConstraint(SpringLayout.SOUTH, contentPane,6,SpringLayout.SOUTH, textField);

23.         frame.pack();
24.         frame.setVisible(true);
25.       }
26.       public static void main(String[] args) {
27.         javax.swing.SwingUtilities.invokeLater(new Runnable() {
28.           public void run() {
29.             createAndShowGUI();
30.           }
31.         });
32.       }
33.     }
```

```
C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>javac SpringLayoutExample.java

C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>java SpringLayoutExample
```



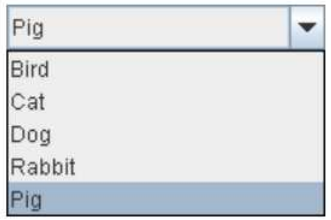
Basic Swing Components

Basic Controls

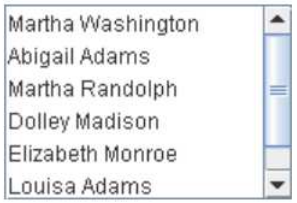
Simple components that are used primarily to get input from the user; they may also show simple state.

JButton

JCheckBox



JComboBox



JList

JMenu

JRadioButton

JSlider

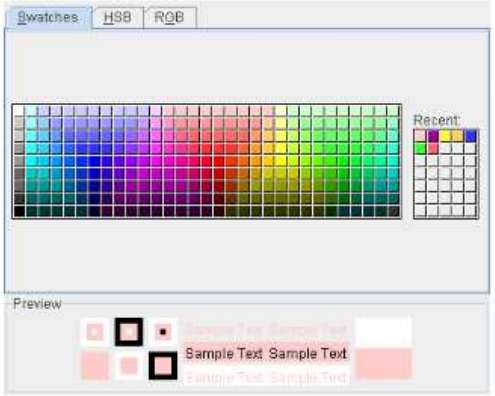
JSpinner

JTextField

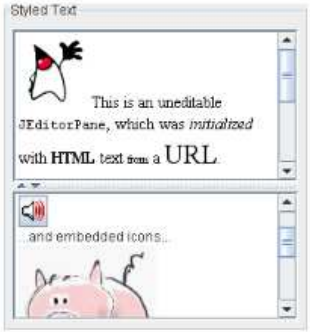
JPasswordField

Interactive Displays of Highly Formatted Information

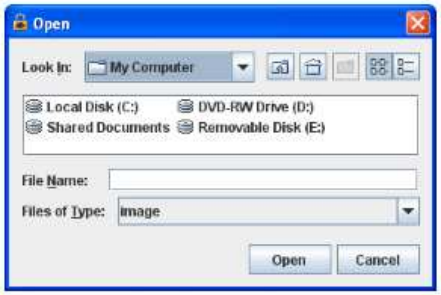
These components display highly formatted information that (if you choose) can be modified by the user.



JColorChooser



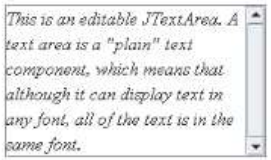
JEditorPane and JTextPane



JFileChooser

Host	User	Password	Last Modified
Blocca Games	Freddy	lfas!6Awwzb	Mar 16, 2006
zabble	ichabod	Tazbl34$IZ	Mar 6, 2006
Sun Developer	fraz@hotmail.co...	AasW541lfbZ	Feb 22, 2006
Heirloom Seeds	shams@gmail....	bkzlADF78l	Jul 29, 2005
Pacific Zoo Shop	seal@hotmail.c...	YbAf124%z	Feb 22, 2006

JTable



JTextArea

JTree

Uneditable Information Displays

These components exist solely to give the user information.

[JLabel]

[JProgressBar]

[JSeparator]

[JToolTip]

Top-Level Containers

At least one of these components must be present in any Swing application.

JApplet

JDialog

JFrame

General-Purpose Containers

These general-purpose containers are used in most Swing applications.

JPanel

JScrollPane

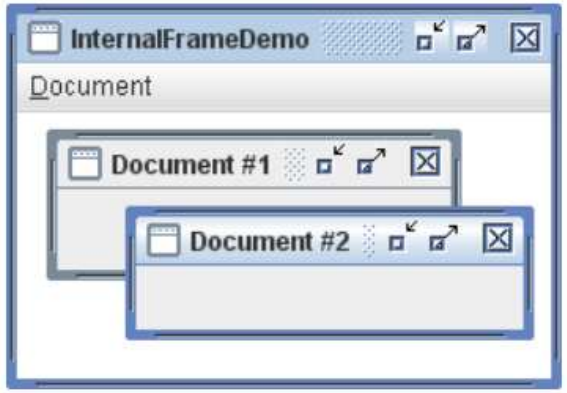
JSplitPane

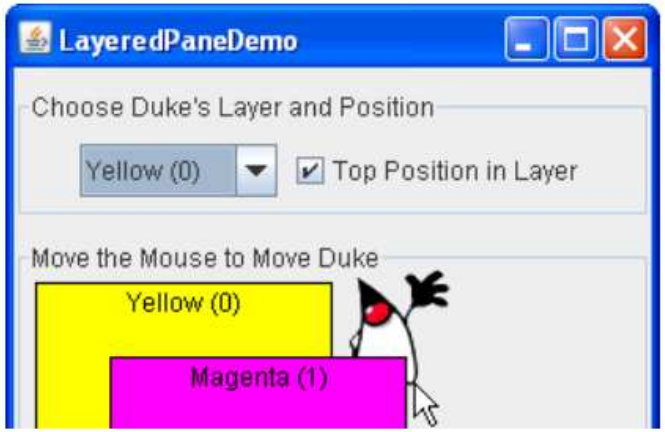
JTabbedPane

JToolBar

Special-Purpose Containers

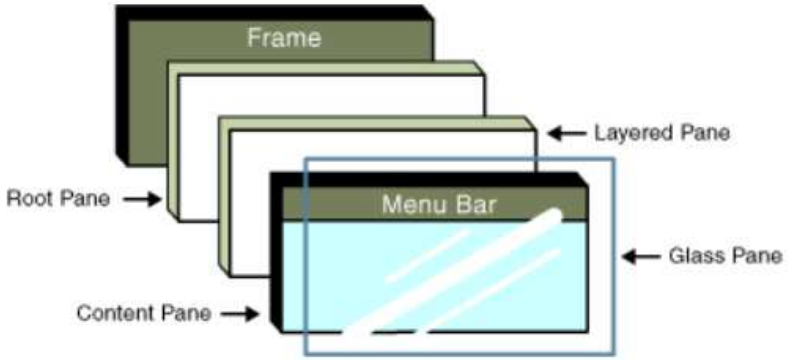
These special-purpose containers play specific roles in the UI.





JInternalFrame

JLayeredPane



Root pane

- Example Program Demonstrating different Swing Components

```
1.  import javax.swing.*;
2.  /**
3.   * Created by Tek Raj Chhetri on 7/31/2018.
4.   */
5.  public class SwingComponents {
6.     static JFrame frame;
7.     private static void initGUI(){
8.        frame = new JFrame( "SWING Components" );
9.        JLabel label = new JLabel( "Name" );
10.       JTextField textField = new JTextField(  );
11.       JOptionPane optionPane = new JOptionPane( "Select" );
```

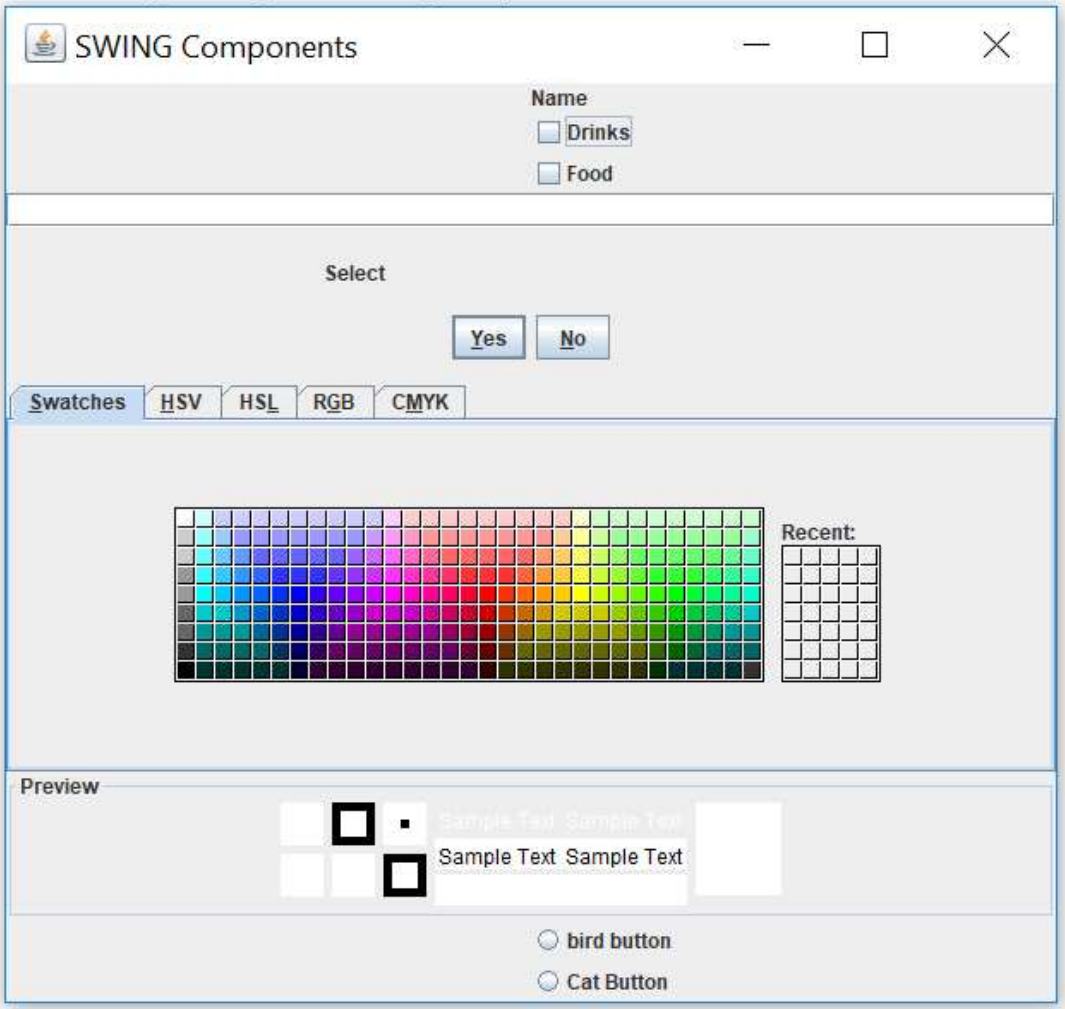
```
12.     optionPane.setOptionType( o );
13.     JColorChooser colorChooser = new JColorChooser(  );
14.     JRadioButton birdButton = new JRadioButton(  "bird button");
15.     JRadioButton catButton = new JRadioButton( "Cat Button" );
16.     //grouping radio button
17.     ButtonGroup group = new ButtonGroup();
18.     group.add(birdButton);
19.     group.add(catButton);

20.     JCheckBox checkBoxFood = new JCheckBox( "Food" );
21.     JCheckBox checkBoxDrinks = new JCheckBox( "Drinks" );
```

```
22.     frame.add( label );
23.     frame.add( checkBoxDrinks );
24.     frame.add( checkBoxFood );
25.     frame.add( textField );
26.     frame.add( optionPane );
27.     frame.add( colorChooser );
28.     frame.add( birdButton );
29.     frame.add( catButton );
30.     frame.setLayout(  new BoxLayout( frame.getContentPane(), BoxLayout.Y_AXIS ));

31.     frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
32.     frame.pack();
```

```
33.         frame.setVisible( true );
34.     }

35.     public static void main(String[] args) {
36.         SwingUtilities.invokeLater( new Runnable() {
37.             @Override
38.             public void run() {
39.                 initGUI();
40.             }
41.         } );
42.     }
43. }
```

C:\WINDOWS\system32\cmd.exe - java SwingComponents

```
C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>javac SwingComponents.java

C:\Users\USER\Desktop\lecture\java\Unit -VII\Programs>java SwingComponents
```



Suggested Readings

- The respective topics in The complete Reference Java 7 (or any higher edition) by Hebert Schildt
- https://docs.oracle.com/javase/tutorial/uiswing/events/index.html
- https://docs.oracle.com/javase/7/docs/api/java/awt/event/package-summary.html
- https://docs.oracle.com/javase/tutorial/uiswing/components/index.html

References

- The complete Reference Java 7 by Hebert Schildt
- Java 8 in Action by Dreamtech press.
- Mit Opencourseware
- http://ee402.eeng.dcu.ie/
- Deitel, H.M., Dietel, P.J. Java How To Program 7th Ed. Upper Saddle River: Pearson, 2007.
- https://www.javatpoint.com/
- http://web.mit.edu/6.005/www/sp14/psets/ps4/java-6-tutorial/components.html
- https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html
- https://docs.oracle.com/javase/tutorial/uiswing/events/index.html
- https://images.google.com for Images