PROGRAMMING

Lecture 19

Sushil Paudel

PREVIOUS TOPIC

- Exception Handling
- Try Catch

TODAY'S TOPIC

- Graphical User Interface (GUI)
- Border Layout
- Flow Layout

GRAPHICAL USER INTERFACE

- GUI is an interface that uses icons or other visual indicators to interact with electronic devices, rather than only text via a command line.
- For example, all versions of Microsoft Windows are a GUI, whereas MS-DOS is a command line.
- A GUI uses windows, icons, and menus to carry out commands, such as opening, deleting, and moving files.
- Although a GUI operating system is primarily navigated using a mouse, the keyboard can also be used to navigate using keyboard shortcuts or the arrow keys.

GRAPHICAL USER INTERFACE

- As an example, if you wanted to open a software program on a GUI operating system, you would move the mouse pointer to the program's icon and double-click the icon.
- GUI programming involves the use of a number of predefined components such as buttons, checkboxes, text fields, windows, menus, etc. that are part of the class hierarchy.

AWT and Swing

- There are two sets of Java APIs for graphics programming: AWT (Abstract Windowing Toolkit) and Swing.
- AWT API was introduced in JDK 1.0. Most of the AWT components have become out of date and should be replaced by newer Swing components.
- Swing is the latest GUI toolkit, and provides a richer set of interface components than the AWT

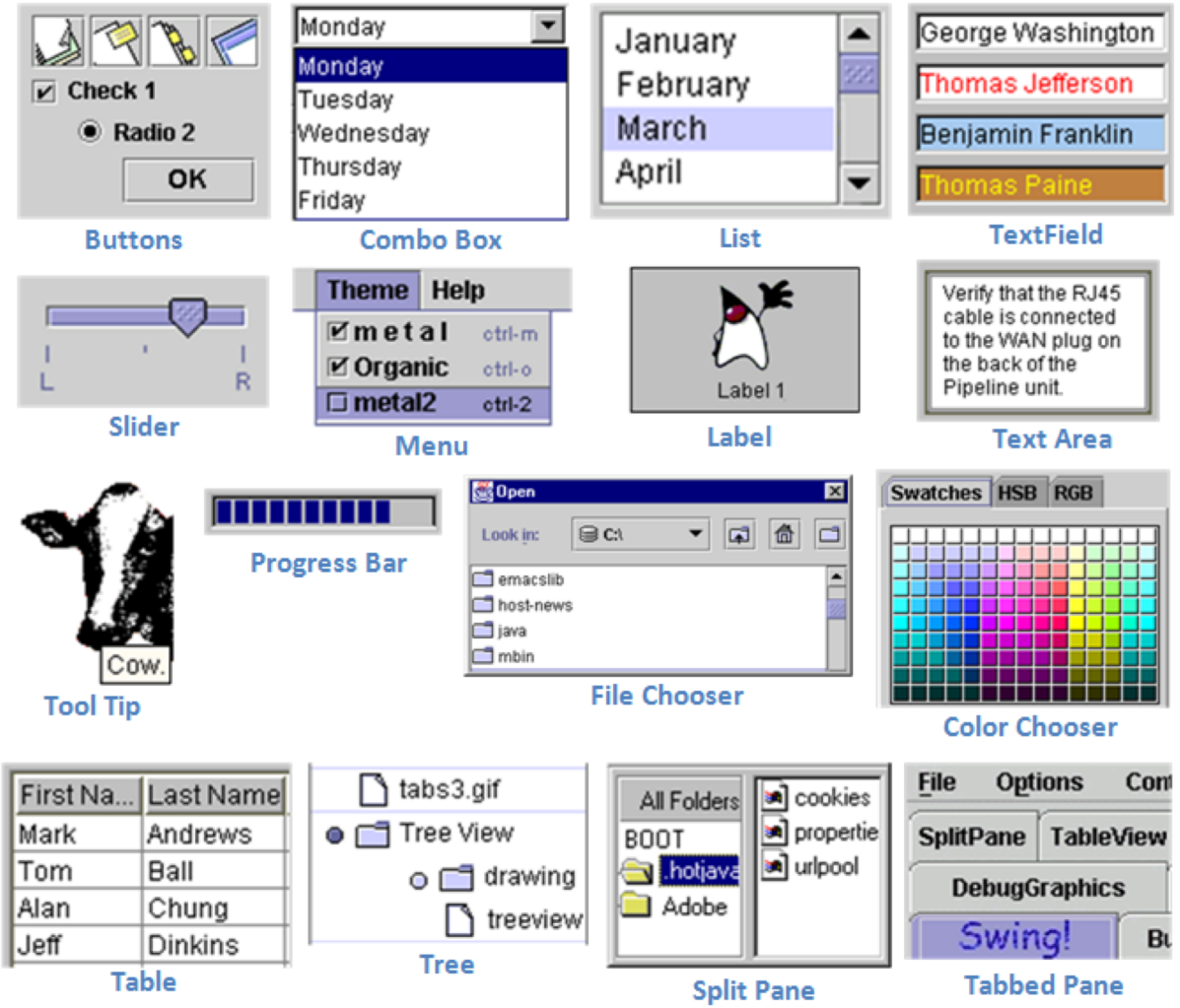
GUI ELEMENTS

There are two types of GUI elements:

- **Component**: Components are elementary GUI entities, such as Button, Label, TextField, etc.

- **Container**: Containers, such as Frame and Panel, are used to hold components in a specific layout (such as FlowLayout or GridLayout). A container can also hold sub-containers.

COMPONENTS


Check 1
Radio 2
OK
Buttons
Monday
Monday
Tuesday
Wednesday
Thursday
Friday
Combo Box
January
February
March
April
List
George Washington
Thomas Jefferson
Benjamin Franklin
Thomas Paine
TextField
L
R
Slider
Theme
Help
metal ctrl-m
Organic ctrl-o
metal2 ctrl-2
Menu
Label 1
Label
Verify that the RJ45 cable is connected to the WAN plug on the back of the Pipeline unit.
Text Area
Cow.
Tool Tip
Progress Bar
Open
Look in: C:\
emacslib
host-news
java
mbin
File Chooser
Swatches
HSB
RGB
Color Chooser
First Na...
Last Name
Mark
Andrews
Tom
Ball
Alan
Chung
Jeff
Dinkins
Table
tabs3.gif
Tree View
drawing
treeview
Tree
All Folders
BOOT
.hotjava
Adobe
cookies
propertie
urlpool
Split Pane
File
Options
SplitPane
TableView
DebugGraphics
Swing!
Tabbed Pane

LAYOUT MANAGER

- The Layout Managers are used to arrange components in a particular manner.
- In Java swing, Layout manager is used to position all its components, with setting properties, such as the size, the shape and the arrangement.
- Different layout managers could have varied different settings on its components.
- The layout manager automatically positions all the components within the container. Even if you do not use the layout manager, the components are still positioned by the default layout manager

LAYOUT MANAGERS

We will learn the following layout managers

- BorderLayout
- FlowLayout
- GridBagLayout
- GridLayout

JFRAME

- The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class.
- JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI.

JFRAME

HOW TO CREATE JFRAME

Generally, we use two ways to create frame.

- **By creating the object of Frame class (association)**
- **By extending Frame class (inheritance)**

JFRAME FROM OBJECT

```
import javax.swing.JFrame;

public class JFrameExample {

    public static void main(String[] args){
        JFrame frame= new JFrame();
        frame.setSize(600, 400);
        frame.setTitle("Welecome to Java Swing");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
        frame.setVisible(true);

    }
}
```

OUTPUT

JFRAME FROM INHERITANCE

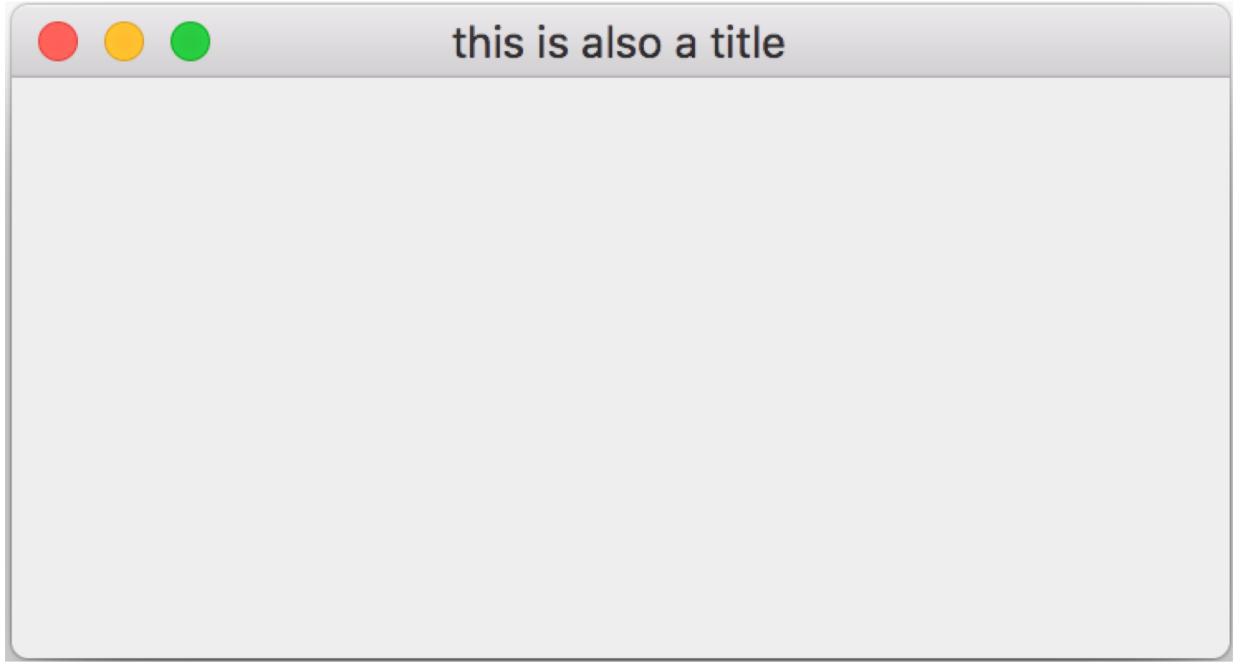
```
import javax.swing.*;

public class Test extends Jframe {

    public void createFrame() {
        setTitle("this is also a title");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400, 500);
        setVisible(true);
    }

    public static void main(String[] args) {
        Test test = new Test();
        test.createFrame();
    }
}
```

OUTPUT



JPANEL

- JPanel, a part of Java Swing package, is a container that can store a group of components.
- The main task of JPanel is to organize components, various layouts can be set in JPanel which provide better organization of components, however it does not have a title bar.

```
// create a panel
JPanel p = new JPanel();
```

JLABEL

- The object of JLabel class is a component for placing text in a container.
- It is used to display a single line of read only text.
- The text can be changed by an application but a user cannot edit it directly.

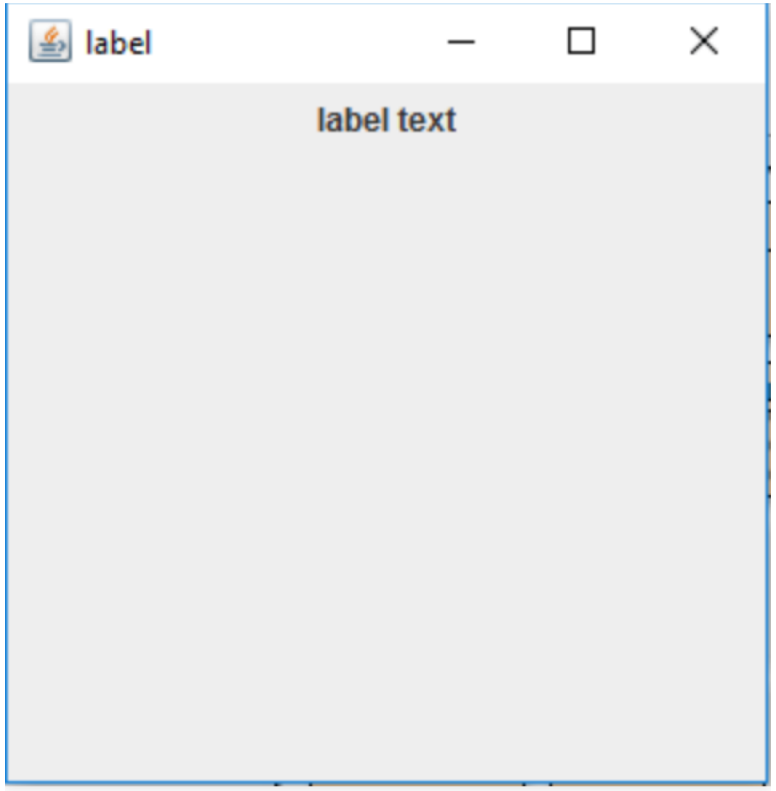
JLABEL

```
public class JButtonInAction {
    public static void main(String[] args) {
        // create a new frame to store text field and button
       JFrame f = new JFrame("label");
        // create a label to display text
        JLabel l = new JLabel();
        // add text to label
        l.setText("label text");

        // create a panel
        JPanel p = new JPanel();
        // add label to panel
        p.add(l);
        // add panel to frame
        f.add(p);

        // set the size of frame
        f.setSize(300, 300);
        f.setVisible(true);
    }
}
```

OUTPUT

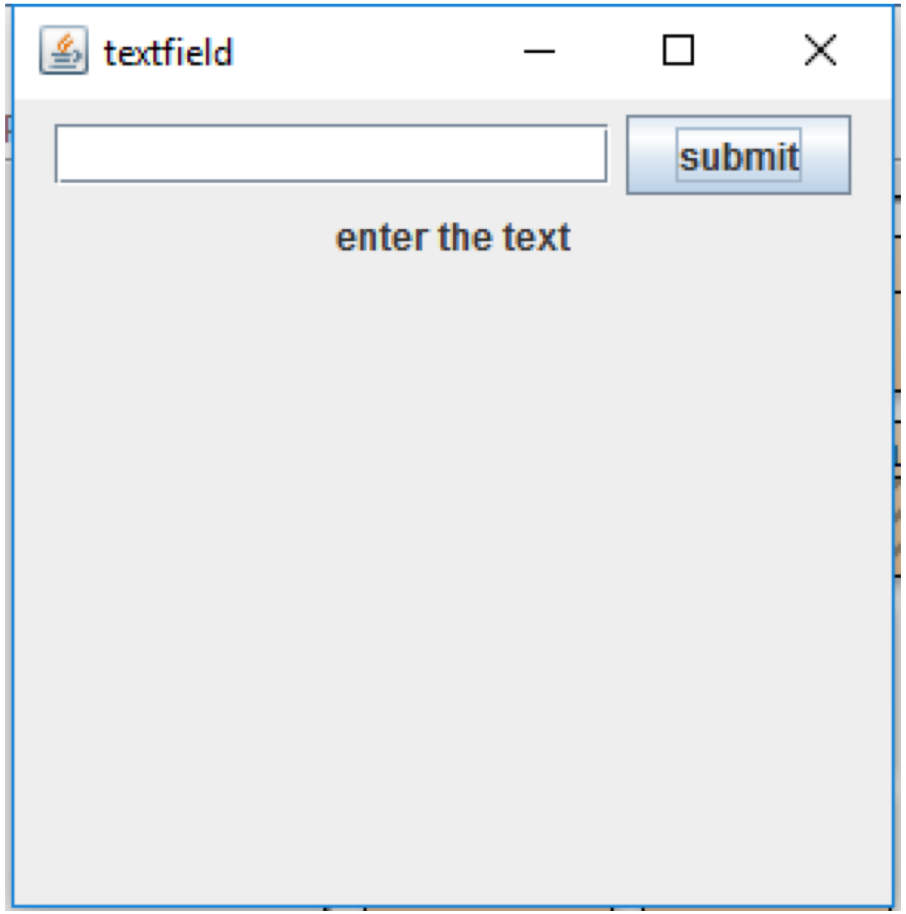


JTEXTFIELD

- The object of a JTextField class is a text component that allows the editing of a single line text.

```
// create a object of JTextField with 16 columns
JTextField tf = new JTextField(16);
```

JTEXTFIELD

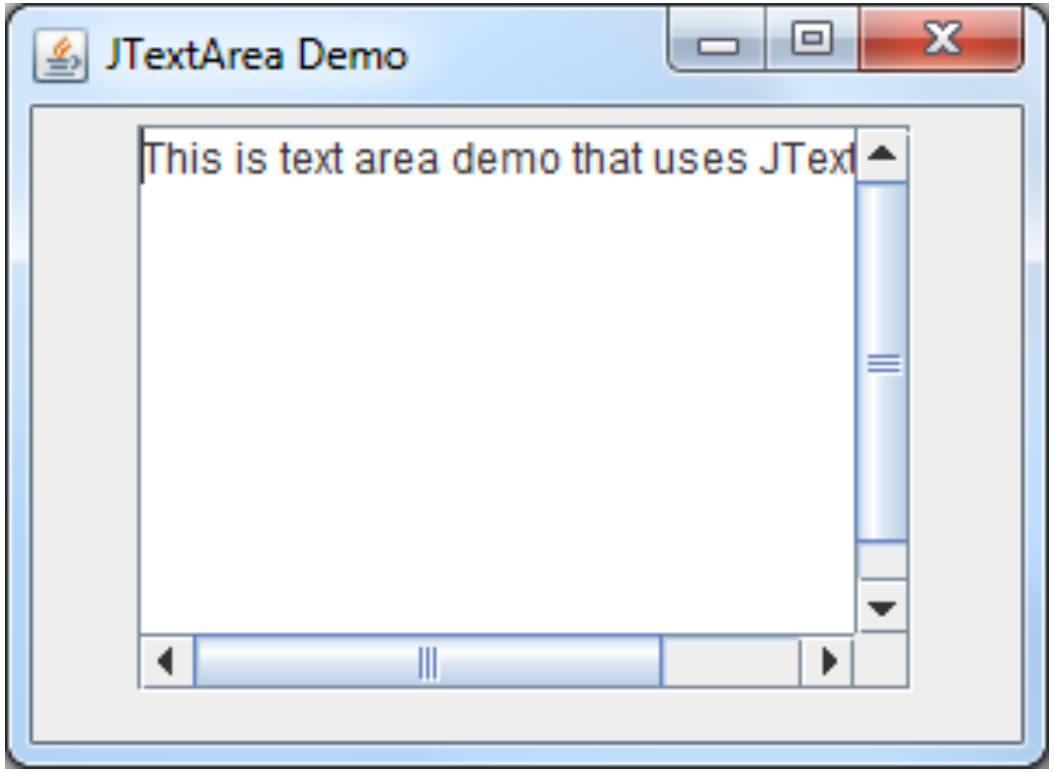


JTEXTAREA

The object of a JTextArea class is a multi-line region that displays text. It allows the editing of multiple line text.

```
// create a text area, specifying the rows and columns
JTextArea jt = new JTextArea(10, 10);
```

JTEXTAREA



JBUTTON

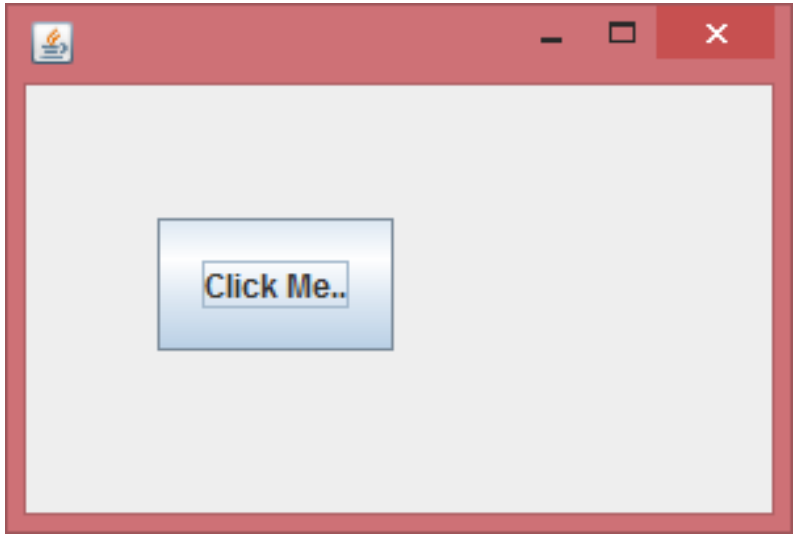
- JButton class is used to create a push button control, which can generate an ActionEvent when it is clicked.
- In order to handle a button click event, ActionListener interface should be implemented.
- JButton is a component which extends JComponent class and it can be added to the container.

JBUTTON

```
import javax.swing.JButton;
import javax.swing.JFrame;
public class JButtonExample {
    JButtonExample(){
          JFrame frame=new JFrame();
        JButton b=new JButton("Click Me..");
        /* This method specifies the location and size of button. In method setBounds(x, y,
width, height)
         * x,y are cordinates from the top left * */
        b.setBounds(50,50,90, 50);
        //Adding button onto the frame
        frame.add(b);
        frame.setSize(300,200);
        frame.setLayout(null);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

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

OUTPUT



BUTTON ACTION

Two ways to implement click action

- Implementing Interface ActionListener
- Implementing anonymously

ACTION LISTENER

```
public interface ActionListener extends EventListener {

    /**
     * Invoked when an action occurs.
     */
    public void actionPerformed(ActionEvent e);

}
```

IMPLEMENTING ACTIONLISTENER

```
public class JButtonInAction extends JFrame implements ActionListener {
        public JButtonInAction() {

                JButton rb = new JButton("Red");
                rb.addActionListener(this);

                add(rb);
                setTitle("Buttons In Action");
                setSize(300, 350);
                setVisible(true);
        }


        public void actionPerformed(ActionEvent e) {
                System.out.println("You clicked me!");
        }

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

IMPLEMENTING ANONYMOUSLY

```
public class JButtonInAction {
    public static void main(String[] args) {

        JFrame f = new JFrame("Button Example");
        JTextField tf = new JTextField();
        tf.setBounds(50, 50, 150, 20);

        JButton b = new JButton("Click Here");
        b.setBounds(50, 100, 95, 30);

        // Adding Action Listener
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                tf.setText("Welcome to Javatpoint.");
            }
        });

        f.add(b);
        f.add(tf);
        f.setSize(400, 400);
        f.setVisible(true);
    }
}
```

WHAT IS ANONYMOUS?

```
interface Manageable {
   void manage();
}
```

IMPLEMENTING ANONYMOUSLY

```
public class AnonymousInterfaceDemo {

    public static void main(String[] args) {
        Manageable m = new Manageable() {
            public void manage(){
                System.out.println("It is manageable");
            }
        }; // anonymous interface implementer closes here
        //m contains an object of anonymous interface implementer of     Manageable.
        m.manage();
    }
}
```

OUTPUT

It is manageable

PROVIDING IMPLEMENTATION

```
public class ImplemtationDemo implements Manageable {

    @Override
    public void manage() {
        System.out.println("It is manageable");
    }

    public static void main(String[] args) {
        ImplemtationDemo id = new ImplemtationDemo();
        id.manage();
    }
}
```

OUTPUT

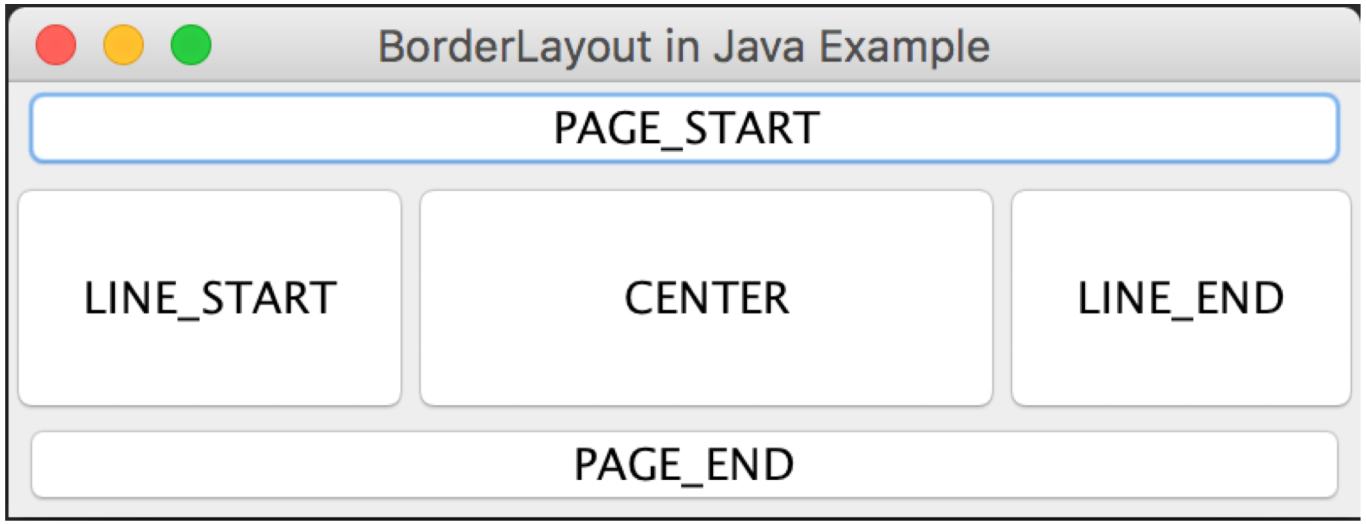
It is manageable

BORDERLAYOUT

The class **BorderLayout** arranges the components to fit in the five regions:

- PAGE_START
- PAGE_END
- LINE_START
- LINE_END
- CENTER

BORDERLAYOUT



BORDERLAYOUT EXAMPLE

```
public class Test extends JFrame {
    public Test() {
        JPanel p = new JPanel();
        BorderLayout bl = new BorderLayout();
        p.setLayout(bl);
        JButton startButton = new JButton(“Button 1”);
        p.add(startButton, BorderLayout.PAGE_START);
        p.add(new JButton(”Button 2"), BorderLayout.PAGE_END);
        p.add(new JButton(”Button 3"), BorderLayout.LINE_START);
        p.add(new JButton(”Button 4"), BorderLayout.LINE_END);
        p.add(new JButton(”Button 5"), BorderLayout.CENTER);
        add(p);

        setTitle("BorderLayout in Java Example");
        setSize(400,150);
        setVisible(true);
    }


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

FLOWLAYOUT

- The FlowLayout is used to arrange the components in a line, one after another (in a flow).
- It is the default layout of panel.
- Flow layout puts components (such as text fields, buttons, labels, etc) in a row, if horizontal space is not enough to hold all components then Flow layout adds them in a next row and so on.
- All rows in Flow layout are center aligned by default.

FLOWLAYOUT EXAMPLE

```
public class MyFlowLayout{
        public MyFlowLayout(){
                JFrame f=new JFrame();

                JTextField t1=new JTextField(5);
                JTextField t2=new JTextField(10);
                JButton b3=new JButton("3");
                JButton b4=new JButton("4");
                JLabel l1=new JLabel("L1");

                f.add(t1);
                f.add(t2);
                f.add(b3);
                f.add(b4);
                f.add(l1);

                f.setLayout(new FlowLayout());
                f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                f.setSize(300,300);
                f.setVisible(true);
        }

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

OUTPUT

THANK YOU!

Any questions?