

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.
- 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 Most of the AWT components have become out of date and should be replaced by newer Swing components.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



Buttons



Combo Box



List



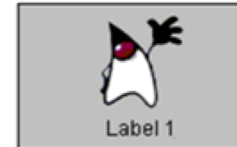
TextField



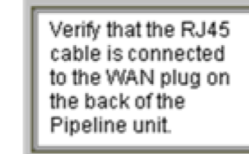
Slider



Menu



Label



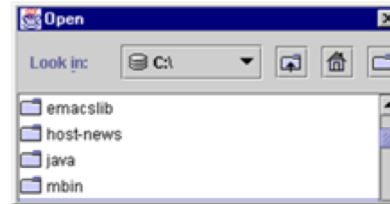
Text Area



Tool Tip



Progress Bar



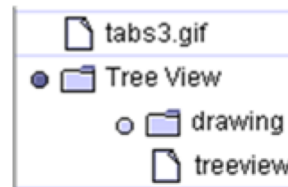
File Chooser



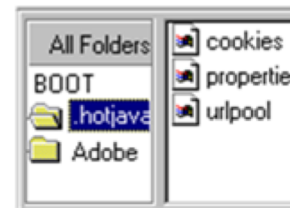
Color Chooser

First Na...	Last Name
Mark	Andrews
Tom	Ball
Alan	Chung
Jeff	Dinkins

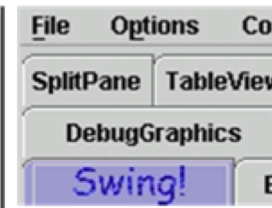
Table



Tree

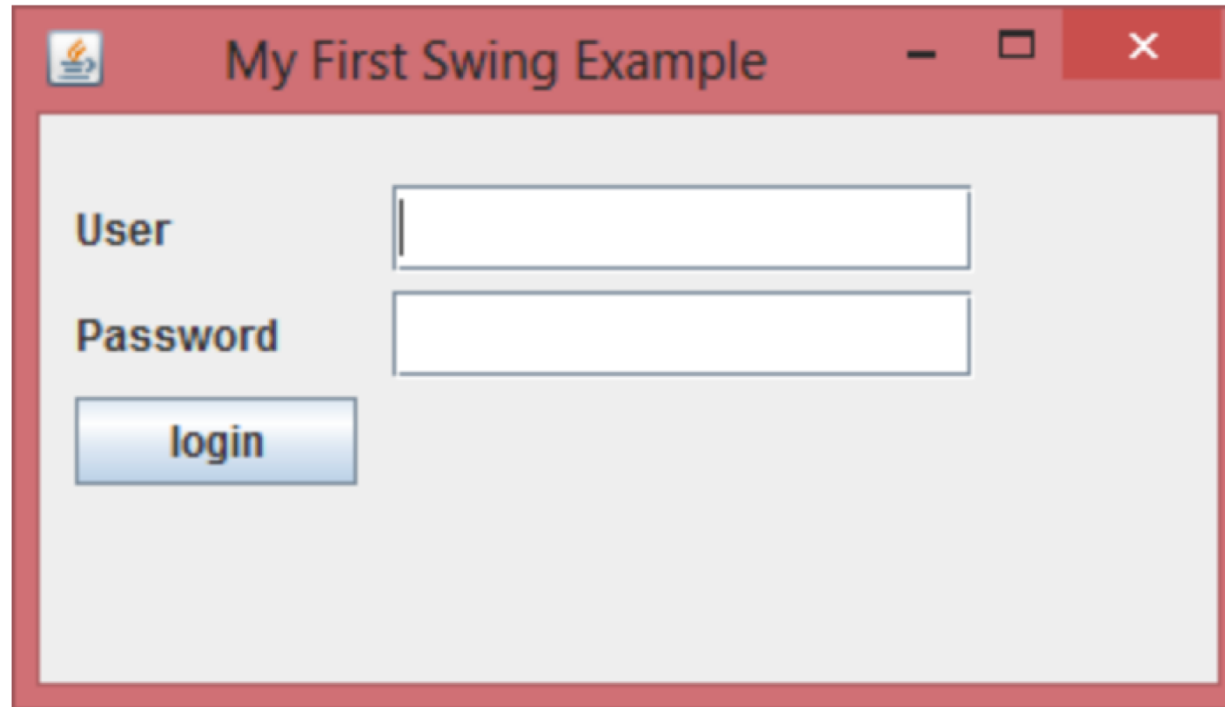


Split Pane



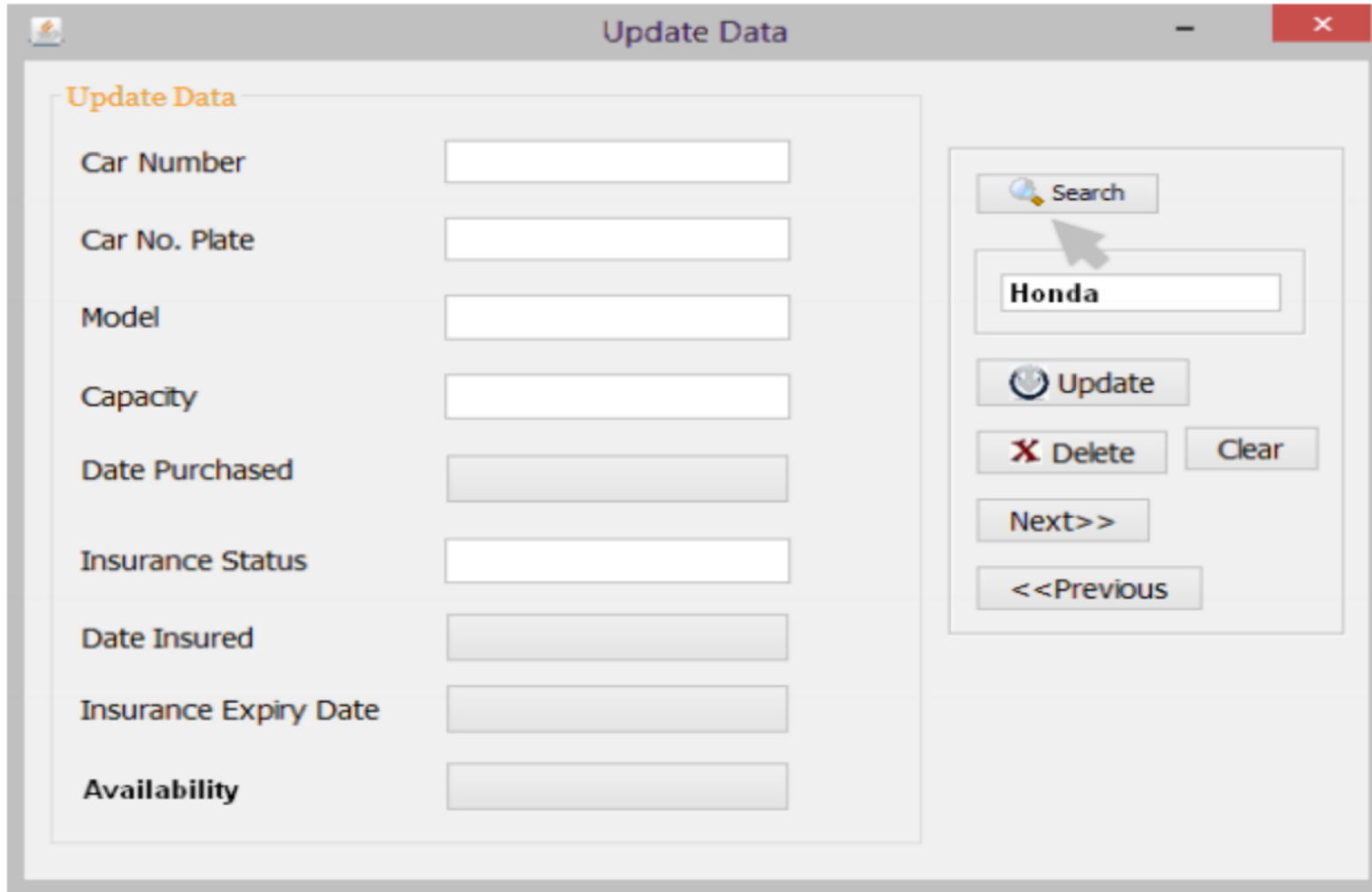
Tabbed Pane

EXAMPLE



A screenshot of a Java Swing window titled "My First Swing Example". The window has a red title bar with standard Windows-style controls (minimize, maximize, close). Inside the window, there is a light gray panel containing a login form. The form consists of two text input fields: the first is labeled "User" and the second is labeled "Password". Below these fields is a blue button with the text "login".

EXAMPLE



The screenshot shows a software window titled "Update Data" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, the title "Update Data" is repeated in orange text. The main area is divided into two sections. The left section contains a list of car attributes, each followed by an input field: "Car Number", "Car No. Plate", "Model", "Capacity", "Date Purchased", "Insurance Status", "Date Insured", "Insurance Expiry Date", and "Availability". The right section contains a set of control buttons: a "Search" button with a magnifying glass icon, a text box containing the word "Honda", an "Update" button with a circular arrow icon, a "Delete" button with a red 'X' icon, a "Clear" button, a "Next>>" button, and a "<<Previous" button. A mouse cursor is pointing at the "Search" button.

Field Label	Field Type
Car Number	Text Input
Car No. Plate	Text Input
Model	Text Input
Capacity	Text Input
Date Purchased	Date Picker
Insurance Status	Text Input
Date Insured	Date Picker
Insurance Expiry Date	Date Picker
Availability	Text Input

Control Buttons:

- Search (Magnifying Glass Icon)
- Update (Circular Arrow Icon)
- Delete (Red X Icon)
- Clear
- Next>>
- <<Previous

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("Welcome to Java Swing");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
        frame.setVisible(true);
    }
}
```

OUTPUT



JFRAME FROM INHERITANCE

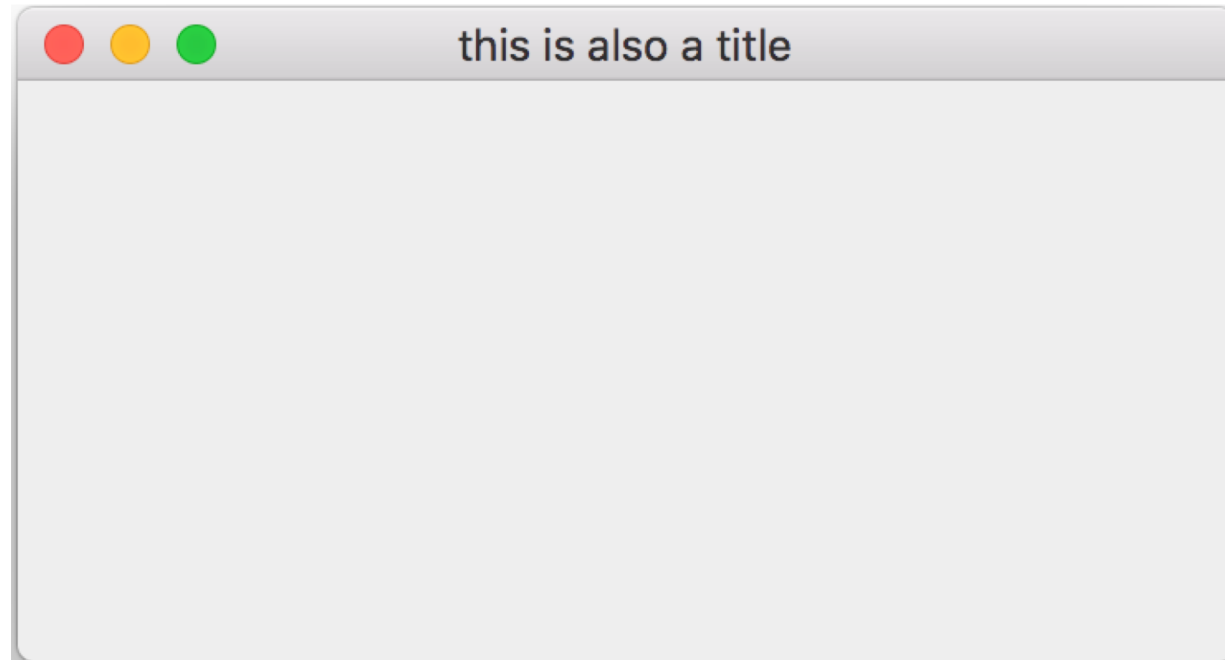
```
import javax.swing.*;

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



JFRAME

```
import javax.swing.*;

public class FrameDemo 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) {
        FrameDemo demo = new FrameDemo();
        demo.createFrame();
    }
}
```

```
import javax.swing.JFrame;

public class JFrameExample {

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

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();
```

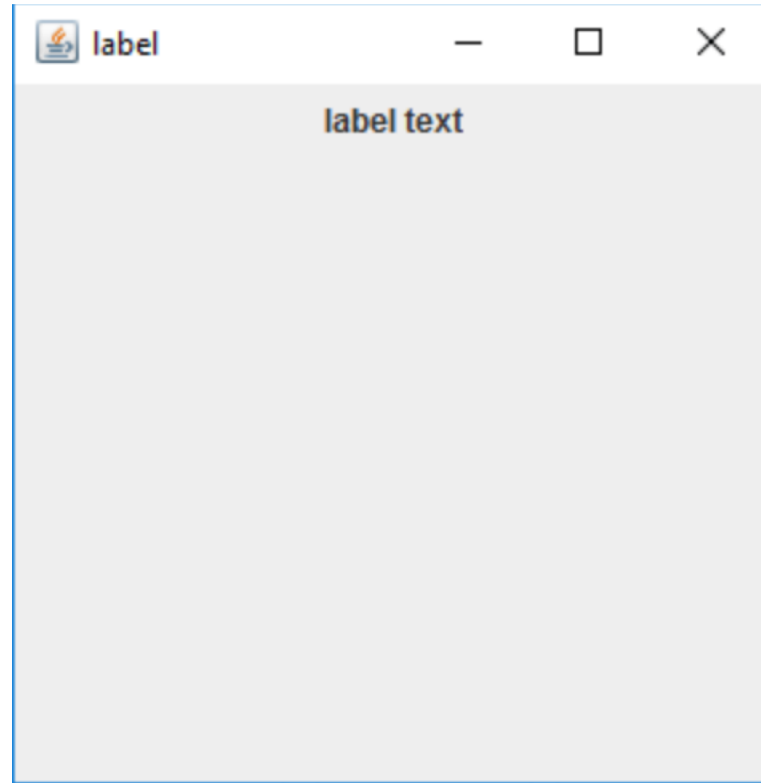
JLABEL

- 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 frame = new JFrame("label");  
        // create a label to display text  
        JLabel label = new JLabel();  
        // add text to label  
        label.setText("label text");  
  
        // create a panel  
        JPanel panel = new JPanel();  
        // add label to panel  
        panel.add(label);  
        // add panel to frame  
        frame.add(panel);  
  
        // set the size of frame  
        frame.setSize(300, 300);  
        frame.setVisible(true);  
    }  
}
```

OUTPUT

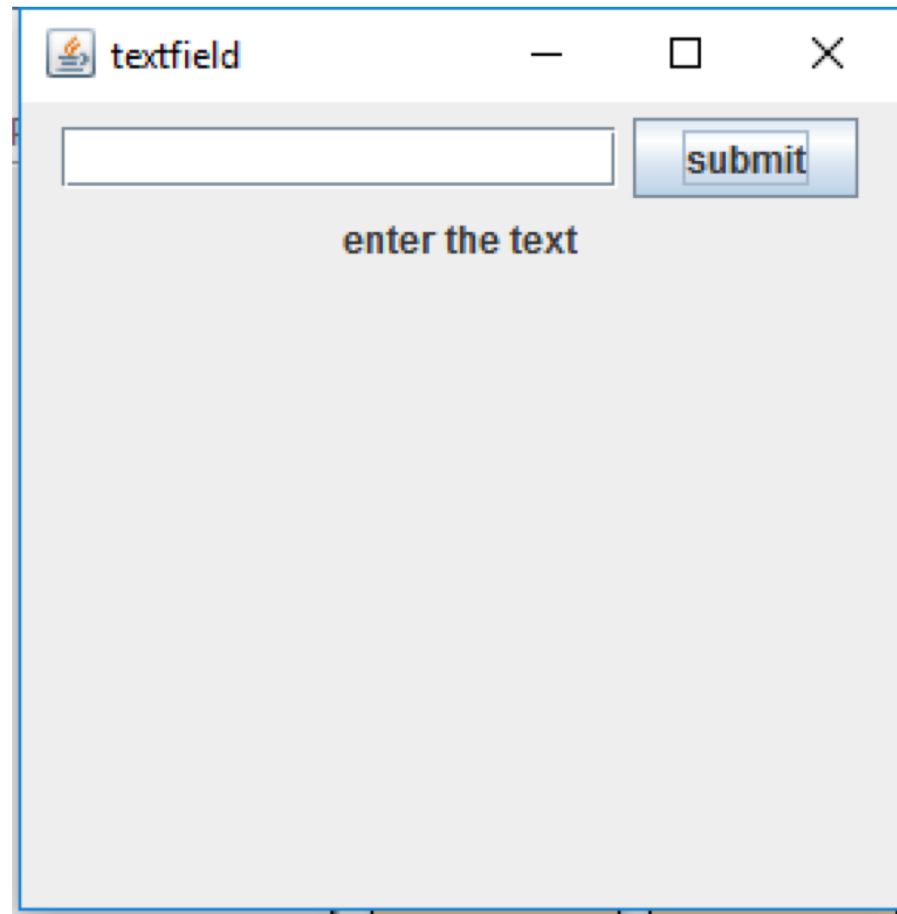


JTEXTFIELD

- 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

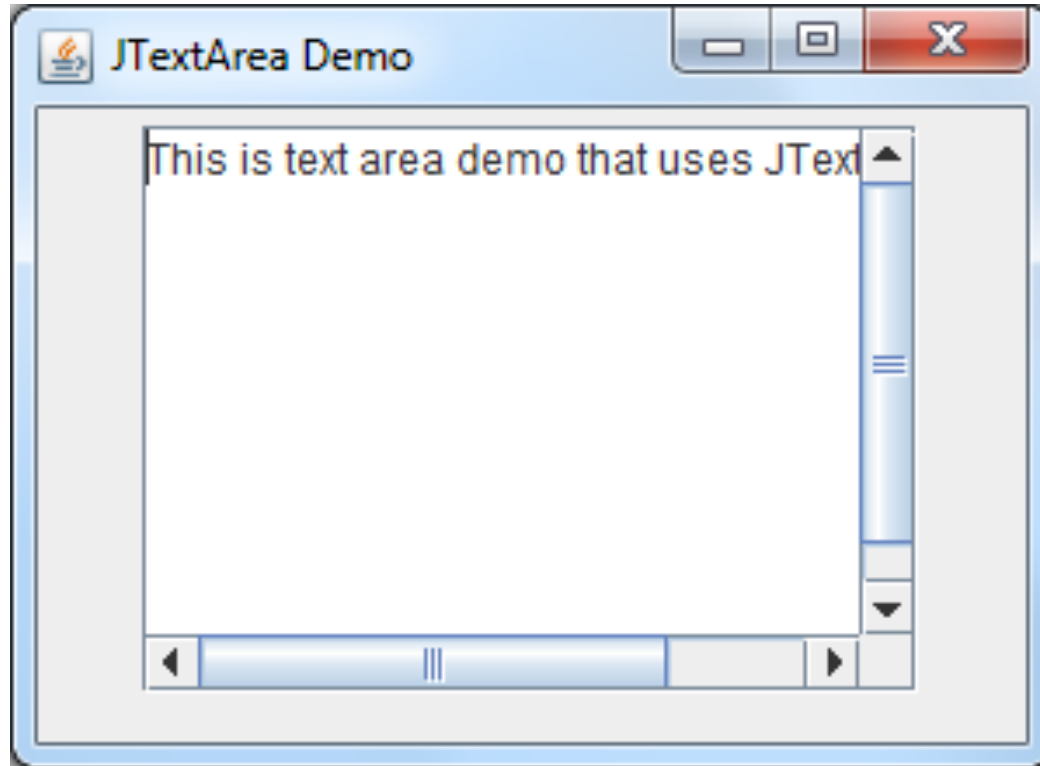


JTEXTAREA

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.

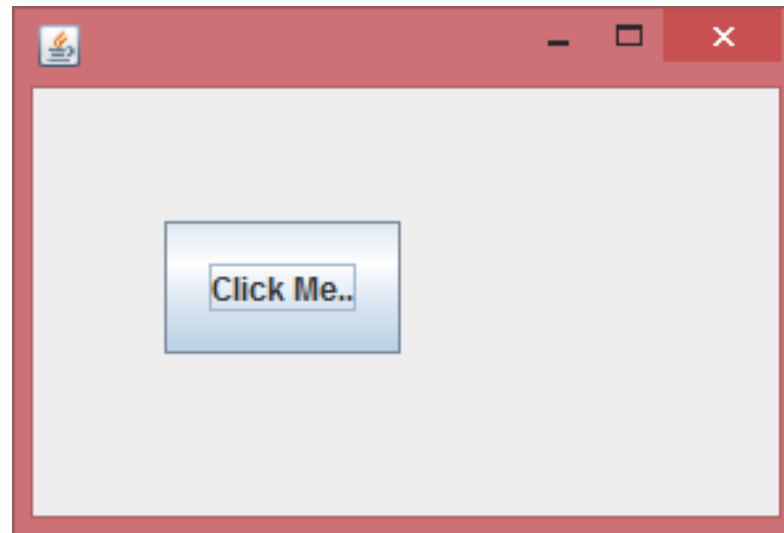
JBUTTON

```
import javax.swing.JButton;
import javax.swing.JFrame;
public class JButtonExample {
    JButtonExample(){
        JFrame frame=new JFrame();
        JButton button=new JButton("Click Me..");

        //Adding button onto the frame
        frame.add(button);
        frame.setSize(300,200);
        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 redButton = new JButton("Red");  
        redButton.addActionListener(this);  
  
        add(redButton);  
        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 frame = new JFrame("Button Example");  
        JTextField textField = new JTextField();  
  
        JButton button = new JButton("Click Here");  
  
        // Adding Action Listener  
        button.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                textField.setText("Welcome to Java Swing.");  
            }  
        });  
  
        frame.add(button);  
        frame.add(textField);  
        frame.setSize(400, 400);  
        frame.setVisible(true);  
    }  
}
```

IMPLEMENTING ACTIONLISTENER

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

        JFrame frame = new JFrame("Button Example");
        JButton button = new JButton("Red");

        // Adding Action Listener
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Welcome to GUI");
            }
        });

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

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

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

        add(button);
        setTitle("Button Example");
        setSize(400, 400);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        System.out.println("Welcome to GUI");
    }

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

WHAT IS ANONYMOUS?

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

PROVIDING IMPLEMENTATION

```
public class ImplementationDemo implements Manageable {  
  
    @Override  
    public void manage() {  
        System.out.println("It is manageable");  
    }  
  
    public static void main(String[] args) {  
        ImplementationDemo id = new ImplementationDemo();  
        id.manage();  
    }  
}
```

OUTPUT

It is manageable

IMPLEMENTING ANONYMOUSLY

```
public class AnonymousInterfaceDemo {  
    public static void main(String[] args) {  
        Manageable m = new Manageable() {  
            public void manage(){  
                System.out.println("It is manageable");  
            }  
        };  
        m.manage();  
    }  
}
```

OUTPUT

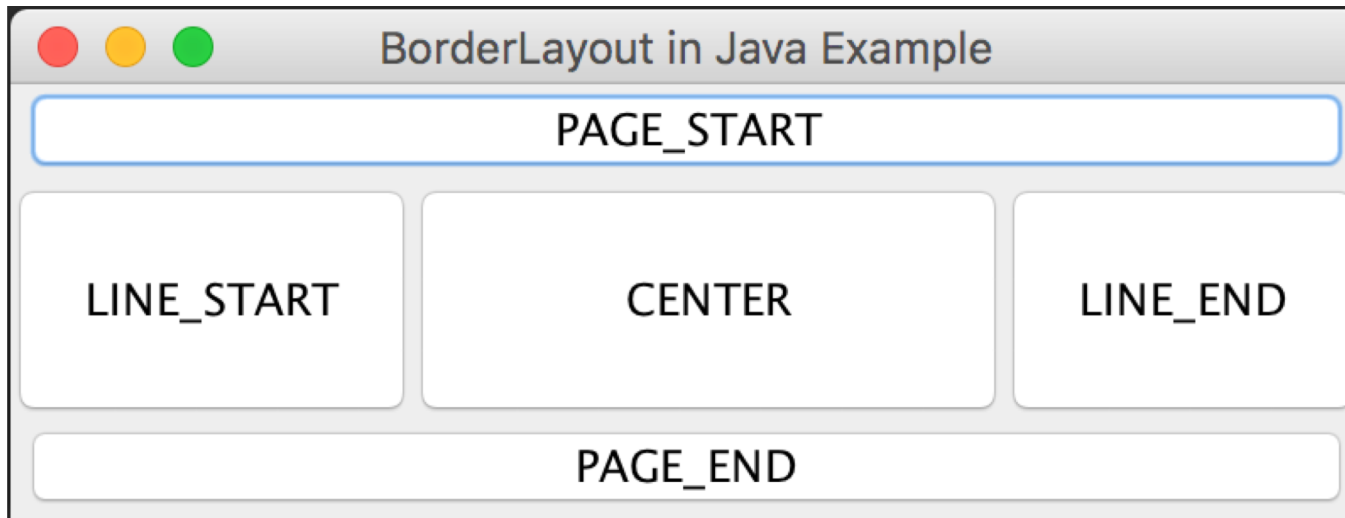
It is manageable

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 BorderLayoutDemo extends JFrame {  
    public BorderLayoutDemo() {  
        JPanel panel = new JPanel();  
        BorderLayout borderLayout = new BorderLayout();  
        panel.setLayout(borderLayout);  
  
        panel.add(new JButton("Button 1"), BorderLayout.PAGE_START);  
        panel.add(new JButton("Button 2"), BorderLayout.PAGE_END);  
        panel.add(new JButton("Button 3"), BorderLayout.LINE_START);  
        panel.add(new JButton("Button 4"), BorderLayout.LINE_END);  
        panel.add(new JButton("Button 5"), BorderLayout.CENTER);  
        add(panel);  
  
        setTitle("BorderLayout in Java Example");  
        setSize(400,150);  
        setVisible(true);  
    }  
  
    public static void main(String args[]) {  
        new BorderLayoutDemo();  
    }  
}
```

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 frame=new JFrame();

        JTextField tf1=new JTextField(5);
        JTextField tf2=new JTextField(10);
        JButton button1=new JButton("3");
        JButton button2=new JButton("4");
        JLabel label1=new JLabel("L1");

        frame.add(tf1);
        frame.add(tf2);
        frame.add(button1);
        frame.add(button2);
        frame.add(label1);

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

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

OUTPUT



THANK YOU!

Any questions?