

Object Oriented Programming using Java

Prepared By:
Suyel, PhD
Assistant Professor
Dept. of CSE, NIT Patna



Outline

1. Basics of Swing
2. Components and Containers
3. A Simple Swing Program
4. Painting in Swing
5. Swing Components
6. Displaying Graphics
7. Changing Icon of the TitleBar

Basics of Swing

- ❑ AWT defines a basic set of controls, windows and dialog boxes that support a usable, but limited graphical interface.
- ❑ Swing is a set of classes that provides more powerful and flexible GUI components than AWT.
- ❑ Swing is built on the foundation of the AWT.
- ❑ Two key features of Swing:
 - ❖ **Components are Lightweight:** This means that they are written entirely in Java and do not map directly to platform-specific peers.
 - ❖ **Supports Pluggable Look and Feel (PLAF):** It means that it is possible to separate the look and feel of a component from the logic of the components.

Basics of Swing (Cont...)

❑ Model View Controller (MVC) Connection:

- ❖ A visual component is a composite of three distinct aspects:
 1. The way that the component looks when rendered on the screen
 2. The way that the component reacts to the user
 3. The state information associated with the component
- ❖ MVC architecture has proven itself to be exceptionally effective.
- ❖ Swing uses a modified version of MVC that combines the view and the controller into a single logical entity called the UI delegate. So, Swing's approach is known as the Model-Delegate architecture.
- ❖ To support the Model-Delegate architecture, most Swing components contain two objects:
 1. **Model:** Models are defined by interfaces.
 2. **UI delegate:** UI delegates are classes that inherit ComponentUI.

Basics of Swing (Cont...)

❑ Differences between AWT and Swing

AWT	Swing
AWT components are platform dependent.	Swing components are platform independent.
AWT components are heavyweight.	Swing components are lightweight.
AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
AWT provides less components than Swing.	Swing provides more powerful components, such as tables, lists, scrollpanes, colorchooser, tabbedpane, etc.
AWT doesn't follow MVC.	Swing follows MVC. In fact, advanced version of MVC.

Components and Containers

- ❑ A Swing GUI consists of two key items: components and containers.
- ❑ The difference between the key items is found in their intended purpose.
- ❑ A component is an independent visual control, such as a push button.
- ❑ A container holds a group of components. Thus, a container is a special type of component that holds other components.
- ❑ In order to display a component, it must be held within a container. So, all Swing GUIs have at least one container.

Components and Containers (Cont...)

❑ Components:

- ❖ Swing components are derived from the JComponent class (exception is the top level containers).
- ❖ JComponent provides the functionality that is common to all components.
- ❖ JComponent inherits the AWT classes Container and Component. Thus, a Swing component is compatible with an AWT component.
- ❖ All of Swing's components are represented by classes defined within the package javax.swing.
- ❖ Some class names for Swing components are: JFrame, JList, JApplet, JMenu, JTable, JMenuBar, JTree and many more.

Components and Containers (Cont...)

❑ Containers:

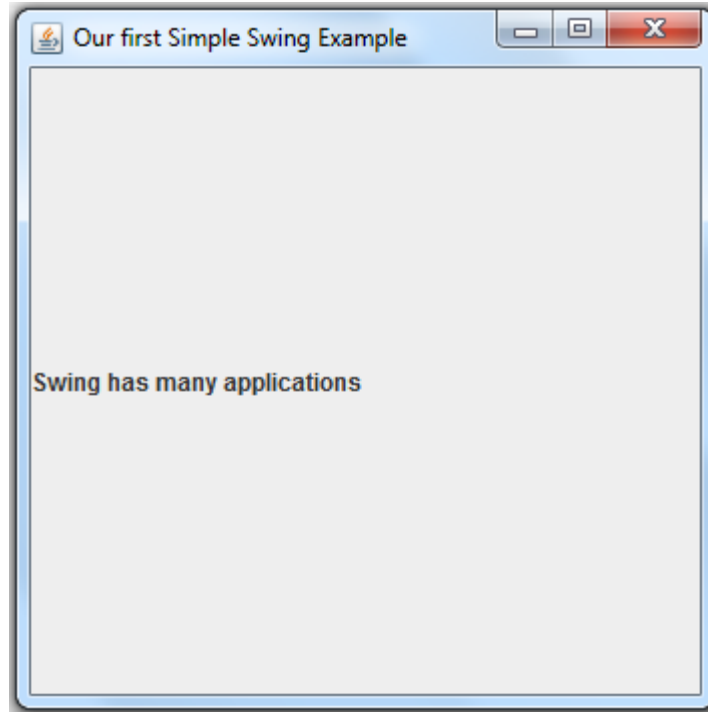
- ❖ Swing defines two types of containers.
- ❖ The first are top-level containers: JFrame, JApplet, JWindow and JDialog. These containers do not inherit JComponent. However, they inherit the AWT classes Component and Container.
- ❖ Unlike Swing's other components, the top-level containers are heavyweight and they are not contained within any other container.
- ❖ The second type of containers supported by Swing are lightweight containers. They inherit JComponent. Example: JPanel.
- ❖ Lightweight containers are often used to organize and manage groups of related components because a lightweight container can be contained within another container.

A Simple Swing Program

```
import javax.swing.*;
class Swing1
{
    Swing1()
    {
        JFrame f = new JFrame("Our first Simple Swing Example"); //Create a new JFrame container
        f.setSize(350, 350); //Give the frame an initial size
        f.setVisible(true); //Display the frame
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Terminate the program
        JLabel l = new JLabel("Swing has many applications"); //Create a text-based label
        f.add(l); //Add the label to the content pane
    }
    public static void main(String args[])
    {
        new Swing1();
    }
}
```

A Simple Swing Program (Cont...)

❑ Output:



A Simple Swing Program (Cont...)

❑ **setDefaultCloseOperation()**

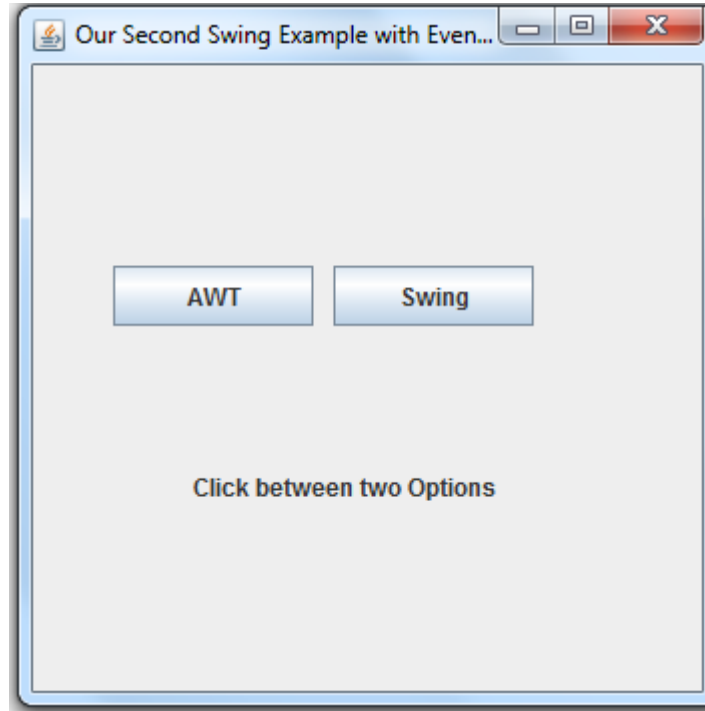
- ❖ `setDefaultCloseOperation()` method is used to specify one of several options for the close button. We can use one of the four forms:
 - `JFrame.EXIT_ON_CLOSE` //Exit the application
 - `JFrame.HIDE_ON_CLOSE` //Hide the frame, but keep the application running
 - `JFrame.DISPOSE_ON_CLOSE` //Dispose of the frame object, but keep the application running.
 - `JFrame.DO_NOTHING_ON_CLOSE` //Ignore the click.
- ❖ If we forget to call `setDefaultCloseOperation()`, by default `JFrame.HIDE_ON_CLOSE` is called. So, program is running, but we cannot see frame.

A Simple Swing Program (Cont...)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class Swing2
{
    Swing2()
    {
        JFrame f = new JFrame("Our Second Swing Example with Event Handling");
        JLabel l=new JLabel("Click between two options");
        l.setBounds(80, 200, 200, 20);
        JButton b1= new JButton("AWT");
        b1.setBounds(40, 100, 100, 30);
        JButton b2= new JButton("Swing");
        b2.setBounds(150, 100, 100, 30);
        b1.addActionListener(new ActionListener()
        {
            public void actionPerformed (ActionEvent ae)
            {
                l.setText("You have pressed AWT");
            }
        });
        b2.addActionListener(new ActionListener()
        {
            public void actionPerformed (ActionEvent ae)
            {
                l.setText("Swing is pressed");
            }
        });
        f.add(b1); f.add(b2); f.add(l);
        f.setSize(350, 350);
        f.setLayout(null);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[])
    {
        new Swing2();
    }
}
```

A Simple Swing Program (Cont...)

□ Output:



When we will click the buttons, different comments will be displayed.

Painting in Swing

- ❑ Swing's approach for painting is built on the original AWT-based mechanism.
- ❑ To write output directly to the surface of a component, we use one or more drawing methods defined by the AWT.
- ❑ As JComponent inherits Component, all Swing's lightweight components inherit the `paint()` method.
- ❑ Swing uses more sophisticated approach to paint that involves 3 distinct methods: `paintComponent()`, `paintBorder()` and `paintChildren()`.
- ❑ These methods paint the indicated portion of a component and divide the painting process into its 3 logical actions.

Painting in Swing (Cont...)

- ❑ To paint to the surface of a Swing component, we create a subclass of the component. Then, override its `paintComponent()`.
- ❑ The above mentioned process is used to paint the interior of the component. Here, we do not normally override the other two painting methods.
- ❑ When overriding `paintComponent()`, we must first call `super.paintComponent()`, so that the superclass portion of the painting process takes place. Then, we write the output that we want to display.

```
void paintComponent(Graphics g)
```

Painting in Swing (Cont...)

- ❑ To cause a component to be painted under program control, we call `repaint()`, which is defined by `Component`.
- ❑ Calling `repaint()` causes the system to call `paint()` as soon as it is possible to do so.
- ❑ Because painting is a time-consuming operation, this mechanism allows the run-time system to defer painting momentarily until some higher-priority tasks are completed.
- ❑ To output to the surface of a component, the program stores the output until `paintComponent()` is called.
- ❑ Inside the overridden `paintComponent()`, we draw the stored output.

Painting in Swing (Cont...)

❑ Compute the Paintable Area:

- ❖ When drawing to the surface of a component, we must be careful to restrict the output to the area that is inside the border.
- ❖ Here, we compute the paintable area of the component. This is the area defined by:

Current size of the component minus space used by the border

- ❖ Therefore, we must obtain the width of the border by `getInsets()` before painting to a component. Then, adjust the drawing accordingly.

`Insets insets = component.getInsets();`

This method is defined by `Container` and overridden by `JComponent`.

It returns an `Insets` object that contains the dimensions of the border.

Painting in Swing (Cont...)

❑ Compute the Paintable Area (Cont...):

- ❖ The inset values can be obtained by using these fields:

```
int top;  
int bottom;  
int left;  
int right;
```

The above mentioned values are then used to compute the drawing area given the width and the height of the component.

- ❖ We can obtain the width and height of the component by calling `getWidth()` and `getHeight()` on the component. They are shown here:

```
int getWidth( )  
int getHeight( )
```

Painting in Swing (Cont...)

```

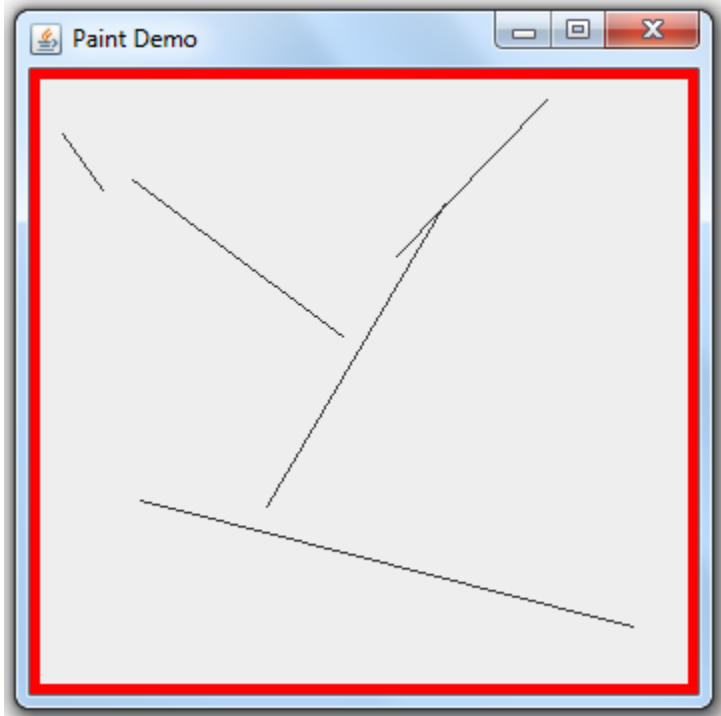
import java.awt.*;    import java.awt.event.*;
import javax.swing.*; import java.util.*;
class Paint1 extends JPanel
{
    Insets ins;
    Random r;
    public Paint1()
    {
        //Holds the panel's insets
        //used to generate random numbers
        // Construct a panel
        setBorder(BorderFactory.createLineBorder(Color.RED, 5)); //Put a border around the panel
        r = new Random();
    }
    protected void paintComponent(Graphics g)
    {
        //Override the paintComponent() method
        super.paintComponent(g);
        //Always call the superclass method first
        int x1, y1, x2, y2;
        int height = getHeight();
        int width = getWidth();
        //Get the height and width of the component
        ins = getInsets();
        // Get the insets
        for(int i=0; i < 5; i++)
        {
            //Draw 5 lines whose endpoints are randomly generated
            x1 = r.nextInt(width-ins.left);
            y1 = r.nextInt(height-ins.bottom);
            x2 = r.nextInt(width-ins.left);
            y2 = r.nextInt(height-ins.bottom);
            g.drawLine(x1, y1, x2, y2);
            //Draw the line
        }
    }
}
class PaintMain
{
    //Demonstrate painting directly onto a panel
    Paint1 pp;
    PaintMain()
    {
        JFrame f = new JFrame("Paint Demo");
        //Create a new JFrame container
        f.setSize(350, 350);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pp = new Paint1();
        //Create the panel that will be painted
        f.add(pp);
    }
    public static void main(String args[])
    {
        new PaintMain();
    }
}

```

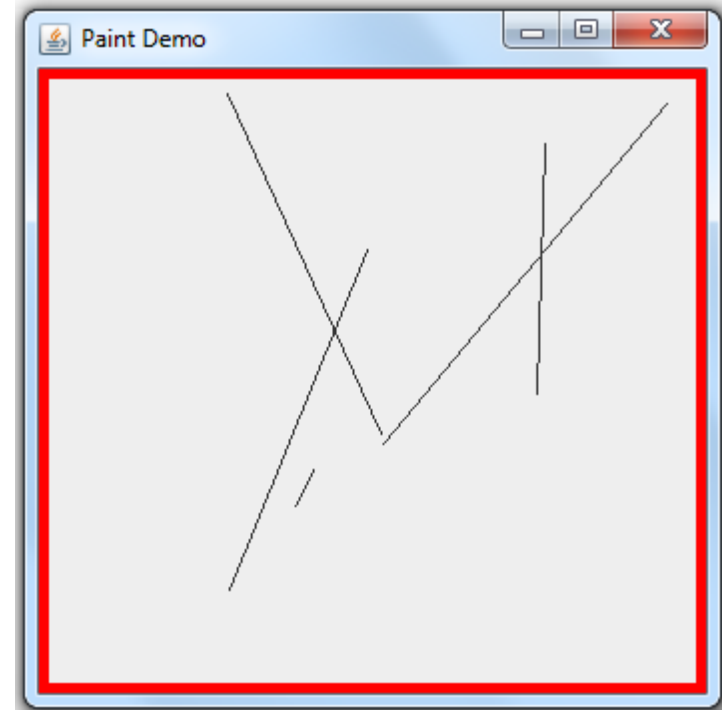
Painting in Swing (Cont...)

□ Output

1st Run



2nd Run



Swing Components

❑ JLabel and ImageIcon

- ❖ It is a passive component that does not respond to user input.

- ❖ JLabel defines several constructors. Three of them are:

JLabel(Icon *icon*) //*icon* is specified by object type Icon. It is the image on the label

JLabel(String *str*) //*str* is the string within the label

JLabel(String *str*, Icon *icon*, int *align*) //Horizontal alignment. It must be among LEFT, RIGHT, CENTER, LEADING or TRAILING

- ❖ The easiest way to obtain an icon is to use the ImageIcon class. An object of type ImageIcon can be passed as an argument to the Icon parameter of JLabel's constructor.

ImageIcon(String *filename*) //*filename* is the image name.

Swing Components (Cont...)

❑ JLabel and ImageIcon (Cont...)

❖ Methods:

Icon getIcon(): This method is used to get an Image on the Label.

String getText(): It is used to get String within the Label.

void setIcon(Icon *icon*): It is used to set an Image on a Label.

void setText(String *str*): This method is used to set or change the text within a Label.

Swing Components (Cont...)

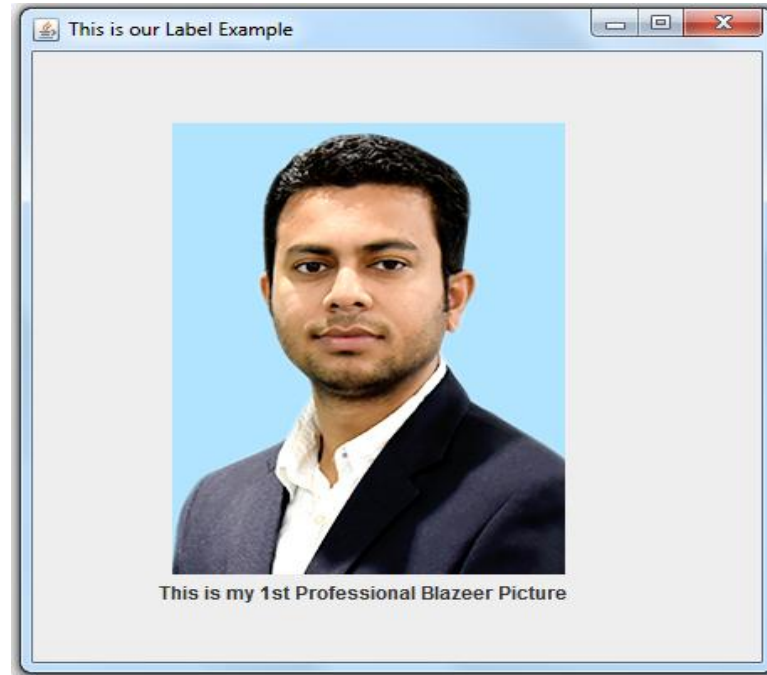
❑ JLabel and ImageIcon (Cont...)

```
import java.awt.*;
import javax.swing.*;
class Label1
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("This is our Label Example");
        JLabel l1,l2;
        l1=new JLabel("This is my 1st Professional Blazeer Picture");
        l1.setBounds(75, 350, 300, 30);
        ImageIcon icon = new ImageIcon("C:/Users/SUYEL/Desktop/pass.jpg");
        l2=new JLabel(icon);
        l2.setBounds(0, 0, 400, 400);
        f.add(l1); f.add(l2);
        f.setSize(450, 450);
        f.setLayout(null);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Swing Components (Cont...)

❑ JLabel and ImageIcon (Cont...)

❖ Output:



Swing Components (Cont...)

❑ JTextField:

- ❖ JTextField is the simplest Swing text component.
- ❖ It allows to edit one line of text, and it is derived from JTextComponent.
- ❖ Three of JTextField's constructors are given below:

<code>JTextField(int cols)</code>	<code>//Cols is the number of columns in the text field</code>
<code>JTextField(String str, int cols)</code>	<code>//str is the string to be initially presented</code>
<code>JTextField(String str)</code>	
- ❖ JTextField generates events in response to user interaction.
- ❖ `getText()` method is used to obtain the text currently in the text field.

Swing Components (Cont...)

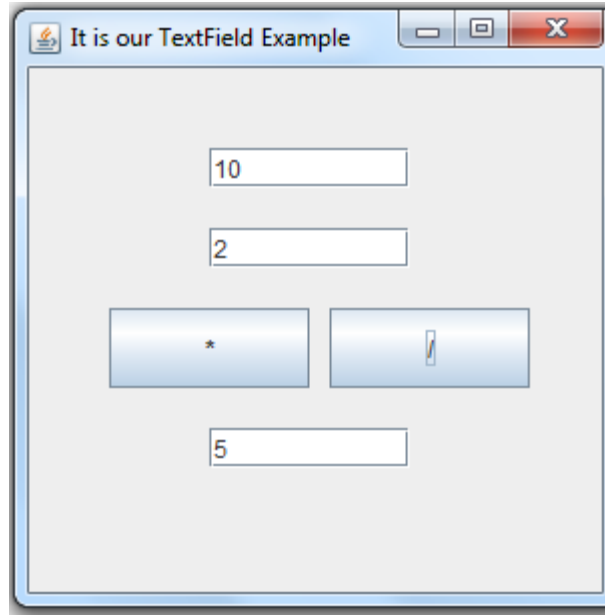
□ JTextField (Cont...):

```
import javax.swing.*;
import java.awt.event.*;
public class TextField1 implements ActionListener
{
    JTextField f1, f2, f3;
    JButton b1, b2;
    TextField1()
    {
        JFrame f= new JFrame("It is our TextField Example");
        f1=new JTextField();
        f1.setBounds(90, 40, 100, 20);
        f2=new JTextField();
        f2.setBounds(90, 80, 100, 20);
        f3=new JTextField();
        f3.setBounds(90, 180, 100, 20);
        b1=new JButton("*");
        b1.setBounds(40, 120, 100, 40);
        b2=new JButton("/");
        b2.setBounds(150, 120, 100, 40);
        b1.addActionListener(this);
        b2.addActionListener(this);
        f.add(f1); f.add(f2); f.add(f3); f.add(b1); f.add(b2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void actionPerformed(ActionEvent ae)
    {
        int c=0;
        String s1=f1.getText();
        String s2=f2.getText();
        int a=Integer.parseInt(s1);
        int b=Integer.parseInt(s2);
        if(ae.getSource()==b1)
        {
            c=a*b;
        } else if(ae.getSource()==b2)
        {
            c=a/b;
        }
        String s3=String.valueOf(c);
        f3.setText(s3);
    }
    public static void main(String[] args)
    {
        new TextField1();
    }
}
```

Swing Components (Cont...)

❑ JTextField (Cont...):

❖ Output:



Swing Components (Cont...)

❑ JPanel:

- ❖ The JPanel is a simplest container class.
- ❖ It provides space in which an application can add any other component.
- ❖ This class inherits the JComponents class.
- ❖ There are many constructors. Two are mentioned below:

`JPanel()` //Creates a new JPanel with a double buffer and a flow layout

`JPanel(LayoutManager layout)` //Creates a new JPanel with the specified layout manager

Swing Components (Cont...)

□ JPanel (Cont...):

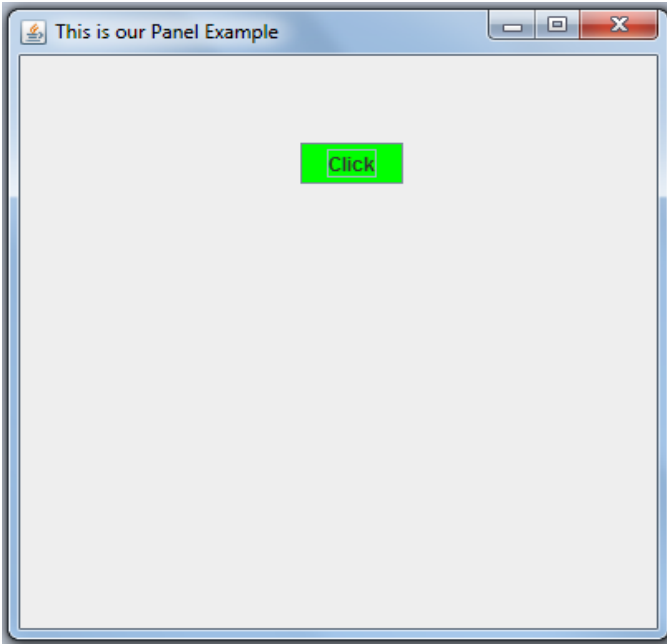
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Panel1
{
    Panel1()
    {
        JFrame f = new JFrame("This is our Panel Example");
        JPanel jp=new JPanel();
        jp.setBounds(50, 50, 300, 300);
        JButton b1=new JButton("Click");
        b1.setBackground(Color.GREEN);
        jp.add(b1);
        f.add(jp);
        f.setSize(400, 400);
        f.setLayout(null);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        b1.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                jp.setBackground(Color.BLUE);
            }
        });
    }
    public static void main(String[] args)
    {
        new Panel1();
    }
}
```

Swing Components (Cont...)

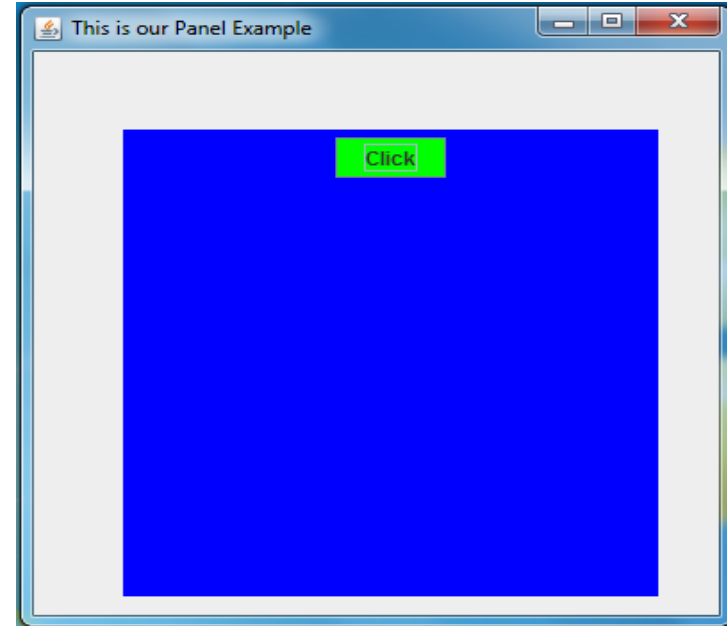
❑ JPanel (Cont...):

❖ Output

Before Clicking Button



After Clicking Button



Swing Components (Cont...)

❑ Swing Buttons:

- ❖ Swing defines four types of buttons: JButton, JToggleButton, JCheckBox and JRadioButton.
- ❖ All these buttons are subclasses of the AbstractButton class, which extends JComponent.
- ❖ AbstractButton contains many methods that allow us to control the behavior of buttons.
- ❖ We can define different icons that are displayed for button, when it is disabled, pressed or selected. The following methods set these icons:

```
void setDisabledIcon(Icon di)
```

```
void setPressedIcon(Icon pi)
```

```
void setSelectedIcon(Icon si)
```

```
void setRolloverIcon(Icon ri) //This is used as a rollover icon, which is displayed  
when the mouse is positioned over a button.
```

Swing Components (Cont...)

❑ Swing Buttons (Cont...):

- ❖ **String getText()**: It is used to read text associated with a button.
- ❖ **void setText(String *str*)**: Used to write text assigned with a button.
- ❖ All the buttons are defined by the ButtonModel interface.
- ❖ **JButton**:
 - JButton class provides the functionality of a push button.
 - It allows an icon, a string or both to associate with the push button.
Three of its constructors are mentioned below:
JButton(Icon *icon*)
JButton(String *str*)
JButton(String *str*, Icon *icon*)
 - **setActionCommand()** and **getActionCommand()** are used to set and get action command associated with a button, respectively.

Swing Components (Cont...)

❑ Swing Buttons (Cont...): JButton (Cont...):

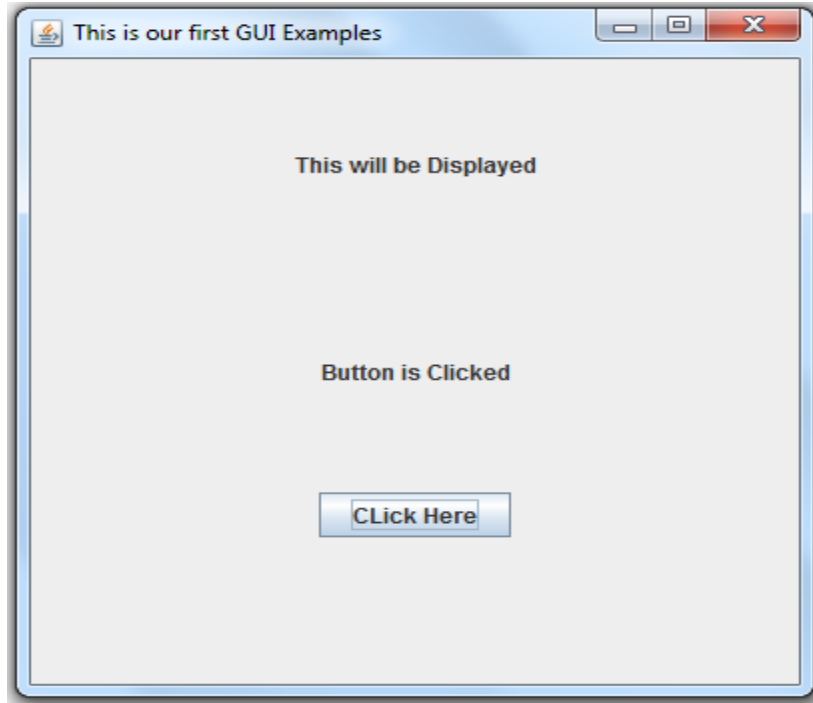
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Button1
{
    JFrame f; JLabel l1; JLabel l2; JPanel p1;
    public Button1()
    {
        FirstGUI();
    }
    public void FirstGUI()
    {
        f = new JFrame("This is our first GUI Examples");
        l1 = new JLabel("This will be Displayed", JLabel.CENTER );
        l2 = new JLabel(" ", JLabel.CENTER);
        p1 = new JPanel();
        p1.setLayout(new FlowLayout());
        f.add(l1); f.add(l2); f.add(p1);
        f.setSize(400,400);
        f.setLayout(new GridLayout(3, 1));
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void showActionListenerDemo()
    {
        JPanel p2 = new JPanel();
        JButton b1 = new JButton("Click Here");
        b1.addActionListener(new CustomActionListener());
        p2.add(b1);
        p1.add(p2);
    }
    class CustomActionListener implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            l2.setText("Button is clicked");
        }
    }
    public static void main(String[] args)
    {
        Button1 obj_button1 = new Button1();
        obj_button1.showActionListenerDemo();
    }
}
```

Swing Components (Cont...)

❑ Swing Buttons (Cont...):

❖ JButton (Cont...):

Output



Swing Components (Cont...)

❑ Swing Buttons (Cont...):

❖ JToggleButton:

- A toggle button looks just like a push button, but it acts differently because it has two states: pushed and released.
- When, we press a toggle button for the first time, it stays pressed rather than popping back up as a regular push button does.
- When we press the toggle button for the second time, it releases pops up.
- Toggle buttons are objects of the JToggleButton class and it implements AbstractButton.
- JToggleButton is a superclass for two Swing components that also represent two-state controls. These are JCheckBox and JRadioButton.

Swing Components (Cont...)

❑ Swing Buttons (Cont...):

❖ JToggleButton (Cont...):

- When a JToggleButton is pressed in, it is selected. When it is popped out, it is deselected.
- JToggleButton defines several constructors. One is defined below:

```
JToggleButton(String str) //Creates a toggle button that contains the text passed in str.
```

By default, the button is in the off position.
- Like JButton, JToggleButton generates an action event each time it is pressed.
- Unlike JButton, JToggleButton also generates an item event. This event is used by the components that support the concept of selection.

Swing Components (Cont...)

❑ Swing Buttons (Cont...):

❖ JToggleButton (Cont...):

- To handle item events, we must implement the `ItemListener` interface.
- Each time an item event is generated, it is passed to the `itemStateChanged()` method defined by `ItemListener`.
- The easiest way to determine a toggle button's state is by calling the `isSelected()` method, which is inherited from `AbstractButton` on the button that generated the event. It is shown below:

`boolean isSelected()` //Returns true, if the button is selected. Otherwise, returns false.

Swing Components (Cont...)

❑ Swing Buttons (Cont...):

JToggleButton (Cont...):

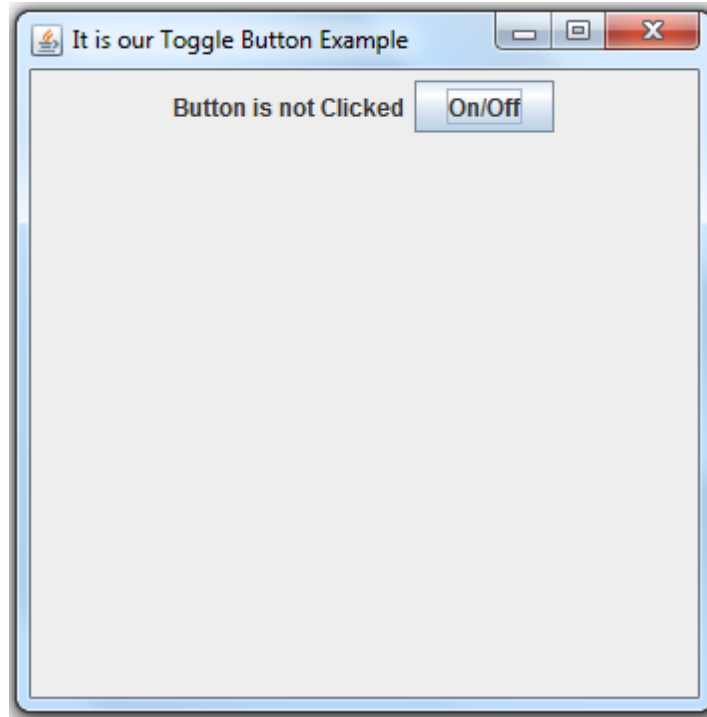
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Button2
{
    JFrame f; JLabel l; JToggleButton b;
    public Button2()
    {
        f = new JFrame("It is our Toggle Button Example");
        l = new JLabel("Button is not Clicked");
        b = new JToggleButton("On/Off");
        f.add(l); f.add(b);
        f.setSize(350, 350);
        f.setLayout(new FlowLayout());
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        b.addItemListener(new ItemListener()
        {
            public void itemStateChanged (ItemEvent ie)
            {
                if(b.isSelected())
                {
                    l.setText("Button is Clicked");
                }
                else
                {
                    l.setText("Button is not Clicked");
                }
            }
        });
    }
    public static void main(String[] args)
    {
        new Button2();
    }
}
```

Swing Components (Cont...)

❑ Swing Buttons (Cont...):

❖ JToggleButton (Cont...):

Output



Swing Components (Cont...)

❑ Swing Buttons (Cont...):

❖ JCheckBox:

- JCheckBox class provides the functionality of a check box.
- Its immediate superclass is JToggleButton.
- JCheckBox defines several constructors. Some are:

`JCheckBox()` //Creates an empty unselected check box button

`JChechBox(String str)` //Creates an initially unselected check box with text

Creates a check box with text and specifies whether it is initially
`sJCheckBox(String text, boolean selected)` //elected or not

`JCheckBox(Action a)` //Creates a check box, where properties are taken from the
 Action supplied.

Swing Components (Cont...)

❑ Swing Buttons (Cont...):

❖ JCheckBox (Cont...):

- `getText()` method is used to get the text for the check box.
- When the user clicks a check box, an `ItemEvent` is generated.
- Inside the `itemStateChanged()` method, `getItem()` is called to obtain a reference to the `JCheckBox` object that generated the event.
- Next, we can call `isSelected()` to determine if the box was selected or cleared.

Swing Components (Cont...)

❑ Swing Buttons (Cont...):

JCheckBox (Cont...):

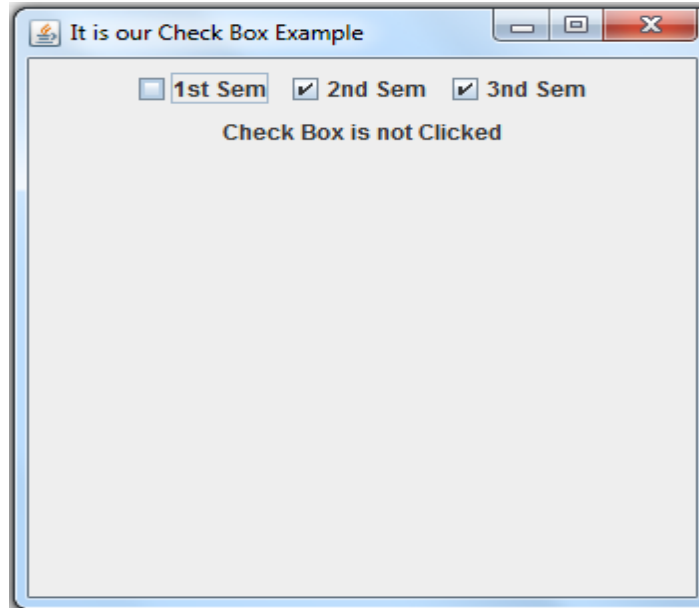
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Checkbox1 extends JFrame implements ItemListener
{
    JLabel l;
    public Checkbox1()
    {
        setTitle("It is our Check Box Example");
        l = new JLabel("Check Box is not Clicked");
        JCheckBox c = new JCheckBox("1st Sem");
        c.addItemListener(this);
        add(c);
        c = new JCheckBox("2nd Sem");
        c.addItemListener(this);
        add(c);
        c = new JCheckBox("3rd Sem");
        c.addItemListener(this);
        add(c);
        add(l);
        setSize(350, 350);
        setLayout(new FlowLayout());
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void itemStateChanged (ItemEvent ie)
    {
        JCheckBox c=(JCheckBox)ie.getItem();
        if(c.isSelected())
            l.setText(c.getText() + " is selected");
        else
            l.setText("Check Box is not Clicked");
    }
    public static void main(String[] args)
    {
        new Checkbox1();
    }
}
```

Swing Components (Cont...)

❑ Swing Buttons (Cont...):

❖ JCheckBox (Cont...):

Output



This program always keeps the updated information of a Check Box.

Swing Components (Cont...)

❑ Swing Buttons (Cont...):

❖ Radio Buttons:

- Radio buttons are a group of mutually exclusive buttons in which only one button can be selected at a time.
- Radio buttons are supported by the `JRadioButton` class, which extends `JToggleButton`.
- `JRadioButton` provides several constructors. Some are:
 - `JRadioButton()` //Creates an unselected radio button with no text
 - `JRadioButton(String str)` //Creates an unselected radio button with specified text
 - `JRadioButton(String str, boolean selected)` //Creates a radio button with the specified text and selected status.
- In order for their mutually exclusive nature to be activated, radio buttons must be configured into a group.

Swing Components (Cont...)

❑ Swing Buttons (Cont...):

❖ Radio Buttons (Cont...):

- A button group is created by the `ButtonGroup` class. Elements are then added to the button group by the following method:

```
void add(AbstractButton ab)    //ab is a reference of the button to be added to the group
```

- A `JRadioButton` generates action events, item events, and changes the events each time the button selection changes.
- Normally, Radio Buttons implements `ActionListener` interface.

Swing Components (Cont...)

❑ Swing Buttons (Cont...):

❖ Radio Buttons (Cont...):

- As ActionListener defines only 1 method i.e. actionPerformed(), we can code inside this in different ways to set which button is selected.
- Here, at first, we can check the action command by calling getActionCommand(). By default, the action command is the same as the button label.
- However, we can set the action command as per our wish by calling setActionCommand() on the radio button.
- Secondly, we can call getSource() on the(ActionEvent) object and check the reference against the buttons.
- Finally, we can simply check each radio button to find out which one is currently selected by calling isSelected() on each button.

Swing Components (Cont...)

❑ Swing Buttons (Cont...): Radio Buttons (Cont...):

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
class RadioButton1 extends JFrame implements ActionListener
{
    JRadioButton r1, r2, r3;
    JButton b;
    RadioButton1()
    {
        setTitle("This is our Radio Button Example");
        r1 = new JRadioButton("Age is 18-22?");
        r1.setBounds(100, 50, 150, 30);
        r2 = new JRadioButton("Age is 23-30?");
        r2.setBounds(100, 100, 150, 30);
        r3 = new JRadioButton("Age is 31-40?");
        r3.setBounds(100, 150, 150, 30);
        ButtonGroup bg = new ButtonGroup();
        bg.add(r1); bg.add(r2); bg.add(r3);
        b = new JButton("Select");
        b.setBounds(100, 200, 100, 30);
        b.addActionListener(this);
        add(r1); add(r2); add(r3); add(b);
        setSize(350, 350);
        setLayout(null);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public void actionPerformed(ActionEvent e)
    {
        if(r1.isSelected())
        {
            JOptionPane.showMessageDialog(this, "You are in age group of 18-22.");
        }
        else if(r2.isSelected())
        {
            JOptionPane.showMessageDialog(this, "You are in age group of 23-30.");
        }
        else
        {
            JOptionPane.showMessageDialog(this, "You are in age group of 31-40.");
        }
    }
    public static void main(String args[])
    {
        new RadioButton1();
    }
}
```

Swing Components (Cont...)

❑ Swing Buttons (Cont...):

❖ Radio Buttons (Cont...):

Output

After Clicking “Select”.



Swing Components (Cont...)

❑ JTabbedPane:

- ❖ JTabbedPane encapsulates a *tabbed pane*.
- ❖ It manages a set of components by linking them with tabs.
- ❖ Selecting a tab causes the component associated with that tab to come to the forefront.
- ❖ JTabbedPane defines three constructors as given below:

`JTabbedPane()` //Creates an empty control with the tabs positioned at the top of the pane

`JTabbedPane(int tabPlacement)` //Creates an empty control with the tab placement

`JTabbedPane(int tabPlacement, int tabLayoutPolicy)` //Creates an empty control with a specified tab placement and tab layout policy

Swing Components (Cont...)

❑ JTabbedPane (Cont...):

- ❖ Tabs are added by calling `addTab()`. Here is one of its forms:

```
void addTab(String name, Component comp)
```

Here, *name* is the name of the tab and *comp* is the component that should be added to the tab.

- ❖ Often, the component added to a tab is a `JPanel` that contains a group of related components.
- ❖ There are three steps to use a tabbed pane:
 1. Create an instance of `JTabbedPane`.
 2. Add each tab by calling `addTab()`.
 3. Add the tabbed pane to the content pane.

Swing Components (Cont...)

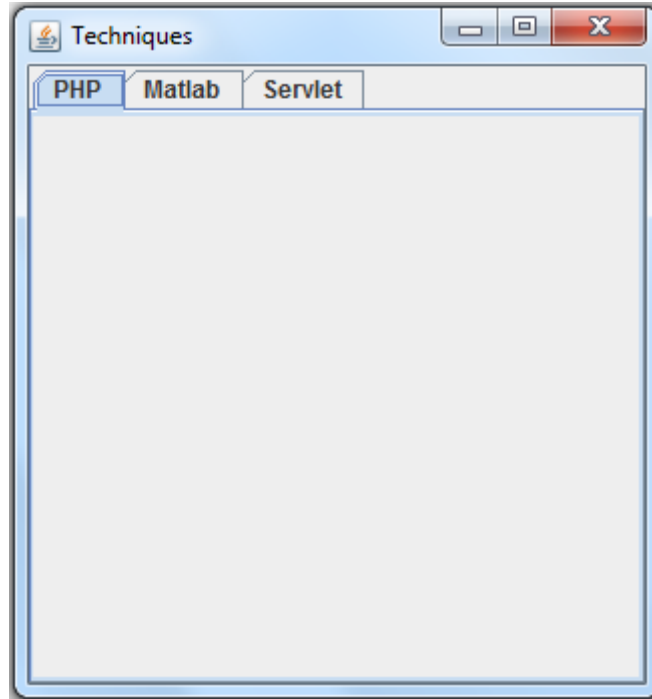
❑ JTabbedPane (Cont...):

```
import javax.swing.*;
import java.awt.*;
public class TabbedPane3
{
    public static void main(String args[])
    {
        JFrame f = new JFrame("Techniques");
        JTabbedPane tab = new JTabbedPane();
        JPanel panel1, panel2, panel3;
        panel1 = new JPanel();
        panel2 = new JPanel();
        panel3 = new JPanel();
        tab.addTab("PHP", panel1);
        tab.addTab("Matlab", panel2);
        tab.addTab("Servlet", panel3);
        f.add(tab);
        f.setSize(550,350);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Swing Components (Cont...)

❑ JTabbedPane (Cont...):

❖ Output



Swing Components (Cont...)

❑ JTabbedPane (Cont...):

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

public class TabbedPane2 extends JFrame
{
    public TabbedPane2()
    {
        setTitle("This is our Second TabbedPane Examples");
        JTabbedPane jtp = new JTabbedPane();
        jtp.addTab("Institute", new InstitutePanel());
        jtp.addTab("Branch", new BranchPanel());
        add(jtp);
        setSize(400, 400);
        setLayout(new FlowLayout());
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    class InstitutePanel extends JPanel
    {
        public InstitutePanel()
        {
            JButton b1= new JButton("NIT Patna");
            JButton b2= new JButton("NIT Delhi");
            add(b1); add(b2);
        }
    }

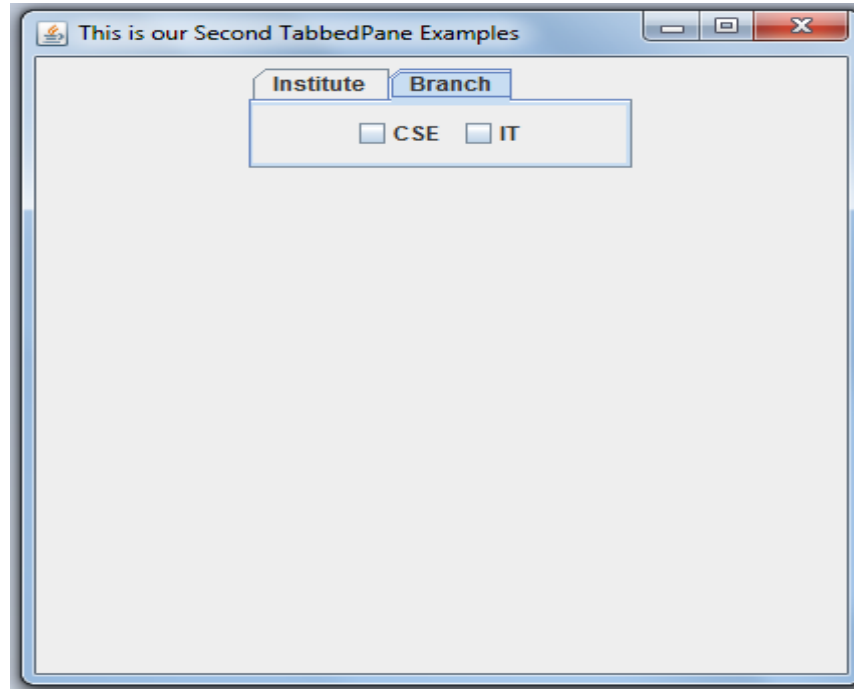
    class BranchPanel extends JPanel
    {
        public BranchPanel()
        {
            JCheckBox c1= new JCheckBox("CSE");
            JCheckBox c2= new JCheckBox("IT");
            add(c1); add(c2);
        }
    }

    public static void main(String[] args)
    {
        new TabbedPane2();
    }
}
```

Swing Components (Cont...)

❑ JTabbedPane (Cont...):

❖ Output



Swing Components (Cont...)

❑ JScrollPane:

- ❖ JScrollPane is a lightweight container that automatically handles the scrolling of another component.
- ❖ The component being scrolled can either be an individual component, such as a table or a group of components contained within another lightweight container, such as a JPanel.
- ❖ In either case, if the object being scrolled is larger than the viewable area, horizontal and/or vertical scroll bars are automatically provided and the component can be scrolled through the pane.

Swing Components (Cont...)

❑ JScrollPane (Cont...):

- ❖ The viewable area of a scroll pane is called the *viewport*. It is a window in which the component being scrolled is displayed.
- ❖ The scroll bars scroll the component through the viewport.
- ❖ In its default behavior, a JScrollPane dynamically add or remove a scroll bar as needed.
- ❖ When the component completely fits within the viewport, the scroll bars are removed.
- ❖ JScrollPane defines several constructors. One of them is given below:
`JScrollPane(Component comp)` //The component to be scrolled is specified by *comp*

Swing Components (Cont...)

❑ JScrollPane (Cont...):

❖ There are mainly three steps to use a scroll pane:

1. Create the component to be scrolled.
2. Create an instance of JScrollPane, passing to it the object to scroll.
3. Add the scroll pane to the content pane.

Swing Components (Cont...)

❑ JScrollPane (Cont...):

```
import java.awt.*;
import javax.swing.*;
public class ScrollPane1
{
    public ScrollPane1()
    {
        JFrame f = new JFrame("This is our Scroll Pane Example");
        JTextArea t = new JTextArea(10, 10);

        JScrollPane sp = new JScrollPane(t);
        sp.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        sp.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

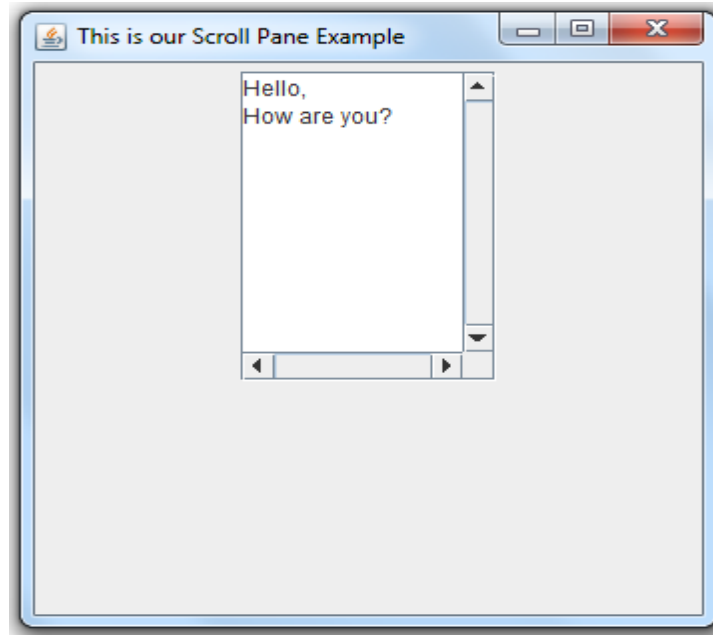
        f.getContentPane().add(sp);

        f.setSize(350, 350);
        f.getContentPane().setLayout(new FlowLayout());
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args)
    {
        new ScrollPane1();
    }
}
```

Swing Components (Cont...)

❑ JScrollPane (Cont...):

❖ Output



***Hello,

How are you?*** is typed manually.

Swing Components (Cont...)

❑ JList:

- ❖ This class supports the selection of one or more items from a list.
- ❖ Although the list often consists of strings, it is possible to create a list of just about any object that can be displayed.
- ❖ JList provides several constructors. Two are given below:

`JList()` `//Creates an empty JList`

`JList(Object[] items)` `//Creates a JList that contains the items in the array i.e. items.`

- ❖ JList is based on two models:

1. **ListModel:** This interface defines how access to the list data is achieved.
2. **ListSelectionModel:** This interface defines methods that determine what list item or items are selected.

Swing Components (Cont...)

❑ JList (Cont...):

- ❖ To support a large number of list, most of the time Jlist is wrapped inside a JScrollPane.
 - ❖ A JList generates a ListSelectionEvent, when a user makes or changes or deselects a selection.
 - ❖ It is handled by implementing ListSelectionListener as given below:
- `void valueChanged(ListSelectionEvent le)`
- ❖ By default, a JList allows the user to select multiple ranges of items within the list, but we can change this by calling `setSelectionMode()`:

`void setSelectionMode(int mode)`

Here, *mode* specifies the selection mode. It must be one of these values defined by ListSelectionModel: `SINGLE_SELECTION`, `SINGLE_INTERVAL_SELECTION` (user can select one range of items) and `MULTIPLE_INTERVAL_SELECTION`.

Swing Components (Cont...)

❑ JList (Cont...):

- ❖ We can obtain the index of the first selected item by `getSelectedIndex()`, which is also the index of the only selected item, when using single-selection mode. It is given below:

```
int getSelectedIndex( )
```

Here, indexing starts at zero. So, if the first item is selected, this method will return 0. If no item is selected, -1 is returned.

- ❖ We can obtain the value associated with the index by calling `getSelectedValue()`:

```
Object getSelectedValue( )
```

It returns a reference to the first selected value. If no value has been selected, it returns null.

Swing Components (Cont...)

❑ JList (Cont...):

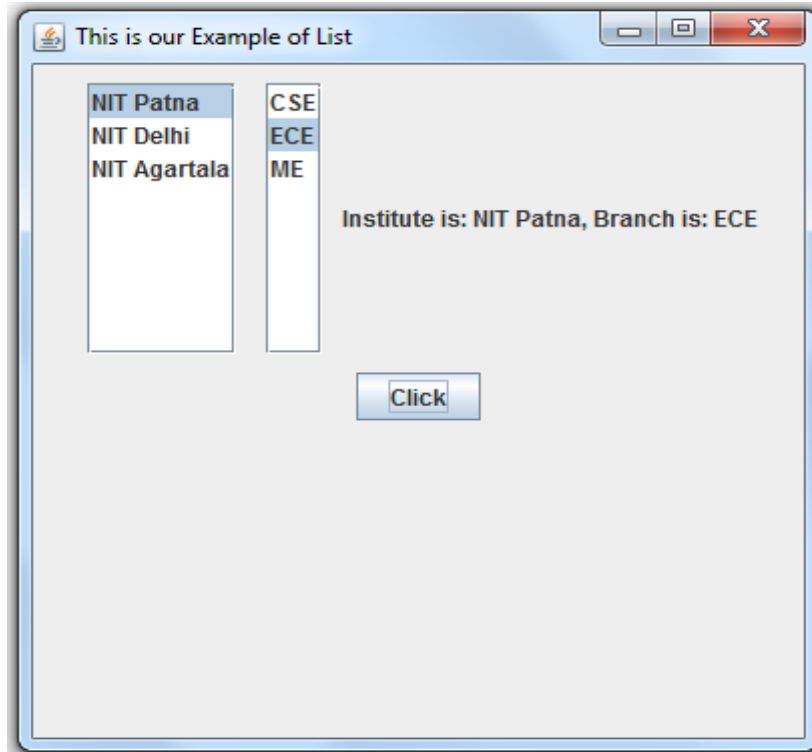
```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class List1
{
    JList lst1, lst2; JLabel l; JScrollPane sp1, sp2;
    String clg[]={"NIT Patna", "NIT Delhi", "NIT Agartala"};
    String branch[]={"CSE", "ECE", "ME"};
    public List1()
    {
        JFrame f= new JFrame("This is our Example of List");
        l = new JLabel("Hello, <--This will be changed after clicking");
        l.setBounds(300, 300, 200, 40);
        JButton b=new JButton("Click");
        b.setBounds(300, 150, 80, 30);
        JPanel jp1= new JPanel();
        JPanel jp2= new JPanel();
        lst1 = new JList(clg);
        lst2 = new JList(branch);
        sp1 = new JScrollPane(lst1);
        sp2 = new JScrollPane(lst2);
        jp1.add(sp1, BorderLayout.CENTER); jp2.add(sp2, BorderLayout.CENTER);
        f.add(jp1); f.add(jp2); f.add(l); f.add(b);
        f.setSize(400, 400);
        f.setLayout(new FlowLayout());
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        b.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                String s = " ";
                if (lst1.getSelectedIndex() != -1)
                {
                    s = "Institute is: " + lst1.getSelectedValue();
                    l.setText(s);
                }
                if (lst2.getSelectedIndex() != -1)
                {
                    s += ", Branch is: " + lst2.getSelectedValue();
                    l.setText(s);
                }
            }
        });
    }
    public static void main(String args[])
    {
        new List1();
    }
}
```

Swing Components (Cont...)

❑ JList (Cont...):

❖ Output



Swing Components (Cont...)

❑ JComboBox:

- ❖ Swing provides a *combo box* (a combination of a text field and a drop-down list) by using the JComboBox class.
- ❖ A combo box normally displays one entry, but it also displays a drop-down list that allows a user to select a different entry.
- ❖ The constructor (used in the example) of JComboBox is shown below:

`JComboBox(Object[] items)` //Here, *items* is an array that initializes the combo box

- ❖ JComboBox uses the ComboBoxModel. Mutable combo boxes (those whose entries can be changed) use the MutableComboBoxModel.
- ❖ We can dynamically add elements to the list of choices by using:
`void addItem(Object obj)` //Here, *obj* is the object to be added to the combo box
This method must be used only with mutable combo boxes.

Swing Components (Cont...)

❑ JComboBox (Cont...):

- ❖ JComboBox generates an action event, when the user selects an item from the list.
- ❖ JComboBox also generates an item event, when the state of selection changes that occurs when an item is selected or deselected.
- ❖ Mostly, in many cases, action events are used. However, both event types are available to use.
- ❖ One way to obtain the item selected in the list is to call `getSelectedItem()` on the combo box. It is given below:

`Object getItemSelected()`

Here, we need to cast the returned value into the type of object stored in the list

Swing Components (Cont...)

❑ JComboBox (Cont...):

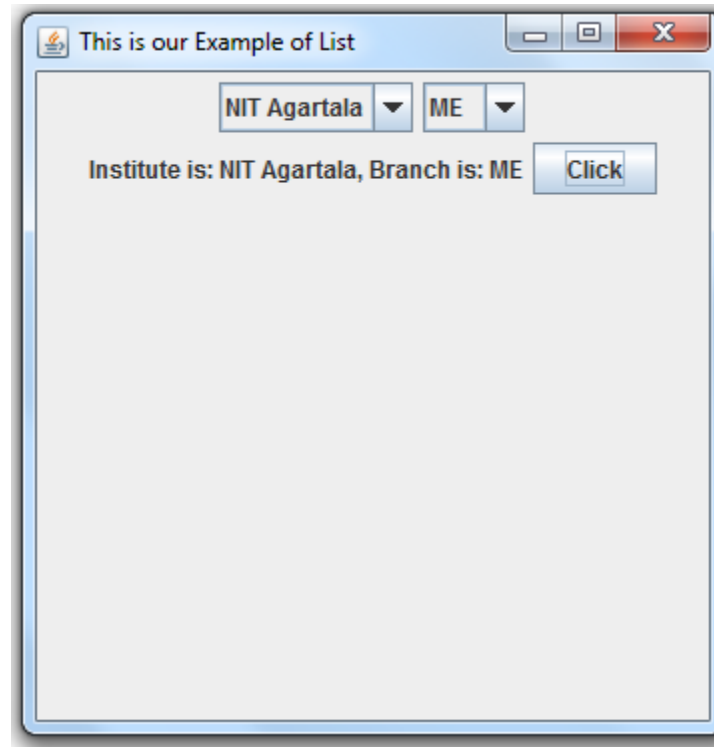
```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class ComboBox1
{
    JComboBox cb1, cb2; JLabel l;
    String clg[]={"NIT Patna", "NIT Delhi", "NIT Agartala"};
    String branch[]={"CSE", "ECE", "ME"};
    public ComboBox1()
    {
        JFrame f= new JFrame("This is our Example of List");
        l = new JLabel("Hello <--This will be changed after Clicking");
        //l.setBounds(300, 300, 200, 40);
        JButton b=new JButton("Click");
        //b.setBounds(300, 150, 80, 30);
        cb1 = new JComboBox(clg);
        cb2 = new JComboBox(branch);
        f.add(cb1); f.add(cb2); f.add(l); f.add(b);
        f.setSize(350, 350);
        f.setLayout(new FlowLayout());
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        b.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                String s = " ";
                if (cb1.getSelectedIndex() != -1)
                {
                    s = "Institute is: " + cb1.getItemAt(cb1.getSelectedIndex());
                    l.setText(s);
                }
                if(cb2.getSelectedIndex() != -1)
                {
                    s += ", Branch is: " + cb2.getItemAt(cb2.getSelectedIndex());
                    l.setText(s);
                }
            }
        });
    }
    public static void main(String args[])
    {
        new ComboBox1();
    }
}
```

Swing Components (Cont...)

❑ JComboBox (Cont...):

❖ Output



Swing Components (Cont...)

❑ **JMenuBar, JMenu and JMenuItem:**

- ❖ The JMenuBar class is used to display menubar on the window or frame. It may have several menus.
- ❖ The object of JMenu class is a pull down menu component, which is displayed from the menu bar. It inherits the JMenuItem class.
- ❖ The object of JMenuItem class adds a simple labeled menu item. The items used in a menu must belong to the JMenuItem or any of its subclass.

Swing Components (Cont...)

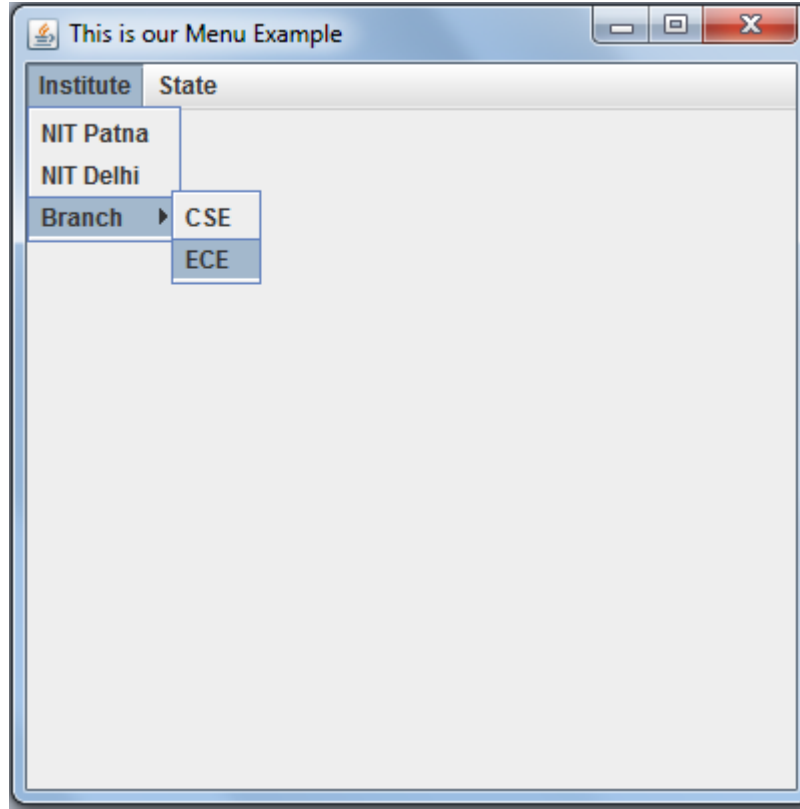
❑ JMenuBar, JMenu and JMenuItem (Cont...):

```
import java.awt.*;
import javax.swing.*;
public class Menu1
{
    JMenu Menu1, Menu2, Menu3;
    JMenuItem i1, i2, i3, i4, i5;
    public Menu1()
    {
        JFrame f= new JFrame("This is our Menu Example");
        JMenuBar mb=new JMenuBar();
        Menu1=new JMenu("Institute");
        Menu2=new JMenu("Branch");
        Menu3=new JMenu("State");
        i1=new JMenuItem("NIT Patna");
        i2=new JMenuItem("NIT Delhi");
        i3=new JMenuItem("CSE");
        i4=new JMenuItem("ECE");
        i5=new JMenuItem("Bihar");
        Menu1.add(i1); Menu1.add(i2); Menu1.add(Menu2);
        Menu2.add(i3); Menu2.add(i4);
        Menu3.add(i5);
        mb.add(Menu1);  mb.add(Menu3);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[])
    {
        new Menu1();
    }
}
```

Swing Components (Cont...)

❑ JMenuBar, JMenu and JMenuItem (Cont...):

❖ Output:



Swing Components (Cont...)

❑ Trees:

- ❖ A *tree* is a component that presents a hierarchical view of data.
- ❖ Trees are implemented in Swing by the `JTree` class. Constructors are:
`JTree(Object obj[])` //Constructed the tree from the elements in the array *obj*
`JTree(Vector<?> v)` //Constructs the tree from the elements of vector *v*
`JTree(TreeNode tn)` //Construct the tree, whose root node specified by *tn*
- ❖ Although `JTree` is packaged in `javax.swing`, its support classes and interfaces are packaged in `javax.swing.tree`.
- ❖ A `JTree` generates many events, but three relate specifically to trees: `TreeExpansionEvent`, `TreeSelectionEvent` and `TreeModelEvent`. A `TreeModelEvent` is fired, when data or structure of the tree changes.
- ❖ The listeners for these events are `TreeExpansionListener`, `TreeSelectionListene` and `TreeModelListener`, respectively.

Swing Components (Cont...)

❑ Trees (Cont...):

- ❖ `TreeSelectionListener` defines only one method, called `valueChanged()` that receives the `TreeSelectionEvent` object.
- ❖ We can obtain the path to the selected object by calling `getPath()` on the event object.

`TreePath getPath()`

It returns a `TreePath` object that describes the path to the changed node.

- ❖ `TreePath` class provides several constructors and methods. `toString()` method returns a string that describes the path.
- ❖ `TreeNode` interface defines methods to get information of a tree node.
- ❖ The `MutableTreeNode` interface extends `TreeNode` interface. It declares methods that can insert and remove child nodes or change the parent node.

Swing Components (Cont...)

❑ Trees (Cont...):

- ❖ The `DefaultMutableTreeNode` class implements the `MutableTreeNode` interface. It represents a node in a tree. One of its constructors is:

`DefaultMutableTreeNode(Object obj)`

Here, *obj* is the object to be enclosed in this tree node. The new tree node doesn't have a parent or children.

- ❖ To create a hierarchy of tree nodes, the `add()` method of `DefaultMutableTreeNode` can be used. Its signature is given below:

`void add(MutableTreeNode child)` //Here, *child* is a mutable tree node is to be added

- ❖ A `JTree` is typically placed within a `JScrollPane`.

- ❖ There are four steps to use a tree:

1. Create an instance of `JTree`.
2. Create a `JScrollPane` and specify the tree as the object to be scrolled.
3. Add the tree to the scroll pane.
4. Add the scroll pane to the content pane.

Swing Components (Cont...)

❑ Trees (Cont...):

```

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

public class Tree1
{
    JFrame f; JTree jt; JScrollPane sp; JLabel l;

    public Tree1()
    {
        f = new JFrame("It is our Tree Examples");
        DefaultMutableTreeNode Student=new DefaultMutableTreeNode("Student");
        DefaultMutableTreeNode College=new DefaultMutableTreeNode("College");
        DefaultMutableTreeNode NIT_Patna=new DefaultMutableTreeNode("NIT Patna");
        DefaultMutableTreeNode NIT_Delhi=new DefaultMutableTreeNode("NIT Delhi");
        Student.add(College);
        College.add(NIT_Patna);
        College.add(NIT_Delhi);
        DefaultMutableTreeNode Branch=new DefaultMutableTreeNode("Branch");
        DefaultMutableTreeNode CSE=new DefaultMutableTreeNode("CSE");
        DefaultMutableTreeNode ECE=new DefaultMutableTreeNode("ECE");
        Student.add(Branch);
        Branch.add(CSE);
        Branch.add(ECE);
        jt=new JTree(Student);
        sp=new JScrollPane(jt);
        JPanel jp= new JPanel();
        jp.add(sp);
        l=new JLabel();
        jp.add(l); f.add(jp);
        f.setSize(350, 350);
        f.setLayout(new FlowLayout());
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jt.addTreeSelectionListener(new TreeSelectionListener()
        {
            public void valueChanged(TreeSelectionEvent tse)
            {
                l.setText("We have selected: " + tse.getPath());
            }
        });
    }

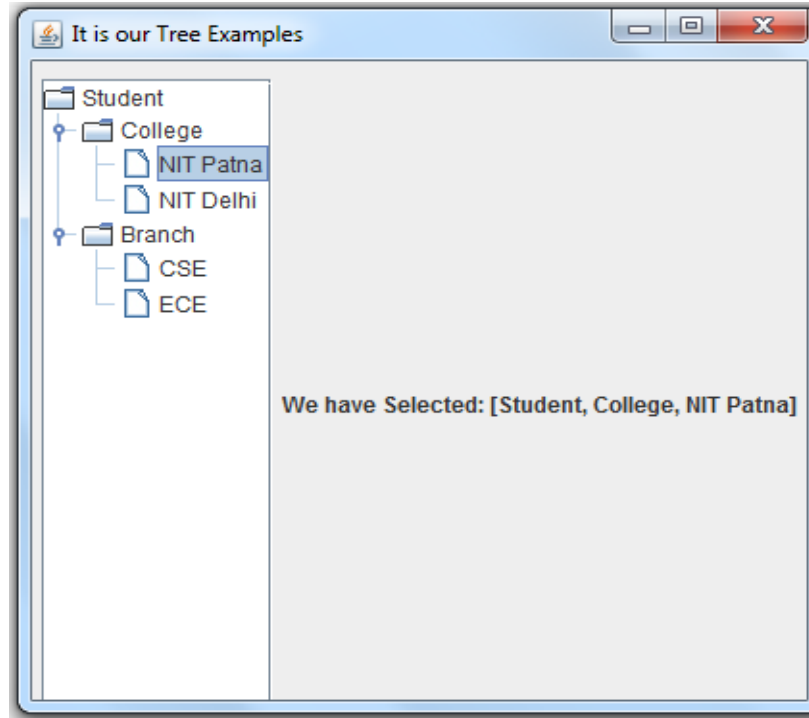
    public static void main(String[] args)
    {
        new Tree1();
    }
}

```

Swing Components (Cont...)

❑ Trees (Cont...):

❖ Output



Swing Components (Cont...)

❑ JTable:

- ❖ JTable is a component that displays rows and columns of data.
- ❖ We can drag the cursor on column boundaries to resize columns. We can also drag a column to a new position.
- ❖ Depending on its configuration, it is also possible to select a row, column or cell within the table and to change the data within a cell.
- ❖ Like JTree, JTable has many classes and interfaces associated with it. These are packaged in `javax.swing.table`.
- ❖ In addition to describing the data in a column, the heading also provides the mechanism by which the user can change the size of a column or change the location of a column within the table.
- ❖ We can normally wrap a JTable inside a JScrollPane.

Swing Components (Cont...)

❑ JTable (Cont...):

❖ JTable supplies several constructors. One of them is given below:

```
JTable(Object data[ ][ ], Object colHeads[ ])
```

Here, *data* is a two-dimensional array of the information to be presented, and *colHeads* is a one-dimensional array with the column headings.

❖ JTable relies on three models:

- 1. Table Model:** It is defined by the TableModel interface. It defines those things related to display data in a two-dimensional format.
- 2. Column Model:** It is represented by TableColumnModel. TableColumnModel specifies the characteristics of a column. Table model and column model are packaged in javax.swing.table.
- 3. Selection Model:** It determines how items are selected, and it is specified by the ListSelectionModel.

Swing Components (Cont...)

❑ JTable (Cont...):

- ❖ JTable can generate many different events. The two most fundamental to a table's operation are `ListSelectionEvent` and `TableModelEvent`.
- ❖ A `ListSelectionEvent` is generated, when the user selects something in the table. By default, JTable allows us to select one or more complete rows, but we can change this behavior.
- ❖ A `TableModelEvent` is generated, when the table's data changes in some way. Handling these events requires more work than above one.
- ❖ There are 4 steps to set up a JTable that can be used to display data:
 1. Create an instance of JTable.
 2. Create a `JScrollPane` object, specifying the table as the object to scroll.
 3. Add the table to the scroll pane.
 4. Add the scroll pane to the content pane.

Swing Components (Cont...)

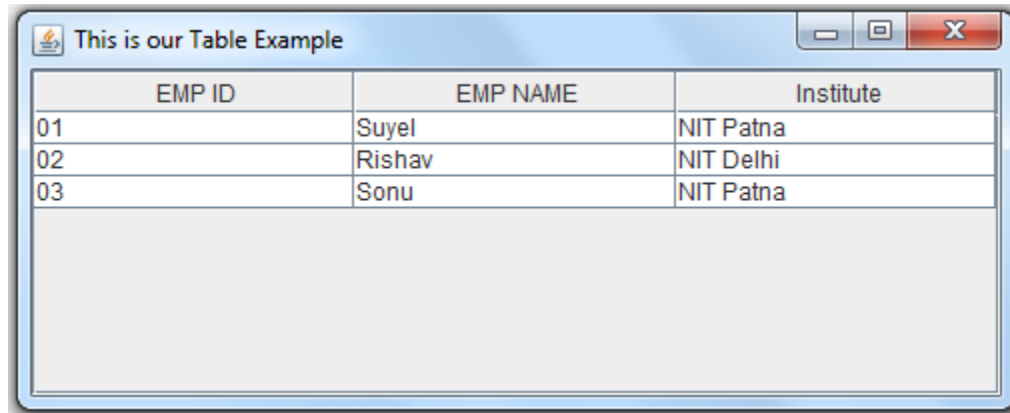
❑ JTable (Cont...):

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;
public class Table2
{
    public Table2()
    {
        JFrame f = new JFrame("This is our Table Example");
        String[] Column = {"EMP ID", "EMP NAME", "Institute"};
        String Data[][] = { {"01", "Suyel", "NIT Patna"}, {"02", "Rishav", "NIT Delhi"}, {"03", "Sonu", "NIT Patna"} };
        JTable t = new JTable(Data, Column);
        JScrollPane sp = new JScrollPane(t);
        f.add(sp);
        f.setSize(500, 200);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args)
    {
        new Table2();
    }
}
```


Swing Components (Cont...)

❑ JTable (Cont...):

❖ Output



EMP ID	EMP NAME	Institute
01	Suyel	NIT Patna
02	Rishav	NIT Delhi
03	Sonu	NIT Patna

Displaying Graphics

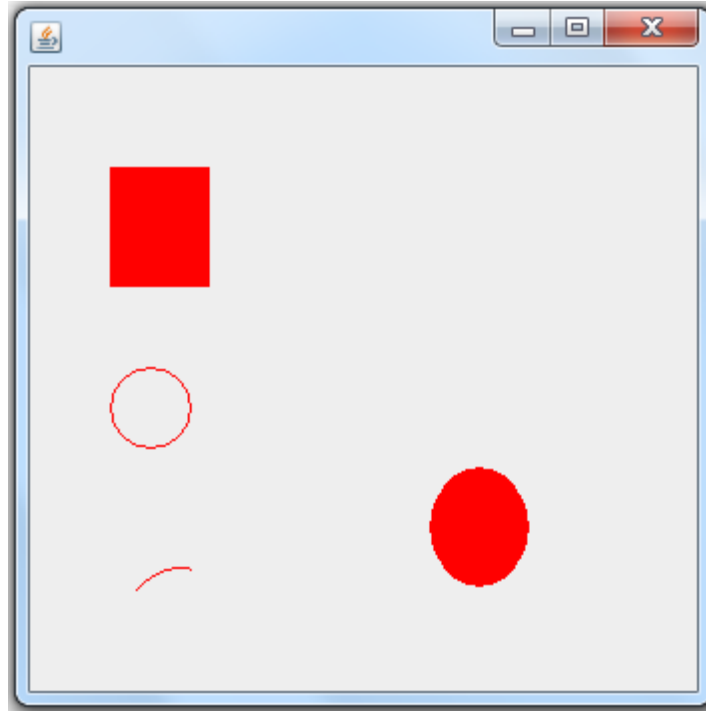
- ❑ There are several ways to create graphics in Java. the simplest way is to use `java.awt.Canvas` and `java.awt.Graphics`.
- ❑ A Canvas is a blank rectangular area of the screen onto which the application can be drawn.
- ❑ The Graphics class provides basic drawing methods such as `drawLine`, `drawRect` and `drawString`.

Displaying Graphics (Cont...)

```
import java.awt.*;
import javax.swing.JFrame;
public class Graphics1 extends Canvas
{
    public void paint(Graphics g)
    {
        g.fillRect(40, 50, 50, 60);
        g.drawOval(40, 150, 40, 40);
        setForeground(Color.RED);
        g.fillOval(200, 200, 50, 60);
        g.drawArc(40, 250, 70, 100, 80, 50);
    }
    public static void main(String[] args)
    {
        Graphics1 g=new Graphics1();
        JFrame f=new JFrame();
        f.add(g);
        f.setSize(350, 350);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Displaying Graphics (Cont...)

□ Output



Changing Icon of the TitleBar

- ❖ The `setIconImage()` method of `Frame` class is used to change the icon of a `Frame` or `Window`.
- ❖ It changes the icon, which is displayed at the left side of `Frame` or `Window`.
- ❖ `Toolkit` class is used to get instance of `Image` class in `AWT` and `Swing`.
- ❖ `Toolkit` class is the abstract super class of every implementation in the `Abstract Window Toolkit(AWT)`.

Changing Icon of the TitleBar (Cont...)

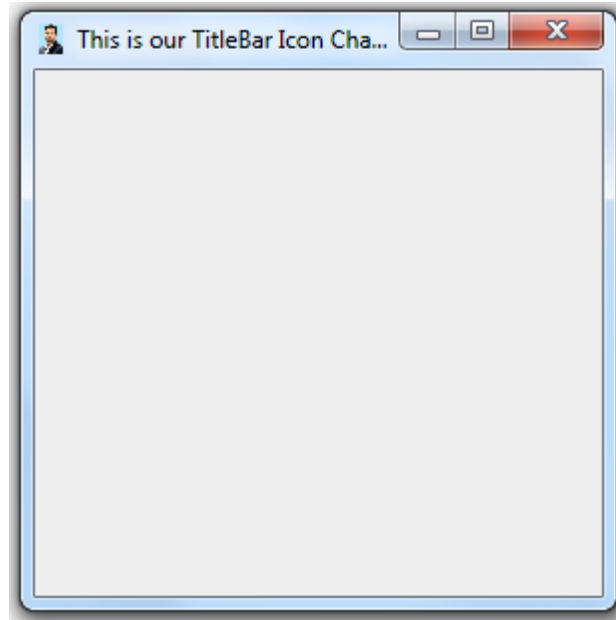
❑ Changing Icon of the TitleBar:

```
import java.awt.*;
import javax.swing.*;
class Label2
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("This is our TitleBar Icon Change Example");
        Image icon = Toolkit.getDefaultToolkit().getImage("C:/Users/SUYEL/Desktop/pass.jpg");
        f.setIconImage(icon);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Changing Icon of the TitleBar (Cont...)

❑ Changing Icon of the TitleBar:

❖ Output





**Slides are prepared from various sources,
such as Book, Internet Links and many
more.**