OBJECT ORIENTED PROGRAMMING WITH JAVA

BCA II SEM(NEP)

NOTES

Prepared By

Mr. Prabhu Kichadi, BE, MTECH

9880437187

UNIT – III

Event and GUI programming:

Event handling in java, Event types, Mouse and key events, GUI Basics, Panels, Frames, Layout Managers: Flow Layout Border Layout, Grid Layout, GUI components like Buttons, Check Boxes, Radio Buttons, Labels, TextFields, Text Areas, Combo Boxes, Lists, Windows, Menus.

Event and GUI programming:

GUI: GUI (Graphical User Interface) in Java refers to the process of creating visual interfaces for desktop applications using the Java programming language.

GUI Allows users to interact with the application through buttons, menus, text boxes, and other graphical components.

JFC: JFC is built on top of the Java AWT (Abstract Window Toolkit) and provides a more advanced and flexible set of components for creating modern and visually appealing GUI applications.

Swing: Swing is a modern and powerful GUI (Graphical User Interface) framework in Java that provides a wide range of components and tools for building desktop applications.

The package for swing components is **javax.swing**

AWT: AWT (Abstract Window Toolkit) is one of the oldest GUI (Graphical User Interface) frameworks in Java.

AWT provides a set of GUI components and tools for creating simple desktop applications in Java. It includes components such as buttons, labels, text fields, checkboxes, and others, and also provides layout managers for positioning components on the screen.

The package for AWT components is **java.awt**

Event: an event is an object that represents an action or occurrence that has happened within the GUI of an application or system.

Whenever an event occurs task/code associated with an event must be executed.
Events are generated when user interacts with the elements in a Graphical User Interface (GUI).

Eg:

1. Clicking on a Button,
2. Chooses a menu item,
3. Presses Enter in a text field,
4. Selecting A radio button,
5. Selecting OR deselecting a checkbox.

Event Sources: These are the objects of components on which event occurs. Source is responsible for providing information of the occurred event to its handler.

Eg: Button, JButton, MenuItem, JMenuItem, Scrollbar, List, Component, Window, etc.

Event Listeners/Handler: Event listeners are interfaces which is responsible to handle events. It is also known as event handler. Listener is responsible for generating response to an event.

Event listener interfaces are present in java.awt.event package.

Eg:
ActionListener ----> Handles Button, MenuItem, List related events (Action Events)
MouseListener ----> Handles events related to mouse movements.
KeyListener --- -> Handles events related to Text Components like Key Presses, Released, Entered etc.

If we want to write handler code, we must create an event handler class that must implements event listeners<<i>>, and provide definitions for abstract methods of listeners to corresponding events.

Ex:

DemoButton is an EvenetListener class for clicking a button event.

```
class DemoButton implements ActionListener
{
        public void actionPerfomed(ActionEvent e)
        {
                //write the code after event occurs,
                //write response code after an event occurs
        }
}
```

Containers: a container is a component that can contain and organize other components, such as buttons, labels, text fields, and others, into a hierarchical structure.

Containers are used in GUI (Graphical User Interface) programming to create the layout of the user interface, and they provide a way to manage the positioning and sizing of the components within them.

Ex: Frame, Panel, Window,

Components: components are the building blocks of the user interface. They are the visible elements that users interact with, such as buttons, labels, text fields, and other widgets. Java AWT provides a wide range of components that can be used to build user interfaces.

Here are some commonly used components(Classes) in Java AWT:

Button: A button is a component that can be clicked to perform an action.

Label: A label is a component that displays a text message or an image.

TextField: A text field is a component that allows the user to input a single line of text.

TextArea: A text area is a component that allows the user to input multiple lines of text.

Checkbox: A checkbox is a component that allows the user to select or deselect an option.

Choice: A choice is a component that allows the user to select an option from a list of predefined options.

List: A list is a component that displays a list of items that the user can select.

Scrollbar: A scrollbar is a component that allows the user to scroll through a large amount of content.

Panel: A panel is a container component that can hold other components.

Frame: A frame is a top-level container that represents the main window of the application.

Event handling in java:

Registering event listeners: The first step in event handling is to register event listeners for the components that generate the events. To register an event listener, you need to create an instance of the appropriate listener interface and add it to the component using the addXXXListener() method.

For example, to handle button clicks, you need to create an instance of the ActionListener interface and add it to the button using the addButtonListener() method.

Implementing event listener methods: Once you have registered an event listener, you need to implement the appropriate listener methods to handle the events. Each listener interface defines a set of methods that need to be implemented to handle the corresponding events.

For example, the ActionListener interface defines the actionPerformed() method, which is called when the user clicks a button.

Writing event handling code: When an event is generated, the appropriate listener method is called by the event handling system. You can write code in this method to respond to the event.

For example, if the user clicks a button, the actionPerformed() method is called, and you can write code in this method to perform the desired action.

General Steps to create a GUI, Register event and handle event.

1. Create a Frame (Container)
2. Set the Properties size, visibility etc.
3. Create Component elements Like Button, MenuItem, Checkbox, etc.
4. Set properties for components also.
5. Register component to Listeners. - addTypeListener()
6. Add components to Frame or Container. - add() method
7. Implement Listeners & override abstract methods (Handler code)
8. Execute the code.

Event types: In Java, events can be classified into different types based on the source of the event or the action that triggers the event.

Here are some of the common event types in Java:

Action events: Action events are generated when the user performs an action such as clicking a button or selecting an item from a menu. These events are typically handled using the ActionListener interface.

Mouse events: Mouse events are generated when the user interacts with the mouse, such as clicking, moving, or dragging the mouse. These events are typically handled using the MouseListener or MouseMotionListener interfaces.

Key events: Key events are generated when the user types a key on the keyboard. These events are typically handled using the KeyListener interface.

Window events: Window events are generated when a window is opened, closed, activated, deactivated, or resized. These events are typically handled using the WindowListener interface.

Focus events: Focus events are generated when a component gains or loses the focus. These events are typically handled using the FocusListener interface.

Component events: Component events are generated when a component is added, removed, or resized. These events are typically handled using the ComponentListener interface.

Container events: Container events are generated when a container component, such as a panel or frame, is added or removed from a parent container. These events are typically handled using the ContainerListener interface.

Mouse and key events:

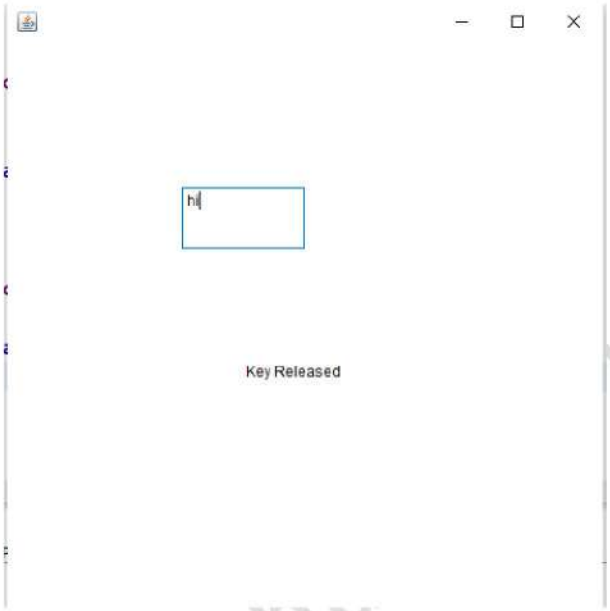
Key Events: Key events are generated when the user presses or releases a key on the keyboard. In Java, key events are typically handled using the **KeyListener** interface, which defines three methods: **keyPressed(), keyReleased(), and keyTyped().**

Ex:

```
package desk;
import java.awt.*;
import java.awt.event.*;
public class DemoKey implements KeyListener
{
	Frame fr;
	TextField tx;
	Label lab;

	public DemoKey()
	{
		fr = new Frame();
		fr.setBounds(100, 100, 500, 500);
		fr.setVisible(true);
		fr.setLayout(null);

		tx = new TextField();
		tx.setBounds(150, 150, 100, 50);

		lab = new Label();
		lab.setBounds(200, 200, 200, 200);
		fr.add(lab);

		fr.add(tx);
		tx.addKeyListener(this);
	}
	public static void main(String[] args)
	{
		DemoKey d = new DemoKey();
	}
	public void keyPressed(KeyEvent arg0)
	{
		lab.setText("Key Pressed");
	}
```

```
        public void keyReleased(KeyEvent arg0)
        {
                lab.setText("Key Released");
        }
        public void keyTyped(KeyEvent arg0)
        {
                lab.setText("Key Typed");
        }
}
```

Output:



Mouse Events: Mouse Events are used to handle user interaction with the mouse. There are several types of mouse events, including

Mouse Click, Mouse Enter, Mouse Exit, Mouse Press, Mouse Release, Mouse Move, and Mouse Drag.

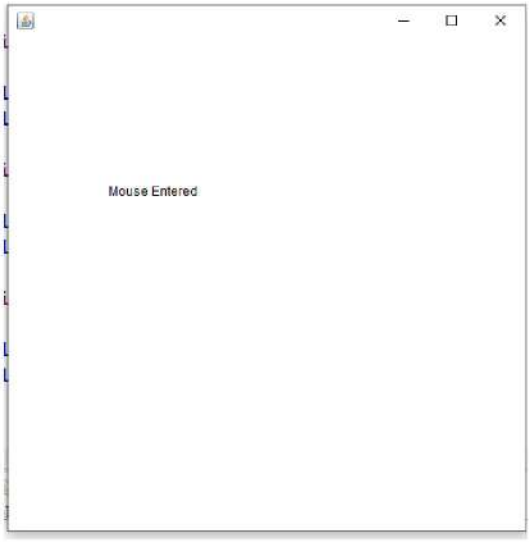
Ex:

```
package desk;

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

```
public class DemoMouse implements MouseListener
{
	Frame fr;
	Label lab;
	public DemoMouse()
	{
		fr = new Frame();
		fr.setBounds(100, 100, 500, 500);
		fr.setVisible(true);
		fr.setLayout(null);

		lab = new Label();

		fr.add(lab);

		fr.addMouseListener(this);
	}
	public static void main(String[] args)
	{
		new DemoMouse();
	}
	public void mouseClicked(MouseEvent arg0)
	{
		lab.setBounds(100, 100, 150, 100);
		lab.setText("Mouse Clicked");
	}

	public void mouseEntered(MouseEvent arg0)
	{
		lab.setBounds(100, 100, 150, 150);
		lab.setText("Mouse Entered");
	}
	public void mouseExited(MouseEvent arg0)
	{
		lab.setBounds(100, 100, 150, 180);
		lab.setText("Mouse Exited");
	}
	public void mousePressed(MouseEvent arg0)
	{
		lab.setBounds(100, 100, 150, 200);
		lab.setText("Mouse Pressed");
	}
	public void mouseReleased(MouseEvent arg0)
	{
		lab.setBounds(100, 100, 150, 220);
		lab.setText("Mouse Released");
```

```
        }
}
```

Output:



GUI Basics:

To create a GUI application in Java, you need to use the Swing API, which is a part of the Java Foundation Classes (JFC). The following are the basic steps involved in creating a GUI application in Java:

Create a Frame: A Frame is a top-level container that represents the main window of your application. To create a Frame, you can use the following code:
Frame frame = new Frame("My Application");

Add components to the Frame: Once you have created a Frame, you can add various components to it, such as buttons, labels, text fields, etc. To add a component to a Frame, you can use the add() method. For example:
Button button = new Button("Click me");
frame.add(button);

Set layout: You can use various layout managers to arrange the components in your Frame. The most commonly used layout managers are BorderLayout, FlowLayout, and GridLayout. For example:
frame.setLayout(new BorderLayout());

Show the Frame: Once you have added the components and set the layout, you can show the Frame by calling the setVisible() method:

```
frame.setVisible(true);

import javax.swing.*;

public class MyApplication
{
  public static void main(String[] args)
 {
    // Create a JFrame
    JFrame frame = new JFrame("My Application");

    // Create a JButton
    JButton button = new JButton("Click me");

    // Add the button to the JFrame
    frame.add(button);

    // Set the layout
    frame.setLayout(new BorderLayout());

    // Show the JFrame
    frame.setVisible(true);
  }
}
```

Panels: a Panel is a container that can hold and organize other components such as buttons, text fields, and labels. A Panel can be used to group related components together and to provide a more organized layout for the user interface of an application.

To create a Panel in Java, you can use the JPanel class which is a part of the Swing API. Here are the basic steps to create a JPanel:

Create a JPanel object: You can create a new JPanel object using the JPanel constructor.
JPanel panel = new JPanel();

Add components to the panel: Once you have created the panel, you can add components to it using the add() method. For example, to add a button to the panel:
JButton button = new JButton("Click me");
panel.add(button);

Set the layout: By default, a JPanel uses the FlowLayout layout manager, which arranges the components in a row. However, you can use other layout managers such as BorderLayout and GridLayout to arrange the components in a more complex manner. For example, to use a BorderLayout:
panel.setLayout(new BorderLayout());

Add the panel to a JFrame: Finally, you can add the panel to a JFrame or another container using the add() method. For example:

```
JFrame frame = new JFrame("My Application");
frame.add(panel);
```

Example:

```
import java.awt.*;
import javax.swing.*;
public class DemoPanel
{
  public static void main(String[] args) {
    // Create a JPanel
    JPanel panel = new JPanel();

    // Create a JButton
    JButton button = new JButton("Click me");
    button.setSize(100, 30);

    // Add the button to the panel
    panel.add(button);

    // Set the layout to BorderLayout
    panel.setLayout(new BorderLayout());

    // Create a JFrame
    JFrame frame = new JFrame("My Application");

    // Add the panel to the JFrame
    frame.add(panel);

    // Set the size of the JFrame
    frame.setSize(300, 200);

    // Show the JFrame
    frame.setVisible(true);
  }
}
```


Frames: In Java, a Frame is a top-level window that provides a container for building graphical user interfaces (GUIs). A Frame can be used to create the main window of an application or a dialog box. To create a Frame in Java, you can use the JFrame/Frame class, which is a part of the Swing API/AWT.

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

public class DemoMouse
{
  public static void main(String[] args)
    {
        // Create a JFrame
          JFrame frame = new JFrame("My Frame");

          // Set the size of the JFrame
          frame.setSize(400, 300);


          // Create aButton
          Button button = new Button("Click me");

          // Add the button to the JFrame
          frame.add(button);

          // Set the layout to FlowLayout
          frame.setLayout(new FlowLayout());

          // Show the JFrame
          frame.setVisible(true);
     }
}
```

Layout Managers: A Layout Manager is a class that is used to arrange the components of a graphical user interface (GUI) in a container. The Layout Manager controls the size and position of each component within the container.

1. Flow Layout:
2. Border Layout:
3. Grid Layout:

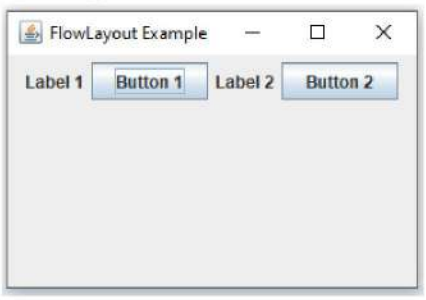
1. Flow Layout: FlowLayout is a Layout Manager that arranges components in a row, from left to right. When the row is full, the next component will be placed on the next row. It is commonly used for arranging buttons, labels, and other components in a simple manner.

To use FlowLayout in Java, you can create a JPanel object and set its Layout Manager to FlowLayout. Then, you can add components to the panel using the add() method.

Here is an example of using FlowLayout in Java:

```
import java.awt.*;
import javax.swing.*;
public class DemoFlow
{
  public static void main(String[] args)
  {
    JFrame frame = new JFrame("FlowLayout Example");
    // Create a JPanel with FlowLayout
    JPanel panel = new JPanel(new FlowLayout());
    // Create some components
    JLabel label1 = new JLabel("Label 1");
    JLabel label2 = new JLabel("Label 2");
    JButton button1 = new JButton("Button 1");
    JButton button2 = new JButton("Button 2");
    // Add the components to the panel
    panel.add(label1);
    panel.add(button1);
    panel.add(label2);
    panel.add(button2);
    // Add the panel to the JFrame
    frame.add(panel);
    // Set the size and show the JFrame
    frame.setSize(300, 200);
    frame.setVisible(true);
  }
}
```

Output:



2. Border Layout: BorderLayout is a Layout Manager that divides a container into five regions: north, south, east, west, and center. Each component added to the container is assigned a region based on a constraint that is specified when the component is added to the container.

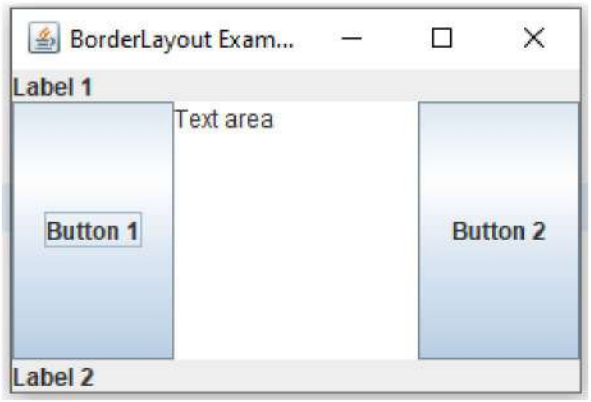
To use BorderLayout in Java, you can create a JPanel object and set its Layout Manager to BorderLayout. Then, you can add components to the panel using the add() method and a constraint.

Here is an example of using BorderLayout in Java:

```
package desk;

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

public class DemoBorder
{
 public static void main(String[] args)
 {
        // Create a JFrame
    JFrame frame = new JFrame("BorderLayout Example");

    // Create a JPanel with BorderLayout
    JPanel panel = new JPanel(new BorderLayout());

    // Create some components
    JLabel label1 = new JLabel("Label 1");
    JLabel label2 = new JLabel("Label 2");
    JButton button1 = new JButton("Button 1");
    JButton button2 = new JButton("Button 2");
    JTextArea textArea = new JTextArea("Text area");

    // Add the components to the panel
    panel.add(label1, BorderLayout.NORTH);
    panel.add(button1, BorderLayout.WEST);
    panel.add(button2, BorderLayout.EAST);
    panel.add(label2, BorderLayout.SOUTH);
    panel.add(textArea, BorderLayout.CENTER);

    // Add the panel to the JFrame
    frame.add(panel);

    // Set the size and show the JFrame
    frame.setSize(300, 200);
    frame.setVisible(true);
```

```
  }
}
```

Output:



3. Grid Layout: GridLayout is a Layout Manager that arranges components in a grid of rows and columns. All the components added to the container are arranged in a uniform size, and each cell in the grid can hold only one component.

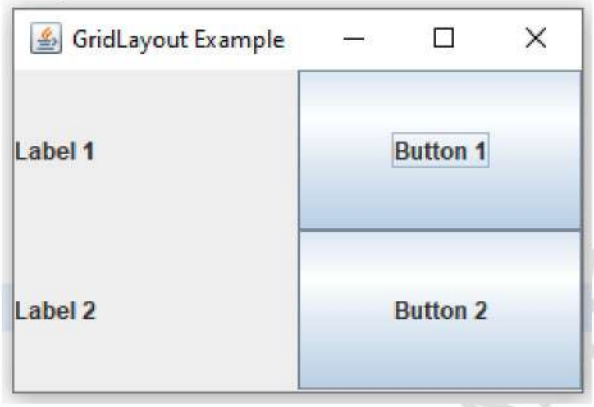
To use GridLayout in Java, you can create a JPanel object and set its Layout Manager to GridLayout. Then, you can add components to the panel using the add() method.

Here is an example of using GridLayout in Java:

```
package desk;

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

public class DemoGrid
{
  public static void main(String[] args)
  {
        // Create a JFrame
    JFrame frame = new JFrame("GridLayout Example");

    // Create a JPanel with GridLayout
    JPanel panel = new JPanel(new GridLayout(2, 2));

    // Create some components
    JLabel label1 = new JLabel("Label 1");
    JLabel label2 = new JLabel("Label 2");
```

```
    JButton button1 = new JButton("Button 1");
    JButton button2 = new JButton("Button 2");
    // Add the components to the panel
    panel.add(label1);
    panel.add(button1);
    panel.add(label2);
    panel.add(button2);
    // Add the panel to the JFrame
    frame.add(panel);
    // Set the size and show the JFrame
    frame.setSize(300, 200);
    frame.setVisible(true);
  }
}
```

Output:



GUI components: Each component has its own properties and methods that can be used to customize its appearance and behavior.

For example, a JButton is a GUI component that represents a clickable button. It has properties such as the button text, background color, font size, and border style, which can be set using its set methods. It also has methods to add and remove ActionListeners, which are invoked when the button is clicked.

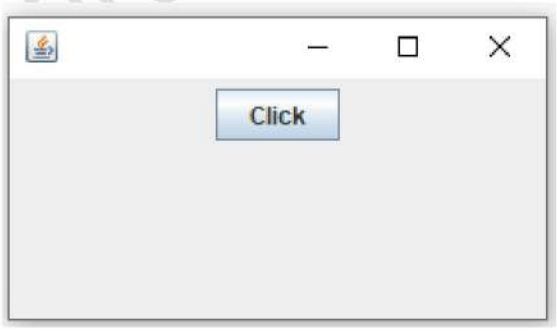
1. **Buttons:**
2. **Check Boxes:**
3. **Radio Buttons:**
4. **Labels:**
5. **TextFields:**
6. **Text Areas:**
7. **Combo Boxes:**
8. **Lists:**
9. **Windows:**
10. **Menus.**

1. Buttons: a button is a GUI component that can be clicked by the user to perform some action. Buttons can be created using the JButton class, which is part of the javax.swing package.

Here's an example of how to create a button in Java:

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

public class DemoButon
{

        JFrame fr;
        JPanel pan;
        JButton btn;

        public DemoButon()
        {
                fr = new JFrame();
                fr.setSize(400, 400);
                fr.setVisible(true);

                pan = new JPanel(new FlowLayout());
                fr.add(pan);

                btn = new JButton("Click");
                pan.add(btn);
        }
        public static void main(String[] args)
        {
                new DemoButon();
        }
}
```

Output:



2. Check Boxes: The JCheckBox component is used to represent a checkbox, which can be checked or unchecked. Here's an example of a JCheckBox:

```
package desk;
import java.awt.*;
import javax.swing.*;

public class DemoCheck
{

	JFrame fr;
	JPanel pan;
	JCheckBox c1,c2;

	public DemoCheck()
	{
		fr = new JFrame();
		fr.setSize(400, 400);
		fr.setVisible(true);

		pan = new JPanel(new FlowLayout());
		fr.add(pan);

		c1 = new JCheckBox("C");
		pan.add(c1);

		c2 = new JCheckBox("Java");
		pan.add(c2);
	}
	public static void main(String[] args)
	{
		new DemoCheck();
	}
}
```

3. Radio Buttons: JRadioButton is a Swing component in Java that allows the user to select only one option from a group of options. Here's an example of using JRadioButton in Java:

Ex:

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

public class DemoRadio
```

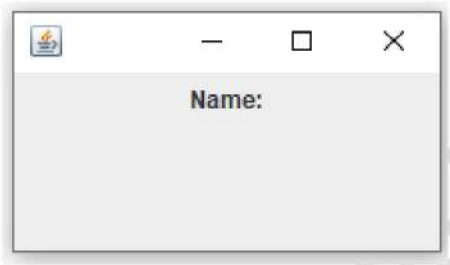
```
{

        JFrame fr;
        JPanel pan;
        JRadioButton r1,r2;
        ButtonGroup bg;

        public DemoRadio()
        {
                fr = new JFrame();
                fr.setSize(400, 400);
                fr.setVisible(true);

                pan = new JPanel(new FlowLayout());
                fr.add(pan);

                r1 = new JRadioButton("Male");
                r2 = new JRadioButton("FeMale");

                bg = new ButtonGroup();

                bg.add(r1);
                bg.add(r2);

                pan.add(r1);
                pan.add(r2);

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

4. Labels: Labels are a type of Swing component in Java that are used to display text or images in a graphical user interface (GUI). These static and non-clickable items. Here's an example of using JLabel in Java:

```
import java.awt.*;
import javax.swing.*;
public class DemoLabel
{
        JFrame fr;
        JPanel pan;
        JLabel lab;
```

```
        public DemoLabel()
        {
                fr = new JFrame();
                fr.setSize(400, 400);
                fr.setVisible(true);

                pan = new JPanel(new FlowLayout());
                fr.add(pan);

                lab = new JLabel("Name:");
                pan.add(lab);
        }

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

Output:

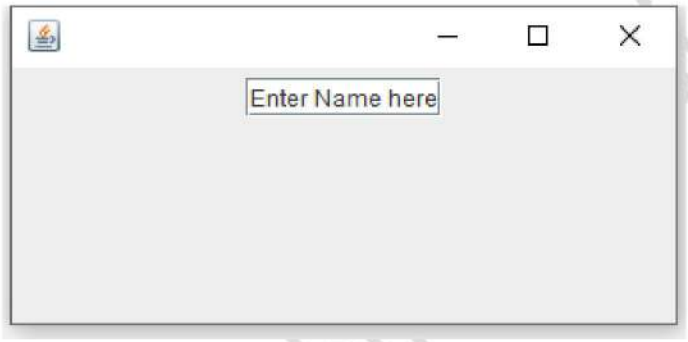


5. TextFields: TextFields are a type of Swing component in Java that allow the user to enter and edit text. Here's an example of using a JTextField in Java:

```
package desk;

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

public class DemoTextField
{
        JFrame fr;
        JPanel pan;
        JTextField tx;

        public DemoTextField()
        {
```

```
                fr = new JFrame();
                fr.setSize(400, 400);
                fr.setVisible(true);

                pan = new JPanel(new FlowLayout());
                fr.add(pan);

                tx = new JTextField("Enter Name here");
                pan.add(tx);
        }

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

}
```

Output:

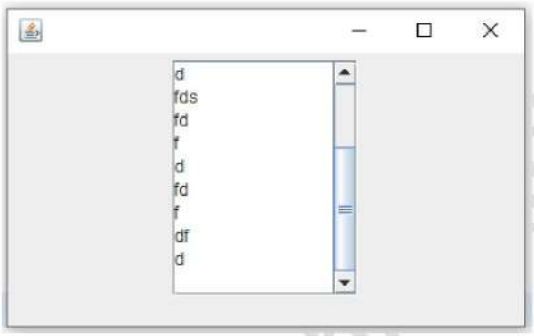


6. Text Areas: TextArea is a type of Swing component in Java that allows the user to enter and edit multiple lines of text. Here's an example of using a JTextArea in Java:

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

public class DemoTextArea
{
        JFrame fr;
        JPanel pan;
        JTextArea tx;
        JScrollPane jsp;

        public DemoTextArea()
        {
                fr = new JFrame();
```

```
                fr.setSize(400, 400);
                fr.setVisible(true);

                pan = new JPanel(new FlowLayout());
                fr.add(pan);

                tx = new JTextArea(10,10);
                pan.add(tx);

                jsp = new JScrollPane(tx);
                pan.add(jsp);
        }

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

Output:

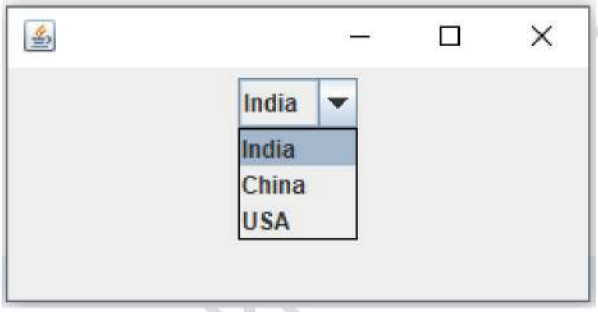


7. Combo Boxes: A JComboBox in Java is a Swing component that allows the user to choose one item from a drop-down list of items. Here's an example of using a JComboBox in Java:

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

public class DemoComboBox
{
        JFrame fr;
        JPanel pan;
        JComboBox jcb;

        public DemoComboBox()
```

```
        {
                fr = new JFrame();
                fr.setSize(400, 400);
                fr.setVisible(true);

                pan = new JPanel(new FlowLayout());
                fr.add(pan);

                String country[] = {"India", "China", "USA"};

                jcb = new JComboBox(country);

                pan.add(jcb);
        }

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

}
```

Output:

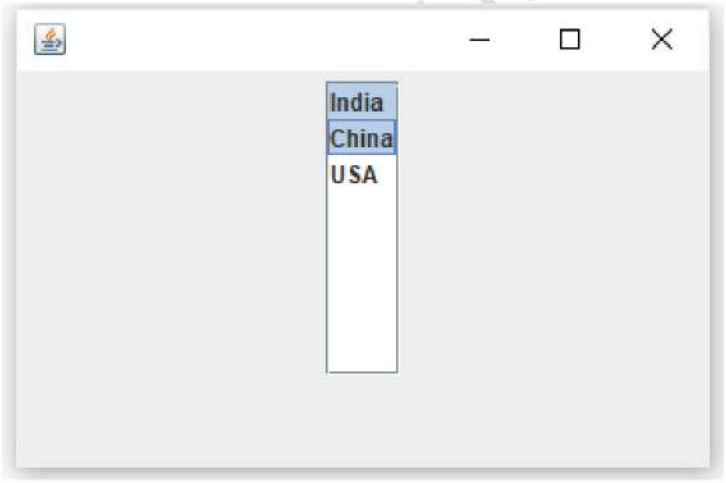


8. Lists: In Java, you can use the JList component to display a list of items in a GUI. We can select multiple items also.
Here's an example of how to use JList in Java:

```
import java.awt.*;
import javax.swing.*;
public class DemoList
{
        JFrame fr;
        JPanel pan;
        JList jl;
        JScrollPane jsp;
```

```
        public DemoList()
        {
                fr = new JFrame();
                fr.setSize(400, 400);
                fr.setVisible(true);

                pan = new JPanel(new FlowLayout());
                fr.add(pan);

                String country[] = {"India", "China", "USA"};

                jl = new JList(country);

                jsp = new JScrollPane(jl);

                pan.add(jsp);

        }

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

Output:



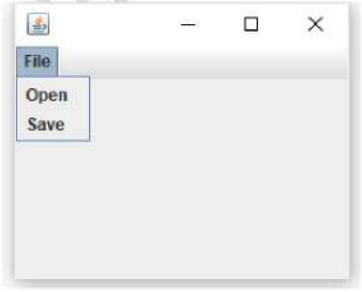
9. Windows:

10. Menus. In Java, you can create menus in your GUI application using the JMenuBar, JMenu, and JMenuItem classes.
Here's an example of how to create a simple menu bar with one menus and two menu items:

Ex:

```
import java.awt.*;
import javax.swing.*;
public class DemoMenu
{
        JFrame fr;
        JMenuBar bar;
        JMenu menu;
        JMenuItem i1,i2;
        public DemoMenu()
        {
                fr = new JFrame();
                fr.setSize(400, 400);
                fr.setVisible(true);
                bar = new JMenuBar();
                menu = new JMenu("File");
                i1 = new JMenuItem("Open");
                i2 = new JMenuItem("Save");
                menu.add(i1);
                menu.add(i2);
                bar.add(menu);
                fr.setJMenuBar(bar);
        }
        public static void main(String[] args)
        {
                new DemoMenu();
        }
}
```

Output:



Note: Refer all Java Lab journal programs on this unit.