

Unit 3

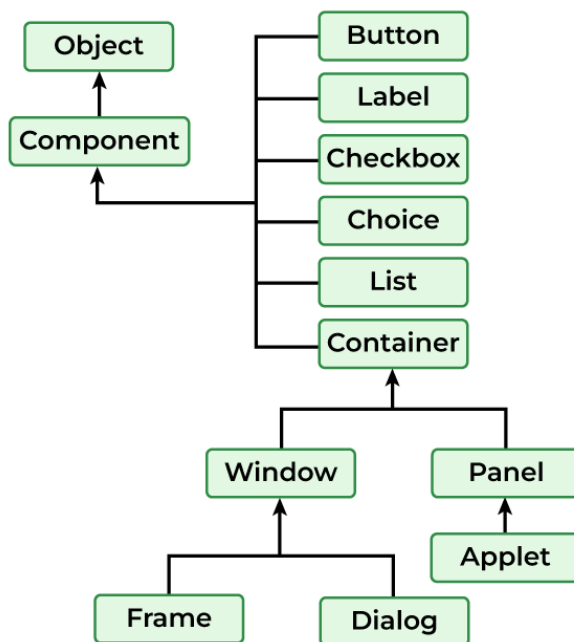
Building Components using Swing and JavaFX 6hrs

3.1 Introduction to AWT and Swing: Concept, Applets, Swing Class Hierarchy, Components/Containers

Introduction to AWT

AWT stands for Abstract Window Toolkit, which is an API for creating graphical user interface (GUI) or windows-based applications in Java. Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS). The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc. that can be used as objects in a Java program.

Java AWT Hierarchy



Components: AWT provides various components such as buttons, labels, text fields, checkboxes, etc used for creating GUI elements for Java Applications.

Containers: AWT provides containers like panels, frames, and dialogues to organize and group components in the Application.

Layout Managers: Layout Managers are responsible for arranging data in the containers some of the layout managers are BorderLayout, FlowLayout, GridLayout etc.

Event Handling: AWT allows the user to handle the events like mouse clicks, key presses, etc. using event listeners and adapters.

Graphics and Drawing: It is the feature of AWT that helps to draw shapes, insert images and write text in the components of a Java Application.

Java Applet

An applet is a program written in the Java programming language that can be included in an HTML page, much in the same way an image is included in a page. When you use a Java technology-enabled browser to view a page that contains an applet, the applet's code is transferred to your system and executed by the browser's Java Virtual Machine (JVM). For information and examples on how to include an applet in an HTML page, refer to this description of the <APPLET> tag.

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{

    public void paint(Graphics g){
        g.drawString("welcome",150,150);
    }
}
```

first.html

```
<html>
<body>
    <applet code="First" width="300" height="300"> </applet>
</body>
</html>
```

To Compile and Run an applet

Open Command Prompt

Compile Java File: javac First.java

to run: appletviewer first.html

[Note: Applets are not supported by java today]

Introduction to Swing

Swing in java is part of Java foundation class which is lightweight and platform independent GUI toolkit which has a wide variety of widgets for building optimized window-based applications. It is built on top of the AWT API and entirely written in java. It is platform independent unlike AWT and has lightweight components.

It becomes easier to build applications since we already have GUI swing components like swing like JButton, JComboBox, JList, JLabel etc. under the javax.swing package.

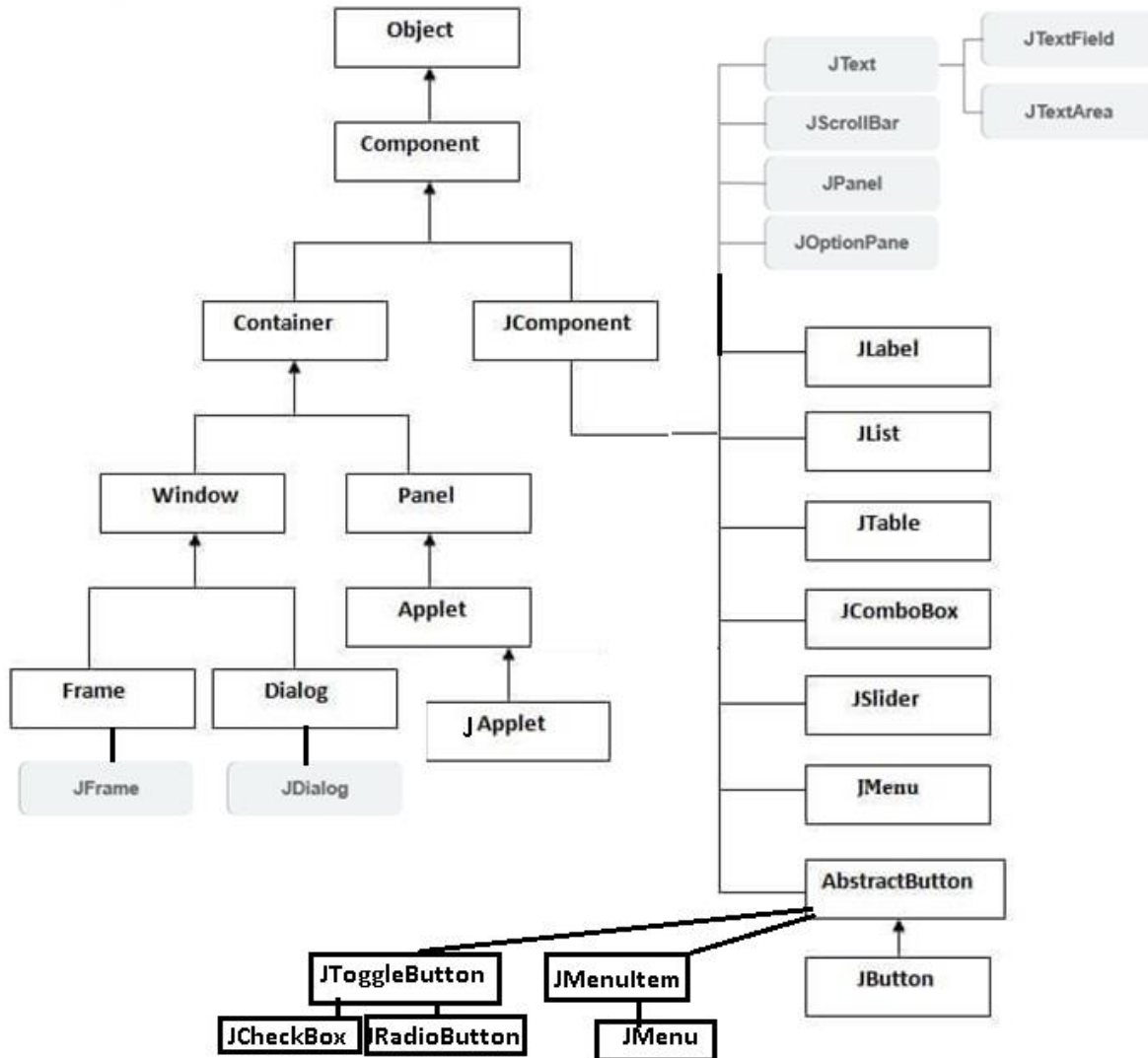
Container Class

Any class which has other components in it is called as a container class. For building GUI applications at least one container class is necessary.

Following are four types of container classes:

1. Window: Window is a top-level container that represents a graphical window.
2. Panel – It is used to organize components on to a window
3. Frame – A fully functioning window with icons and titles
4. Dialog – It is like a pop-up window but not fully functional like the frame

Java Swing Class Hierarchy



All the components in swing like JButton, JComboBox, JList, JLabel are inherited from the JComponent class which can be added to the container classes. Containers are the windows like frame and dialog boxes. Basic swing components are the building blocks of any GUI application. Methods like setLayout override the default layout in each container. Containers like JFrame and JDialog can only add a component to itself.

3.2 Layout Management

Layout managers arrange GUI components in a container for presentation purposes. We can use the layout managers for basic layout capabilities. All layout managers implement the interface LayoutManager (in package java.awt).

FlowLayout

This is the default layout manager. FlowLayout implements a simple layout style, which is similar to how words flow in a text editor. Components are laid out from the upper-left corner, left to right and top to bottom. When no more components fit on a line, the next one appears on the next line. A small space is left between each component, above and below, as well as left and right. Class FlowLayout allows GUI components to be left aligned, centered (the default) and right aligned.

Its constructors are:

FlowLayout(); It creates the default layout, which centers components and leaves five pixels of space between each component.

FlowLayout(int how); It specifies how each line is aligned. And its values are:

FlowLayout.LEFT

FlowLayout.CENTER

FlowLayout.RIGHT

FlowLayout(int how, int horz, int vert); It specifies the horizontal and vertical space left between components in horz and vert.

Simple example of flow layout

```
import java.awt.*;
import javax.swing.*;
public class FlowLayoutDemo{
    public static void main( String[] args ){
        JFrame fr1=new JFrame("FlowLayout Demo");
        FlowLayout layout = new FlowLayout();
        fr1.setLayout(layout);
        JLabel human = new JLabel( "Being Human" );
        fr1.add(human);
        fr1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE );
        fr1.setSize( 400, 300 );
        fr1.setVisible( true );
    }
}
```

BorderLayout

The BorderLayout class implements a common layout style for top-level windows. It has four narrow, fixed-width components as the edges and one large area in the center. The four sides are referred to as north, south, east and west. The middle area is called center.

The components placed in the NORTH and SOUTH regions extend horizontally to the sides of the container and are as tall as the components placed in those regions. The EAST and WEST regions expand vertically between the NORTH and SOUTH regions and are as wide as the components placed in those regions. The component placed in the CENTER region expands to fill all remaining space in the layout.

Its constructors are:

BorderLayout(); It creates a default border layout.

BorderLayout(int horz, int vert): It specifies the horizontal and vertical space left between components in horz and vert. The BorderLayout defines the following constants that specify the regions:

BorderLayout.CENTER
BorderLayout.SOUTH
BorderLayout.EAST
BorderLayout.WEST
BorderLayout.NORTH

While adding component to the applet the following form of add() method is used void add(Component compobj, Object region)

Example:

```
import java.awt.*;
import javax.swing.*;
public class BorderLayoutDemo extends JFrame{
    private JButton button1;
    private JButton button2;
    private JButton button3;
    private JButton button4;
    private JButton button5;

    private BorderLayout layout; // borderlayout object
    // set up GUI and event handling
    public BorderLayoutDemo() {
        super( "BorderLayout Demo" );
        layout = new BorderLayout( 5, 5 ); // 5 pixel gaps
        setLayout( layout ); // set frame layout
        button1 = new JButton( "North" );
        button2 = new JButton( "South" );
        button3 = new JButton( "East" );
        button4 = new JButton( "West" );
        button5 = new JButton( "Center" );
        add( button1, BorderLayout.NORTH ); // add button to north
        add( button2, BorderLayout.SOUTH ); // add button to SOUTH
        add( button3, BorderLayout.EAST ); // add button to east
        add( button4, BorderLayout.WEST ); // add button to west
        add( button5, BorderLayout.CENTER ); // add button to center
    }
    public static void main( String[] args ){
        BorderLayoutDemo borderLayoutFrame = new BorderLayoutDemo ();
        borderLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE
);
        borderLayoutFrame.setSize( 300, 200 ); // set frame size
        borderLayoutFrame.setVisible( true ); // display frame
    }
}
```

GridLayout

GridLayout lays out components in a two-dimensional grid. While instantiating a GridLayout, it define number of rows and columns. Every Component in a GridLayout has the same width and height. Components are added to a GridLayout starting at the top-left cell of the grid and proceeding left to right until the row is full. Then the process continues left to right on the next row of the grid, and so on.

The constructors supported by GridLayout are:

GridLayout(): It creates a single-column grid layout.

GridLayout(int numRows, int numcols): It creates a grid layout with the specified number of rows and columns.

GridLayout(int numRows, int numcols, int horz, int vert): It specifies the horizontal and vertical space left between components in horz and vert.

Example of gridlayout:

```
import java.awt.*;
import javax.swing.*;
public class GridLayoutFrame extends JFrame{
    private JButton button1;
    private JButton button2;
    private JButton button3;
    private JButton button4;
    private Container container; // frame container
    private GridLayout gLayout; // first gridlayout

    // no-argument constructor
    public GridLayoutFrame(){
        super( "GridLayout Demo" );
        gLayout = new GridLayout( 2, 2, 5, 5 ); // 2 by 2; gaps of 5

        container = getContentPane(); // get content pane
        setLayout( gLayout ); // set JFrame layout

        button1= new JButton("One");
        button2= new JButton("Two");
        button3= new JButton("Three");
        button4= new JButton("Four");
        add( button1 );
        add( button2 );
        add( button3 );
        add( button4 );
        add( button4 );
        add( button4 );
    }
}
```

```
    } // end GridLayoutFrame constructor
    public static void main( String[] args ){
        GridLayoutFrame gridLayoutFrame = new GridLayoutFrame();
        gridLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        gridLayoutFrame.setSize( 300, 200 ); // set frame size
        gridLayoutFrame.setVisible( true ); // display frame
    } // end main
} // end class GridLayoutFrame
```

CardLayout

The CardLayout layout manager is significantly different from the other layouts. Whereas the other layout managers attempt to display all the components within the container at once, a CardLayout displays only one component at a time.

The following call to `setLayout()` changes the `LayoutManager` of the current container to `CardLayout`:

```
lm = new CardLayout();
setLayout (lm);
```

Constructors

public CardLayout ()

This constructor creates a `CardLayout` using a horizontal and vertical gap of zero pixels. With `CardLayout`, there is no space between components because only one component is visible at a time; think of the gaps as insets.

public CardLayout (int hgap, int vgap)

This version of the constructor allows you to create a `CardLayout` with a horizontal gap of `hgap` and vertical gap of `vgap` to add some space around the outside of the component that is displayed. The units for gaps are pixels. Using negative gaps chops off components at the edges of the container.

GridBagLayout

The class `GridBagLayout` arranges the components in a horizontal and vertical manner.

GridBagLayout(): Creates a grid bag layout manager.

3.3 GUI Controls

GUI controls are built from GUI components. These are also called controls or widgets. A GUI component is an object with which the user interacts via the mouse, the keyboard or another form of input, such as voice recognition. Swing components are the basic building blocks of an application.

The Swing GUI components are defined in the javax.swing package. Swing has a wide range of various components, including buttons, check boxes, labels, panels, sliders, etc.

Swing Components

JLabel

The object of the JLabel class may be a component for putting text in a container. It's used to display one line of read-only text. The text is often changed by an application but a user cannot edit it directly. It inherits the JComponent class.

Syntax: JLabel jl = new JLabel();

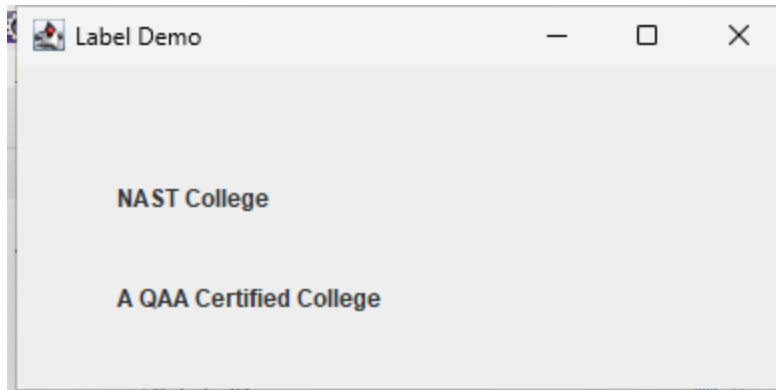
```
import javax.swing.JFrame;
import javax.swing.JLabel;
public class LabelDemo extends JFrame{
    //declare all the components here
    private JLabel lbl, lbl1;
    public LabelDemo() {
        //initialize all the components here
        lbl = new JLabel("NAST College");
        lbl.setBounds(50,50,200,30);

        lbl1 = new JLabel("A QAA Certified College");
        lbl1.setBounds(50,100,200,30);

        //add required controls here
        add(lbl);
        add(lbl1);

        setTitle("Label Demo");
        setLayout(null);
        setSize(400,200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String args[]) {
        new LabelDemo();
    }
}
```

Output:



JButton Class

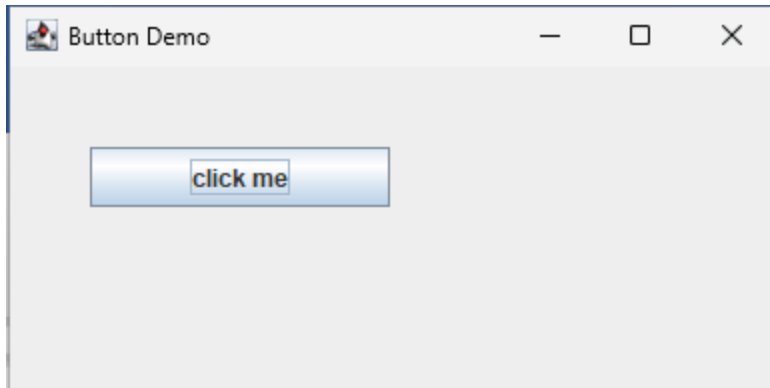
It is used to create a labelled button. Using the ActionListener it will result in some action when the button is pushed. It inherits the AbstractButton class and is platform independent.

```
import javax.swing.JButton;
import javax.swing.JFrame;
public class ButtonDemo extends JFrame{
    //declare all the components here
    private JButton btn;
    public ButtonDemo() {
        //initialize all the components here
        btn = new JButton("click me");
        btn.setBounds(40,40,150,30);

        //add required controls here
        add(btn);

        setTitle("Button Demo");
        setLayout(null);
        setSize(400,200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String args[]) {
        new ButtonDemo();
    }
}
```

Output:



JTextField Class

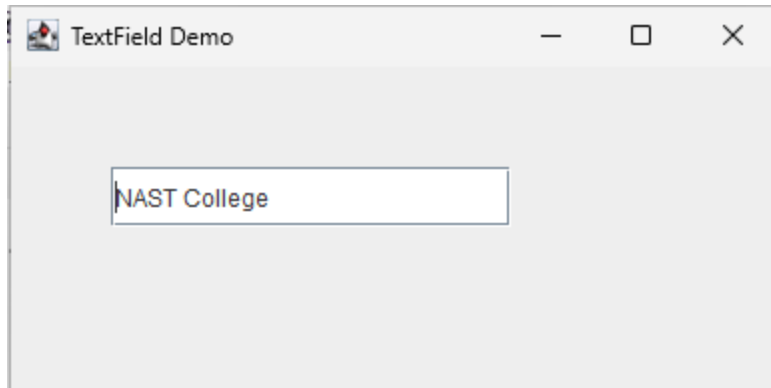
It inherits the JTextComponent class and it is used to allow editing of single line text.

```
import javax.swing.JFrame;
import javax.swing.JTextField;
public class TextFieldDemo extends JFrame{
    //declare all the components here
    private JTextField txt;
    public TextFieldDemo() {
        //initialize all the components here
        txt = new JTextField("NAST College");
        txt.setBounds(50,50,200,30);

        //add required controls here
        add(txt);

        setTitle("TextField Demo");
        setLayout(null);
        setSize(400,200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String args[]) {
        new TextFieldDemo();
    }
}
```

OUTPUT:



JTextArea class

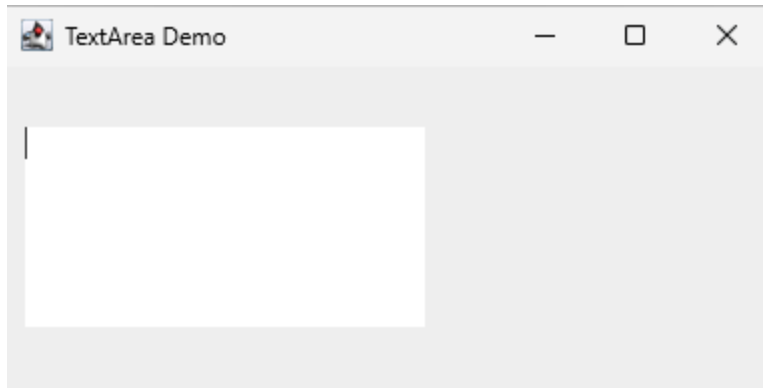
The object of a JTextArea class is a multi-line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class

```
import javax.swing.JFrame;
import javax.swing.JTextArea;
public class TextAreaDemo extends JFrame{
    //declare all the components here
    private JTextArea txt;
    public TextAreaDemo() {
        //initialize all the components here
        txt = new JTextArea(10,20);
        txt.setBounds(10,30, 200,100);
        txt.setAutoscrolls(true);

        //add required controls here
        add(txt);
        add(txt);

        setTitle("TextArea Demo");
        setLayout(null);
        setSize(400,200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String args[]) {
        new TextAreaDemo();
    }
}
```

OUTPUT:



JList Class

It inherits JComponent class, the object of JList class represents a list of text items.

```
import javax.swing.DefaultListModel;
import javax.swing.JFrame;
import javax.swing.JList;
public class ListDemo extends JFrame{
    //declare all the components here
    private JList<String> list;
    public ListDemo() {
        //initialize all the components here

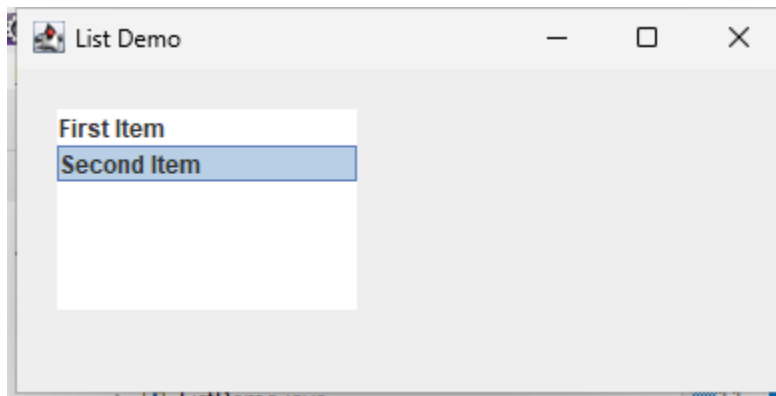
        DefaultListModel<String> items = new DefaultListModel<>();
        items.addElement("First Item");
        items.addElement("Second Item");

        list = new JList<String>(items);
        list.setBounds(20,20,150,100);

        //add required controls here
        add(list);

        setTitle("List Demo");
        setLayout(null);
        setSize(400,200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String args[]) {
        new ListDemo();
    }
}
```

Output:



JComboBox Class

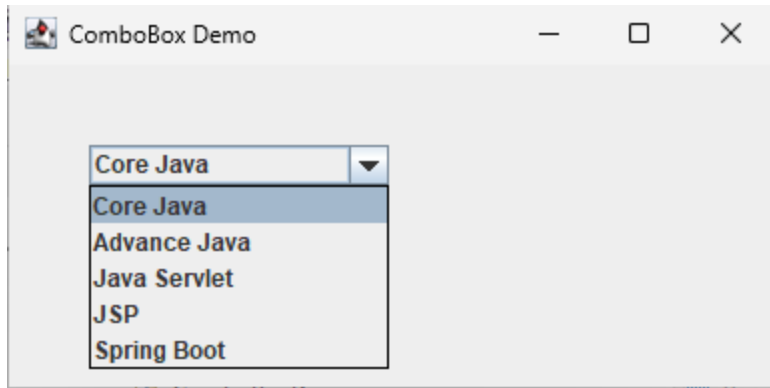
It inherits the JComponent class and is used to show pop up menu of choices.

```
import javax.swing.JComboBox;
import javax.swing.JFrame;
public class ComboBoxDemo extends JFrame{
    //declare all the components here
    private JComboBox<String> cbo;
    public ComboBoxDemo() {
        //initialize all the components here
        String courses[] = { "Core Java", "Advance Java", "Java Servlet", "JSP",
"Spring Boot"};
        cbo = new JComboBox<String>(courses);
        cbo.setBounds(40,40,150,20);

        //add required controls here
        add(cbo);

        setTitle("ComboBox Demo");
        setLayout(null);
        setSize(400,200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String args[]) {
        new ComboBoxDemo();
    }
}
```

Output:



JScrollBar Class

It is used to add scroll bar, both horizontal and vertical.

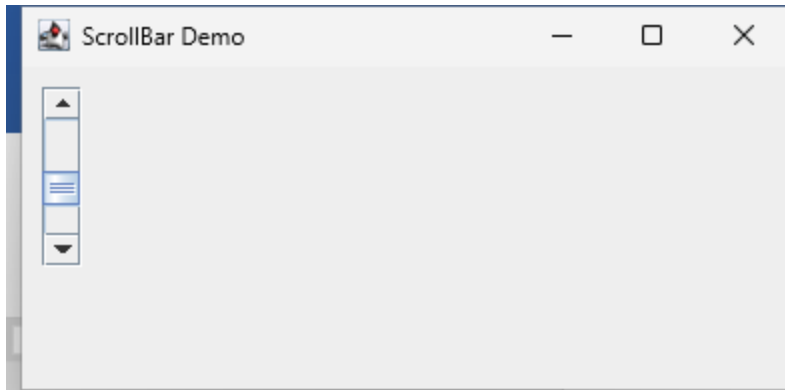
```
import javax.swing.JFrame;
import javax.swing.JScrollBar;
public class ScrollBarDemo extends JFrame{
    //declare all the components here
    private JScrollBar sb;
    public ScrollBarDemo() {
        //initialize all the components here

        sb = new JScrollBar();
        sb.setBounds(10,10,20,90);

        //add required controls here
        add(sb);

        setTitle("ScrollBar Demo");
        setLayout(null);
        setSize(400,200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String args[]) {
        new ScrollBarDemo();
    }
}
```

Output:



JPanel Class

It inherits the JComponent class and provides space for an application which can attach any other component.

```
import java.awt.Color;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollBar;
public class PanelDemo extends JFrame{
    //declare all the components here
    private JScrollBar sb;
    private JPanel panel;
    private JButton btn;
    public PanelDemo() {
        //initialize all the components here

        panel = new JPanel();
        panel.setBounds(10,10,200,100);
        panel.setBackground(Color.red);
        btn= new JButton("Click Me");
        btn.setBounds(60,50,80,40);

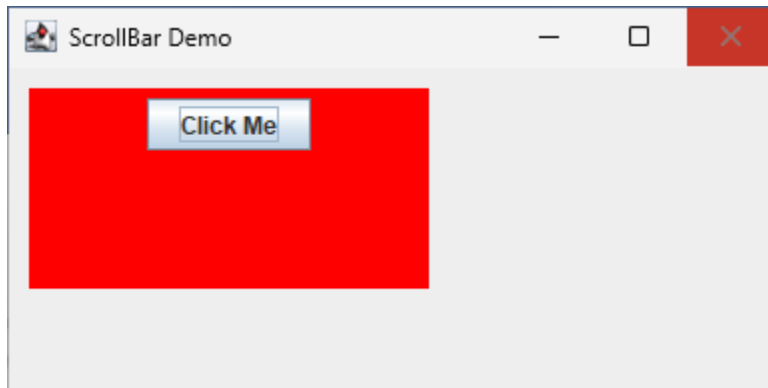
        //add required controls here
        panel.add(btn);
        add(panel);

        setTitle("Panel Demo");
        setLayout(null);
        setSize(400,200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
}
```



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

Output:

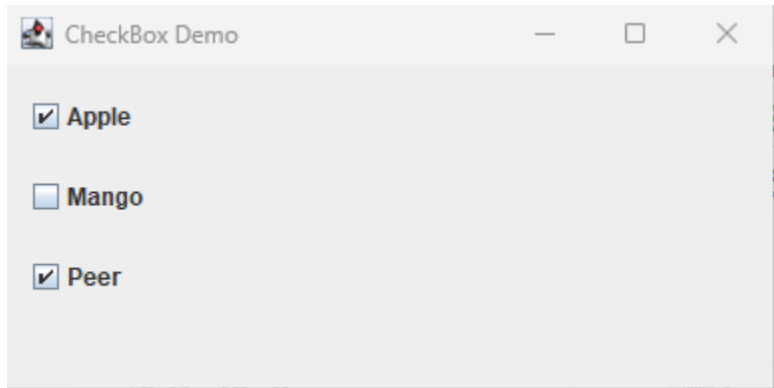


JCheckBox Class

The class JCheckBox is an implementation of a check box - an item that can be selected or deselected, and which displays its state to the user.

```
import javax.swing.JCheckBox;  
import javax.swing.JFrame;  
public class CheckBoxDemo extends JFrame{  
    //declare all the components here  
    private JCheckBox chkApple, chkMango,chkPeer;  
    public CheckBoxDemo() {  
        //initialize all the components here  
  
        chkApple = new JCheckBox("Apple");  
        chkApple.setBounds(10,10,100,30);  
        chkMango = new JCheckBox("Mango");  
        chkMango.setBounds(10,50,100,30);  
        chkPeer = new JCheckBox("Peer");  
        chkPeer.setBounds(10,90,100,30);  
  
        //add required controls here  
  
        add(chkApple);  
        add(chkMango);  
        add(chkPeer);  
  
        setTitle("CheckBox Demo");  
        setLayout(null);  
        setSize(400,200);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
}
```

```
        setVisible(true);
    }
    public static void main(String args[]) {
        new CheckBoxDemo();
    }
}
```



JRadioButton class

The class JRadioButton is an implementation of a radio button - an item that can be selected or deselected, and which displays its state to the user.

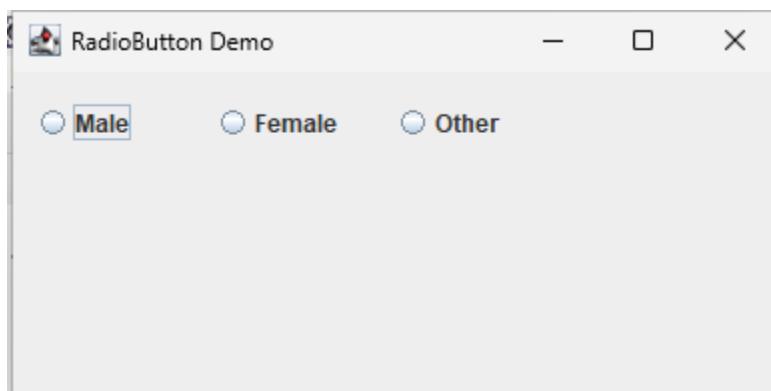
```
import javax.swing.ButtonGroup;
import javax.swing.JFrame;
import javax.swing.JRadioButton;
public class RadioButtonDemo extends JFrame{
    //declare all the components here
    private JRadioButton rdoMale, rdoFemale, rdoOther;
    private ButtonGroup gender;
    public RadioButtonDemo() {
        //initialize all the components here
        gender= new ButtonGroup();
        rdoMale = new JRadioButton("Male");
        rdoMale.setBounds(10,10,80,30);
        rdoFemale = new JRadioButton("Female");
        rdoFemale.setBounds(100,10,80,30);
        rdoOther = new JRadioButton("Other");
        rdoOther.setBounds(190,10,80,30);

        //add required controls here
        gender.add(rdoMale);
        gender.add(rdoFemale);
        gender.add(rdoOther);

        add(rdoMale);
        add(rdoFemale);
        add(rdoOther);
    }
}
```

```
setTitle("RadioButton Demo");
setLayout(null);
setSize(400,200);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setVisible(true);
}
public static void main(String args[]) {
    new RadioButtonDemo();
}
}
```

Output:



JPasswordField class

The class JPasswordField is a component which is specialized to handle password functionality and allows the editing of a single line of text.

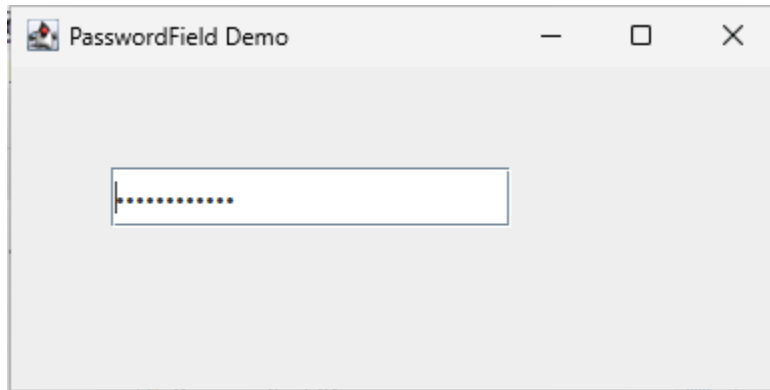
```
import javax.swing.JFrame;
import javax.swing.JPasswordField;
public class PasswordFieldDemo extends JFrame{
    //declare all the components here
    private JPasswordField txt;
    public PasswordFieldDemo() {
        //initialize all the components here
        txt = new JPasswordField("NAST College");
        txt.setBounds(50,50,200,30);

        //add required controls here
        add(txt);

        setTitle("PasswordField Demo");
        setLayout(null);
        setSize(400,200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
}
```

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

Output:



TabbedPane class

It is a pane that can contain tabs and each tab can display any component in the same pane. It is used to switch between a group of components by clicking on a tab with a given title or icon. To add the tabs to the JTabbedPane we can use the following methods:

tp.add(TabName, Components)

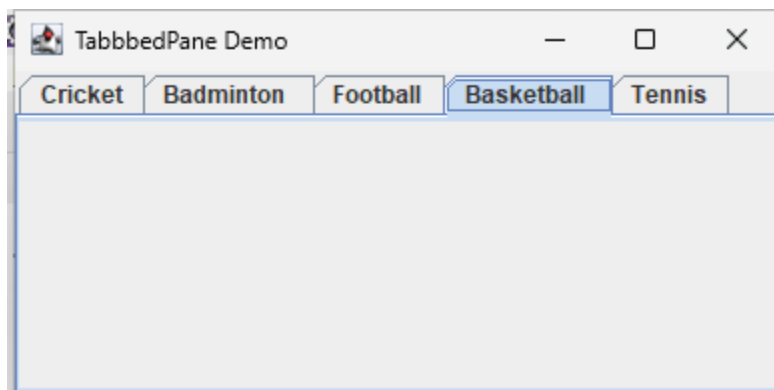
tp.addTab(TabName, Components)

```
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.JTabbedPane;  
public class TabbedPaneDemo extends JFrame{  
    //declare all the components here  
    private JTabbedPane tabbedPane;  
    private JPanel panel1, panel2, panel3, panel4, panel5;  
  
    public TabbedPaneDemo() {  
        //initialize all the components here  
        tabbedPane = new JTabbedPane ();  
        panel1 = new JPanel ();  
        panel2 = new JPanel ();  
        panel3 = new JPanel ();  
        panel4 = new JPanel ();  
        panel5 = new JPanel ();  
    }  
}
```

```
//add required controls here
tabbedPane.addTab ("Cricket", panel1);
tabbedPane.addTab ("Badminton ", panel2);
tabbedPane.addTab ("Football", panel3);
tabbedPane.addTab ("Basketball ", panel4);
tabbedPane.addTab ("Tennis", panel5);
add (tabbedPane);

setTitle("TabbedPane Demo");
//setLayout(null);
setSize(400,200);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setVisible(true);
}
public static void main(String args[]) {
    new TabbedPaneDemo();
}
}
```

Output:



JTable class

The JTable class is a part of Java Swing Package and is generally used to display or edit two-dimensional data that is having both rows and columns. It is similar to a spreadsheet. This arranges data in a tabular form.

Constructors in JTable:

JTable(): A table is created with empty cells.

JTable(int rows, int cols): Creates a table of size rows * cols.

JTable(Object[][] data, Object []Column): A table is created with the specified name where []Column defines the column names.

Functions in JTable:

addColumn(TableColumn []column) : adds a column at the end of the JTable.

clearSelection() : Selects all the selected rows and columns.

editCellAt(int row, int col) : edits the intersecting cell of the column number col and row number row programmatically, if the given indices are valid and the corresponding cell is editable.

setValueAt(Object value, int row, int col) : Sets the cell value as 'value' for the position row, col in the JTable.

```
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
public class TableDemo extends JFrame{
    //declare all the components here
    private JTable table;
    public TableDemo() {
        //initialize all the components here

        // Data to be displayed in the JTable
        String[][] empData = {
            { "Sunil Bahadur Bist", "1010", "BCA" },
            { "Ravi Khadka", "1009", "BE Computer" },
            { "Pusp Raj Joshi", "1014", "BE Computer" },
            { "Harendra Bikram Shah", "1013", "BE Computer" },
            { "Khem Raj Upadhyay", "1020", "BCA" },
            { "Prem Raj Joshi", "6014", "BBA" }
        };

        // Column Names
        String[] columnNames = { "Employee Name", "Code Number",
"Department" };

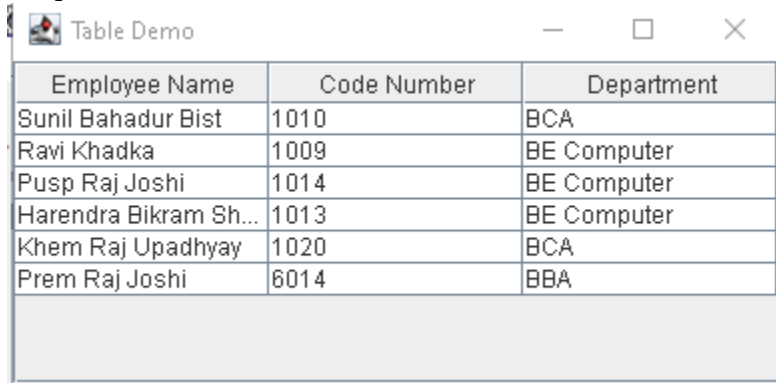
        // Initializing the JTable
        table = new JTable(empData, columnNames);
        table.setBounds(30, 40, 200, 300);

        // adding it to JScrollPane
        JScrollPane sp = new JScrollPane(table);

        //add required controls here
        add(sp);

        setTitle("Table Demo");
        //setLayout(null);
        setSize(400,200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public static void main(String args[]) {
        new TableDemo();
    }
}
```

Output:



The screenshot shows a Java Swing window titled "Table Demo". Inside the window is a table with three columns: "Employee Name", "Code Number", and "Department". The table contains six rows of data. Below the table is a large empty rectangular area.

Employee Name	Code Number	Department
Sunil Bahadur Bist	1010	BCA
Ravi Khadka	1009	BE Computer
Pusp Raj Joshi	1014	BE Computer
Harendra Bikram Sh...	1013	BE Computer
Khem Raj Upadhyay	1020	BCA
Prem Raj Joshi	6014	BBA

An GUI form to have multiple controls

```
import java.awt.FlowLayout;

import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
public class MultipleControlDemo extends JFrame{
    //declare all the components here
    private JPanel panel;
    private JTextField txt;
    private JButton btn;
    private JCheckBox chk;
    private JComboBox<String> combo;

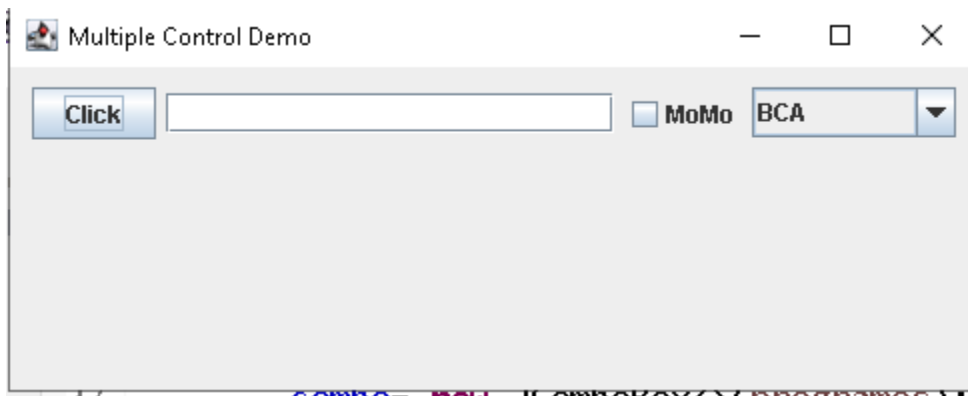
    public MultipleControlDemo() {
        //initialize all the components here

        panel=new JPanel();
        txt=new JTextField(20);
        btn=new JButton("Click");
        chk=new JCheckBox("MoMo");
        String[] programes={"BCA","BE Computer"};
        combo= new JComboBox<>(programes);

        //add required controls here
        panel.add(btn);
```

```
panel.add(txt);
panel.add(chk);
panel.add(combo);
add(panel);

setTitle("Multiple Control Demo");
setLayout(new FlowLayout());
setSize(500,200);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setVisible(true);
}
public static void main(String args[]) {
    new MultipleControlDemo();
}
}
```



3.4 Menu Elements and Tooltips

Menu Elements

The **JMenuBar** class is used to display menu bar 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.

Tooltips

You can create a tool tips for any JComponent with `setToolTipText()` method. This method is used to set up a tool tip for the component.

For example, to add tool tip to `PasswordField`, you need to add only one line of code:

```
field.setToolTipText("Enter your Password");
```

Simple Menu example with tooltips

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

public class MenuDemo extends JFrame implements ActionListener{
    private JMenuBar mb;
    private JMenu file, edit, view, help;
    private JMenuItem cut, copy, paste, selectAll;
    private JTextArea ta;
    public MenuDemo(){

        mb=new JMenuBar();
        file=new JMenu("File");
        file.setToolTipText("File Menu");
        edit=new JMenu("Edit");
        view=new JMenu("View");
        help=new JMenu("Help");

        cut=new JMenuItem("Cut");
        copy=new JMenuItem("Copy");
        paste=new JMenuItem("Paste");
        selectAll=new JMenuItem("Select All");

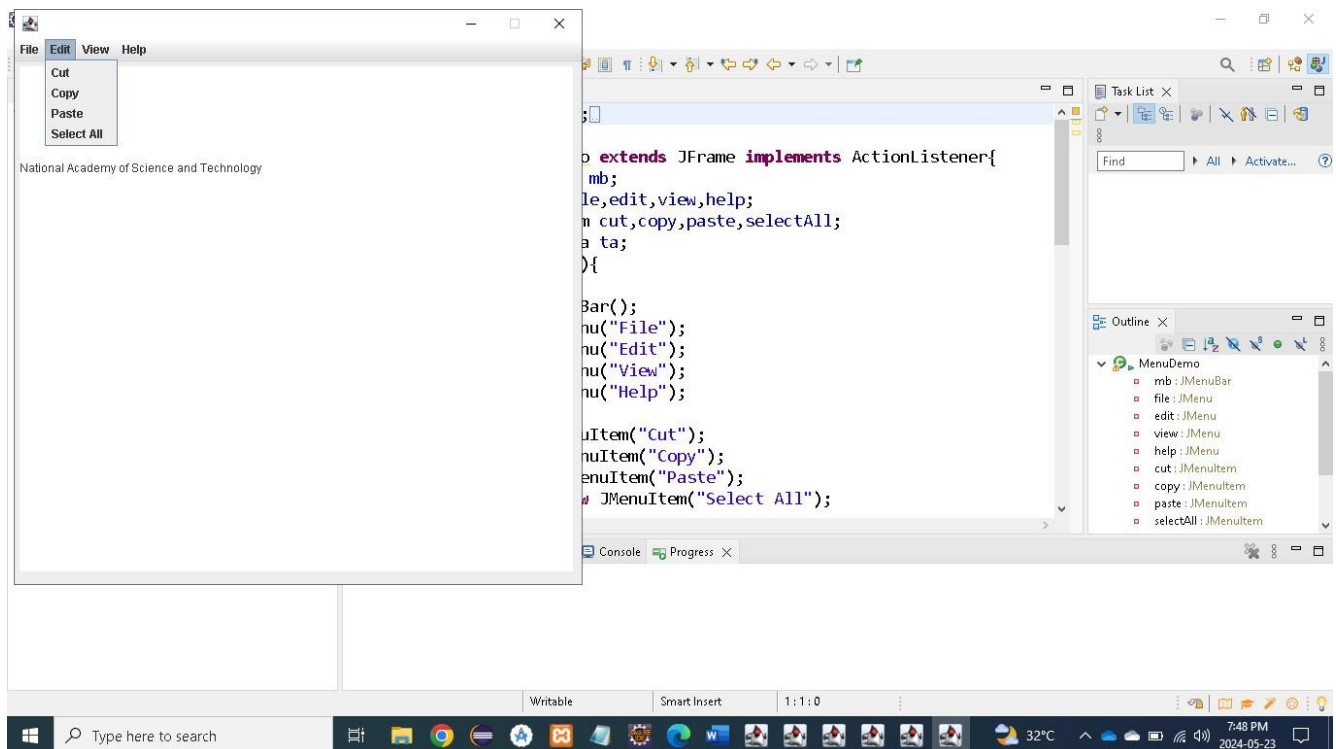
        edit.add(cut);
        edit.add(copy);
        edit.add(paste);
        edit.add(selectAll);

        mb.add(file);
        mb.add(edit);
        mb.add(view);
        mb.add(help);

        ta=new JTextArea();
        ta.setBounds(5,5,570,520);
        ta.setToolTipText("Text place where you write your text.");
        add(ta);

        cut.addActionListener(this);
        copy.addActionListener(this);
        paste.addActionListener(this);
        selectAll.addActionListener(this);
    }
}
```

```
mb.setToolTipText("Menu Bar");
setJMenuBar(mb);
setResizable(false);
setLayout(null);
setSize(600,600);
setVisible(true);
}
public void actionPerformed(ActionEvent e) {
    if(e.getSource()==cut)
        ta.cut();
    if(e.getSource()==paste)
        ta.paste();
    if(e.getSource()==copy)
        ta.copy();
    if(e.getSource()==selectAll)
        ta.selectAll();
}
public static void main(String[] args) {
    new MenuDemo();
}
}
```



3.5 Dialogs and Frames

Dialogs

JDialog is a part Java swing package. The main purpose of the dialog is to add components to it. JDialog can be customized according to user need.

Constructor of the class are:

- JDialog() : creates an empty dialog without any title or any specified owner
- JDialog(Frame o) : creates an empty dialog with a specified frame as its owner
- JDialog(Frame o, String s) : creates an empty dialog with a specified frame as its owner and a specified title
- JDialog(Window o) : creates an empty dialog with a specified window as its owner
- JDialog(Window o, String t) : creates an empty dialog with a specified window as its owner and specified title.
- JDialog(Dialog o) : creates an empty dialog with a specified dialog as its owner
- JDialog(Dialog o, String s) : creates an empty dialog with a specified dialog as its owner and specified title.

Useful Methods

- setLayout(LayoutManager m) : sets the layout of the dialog to specified layout manager
- setJMenuBar(JMenuBar m) : sets the menubar of the dialog to specified menubar
- add(Component c): adds component to the dialog
- setSize(w,h): sets the size of dialog
- isVisible(boolean b): sets the visibility of the dialog, if value of the boolean is true then visible else invisible

Simple Dialog Box example

```
import java.awt.event.*;
import javax.swing.*;
public class DialogDemo extends JFrame implements ActionListener {

    public DialogDemo() {
        JButton b = new JButton("Click Me");
        b.setBounds(20, 20, 100, 35);

        // add actionListener to button
        b.addActionListener(this);

        add(b);
        setLayout(null);
        setSize(200, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
}
```

```
public void actionPerformed(ActionEvent e)    {
    String s = e.getActionCommand();
    if (s.equals("Click Me")) {
        // create a dialog Box
        JDialog d = new JDialog(this, "NAST Dialog Box");

        // create a label
        JLabel l = new JLabel("Dialog Text for NAST");
        l.setBounds(10, 10, 200, 25);
        JTextField txt = new JTextField();
        txt.setBounds(10, 40, 200, 25);
        d.setLayout(null);
        d.add(txt);
        d.add(l);

        // setsize of dialog
        d.setSize(300, 300);

        // set visibility of dialog
        d.setVisible(true);
    }
}

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

Frames

The Java Frame and JFrame is an essential component of Java AWT and Swing, which is a part of the Java SWT (Standard Widget Toolkit). JFrame in Java is a class that allows you to create and manage a top-level window in a Java application. It serves as the main window for GUI-based Java applications and provides a platform-independent way to create graphical user interfaces. In Java JFrame is a part of javax.swing package.

Constructor of Java JFrame

- JFrame(): This is the default constructor for JFrame. It creates a new frame with no title
- JFrame(String title) This constructor creates a new frame with the specified title.
- JFrame(GraphicsConfiguration gc): This constructor creates a JFrame that uses the specified graphics configuration.
- JFrame(String title, GraphicsConfiguration gc): This constructor creates a JFrame with the specified title and using the specified graphics configuration.
- JFrame(Rectangle bounds): This constructor creates a JFrame with the specified bounds.

- JFrame(String title, Rectangle bounds): This constructor creates a JFrame with the specified title and bounds.

Some Useful Methods

- setTitle(String title): Sets the title of the JFrame.
- setSize(int width, int height): Sets the size of the JFrame.
- setDefaultCloseOperation(int operation): Sets the default close operation for the JFrame. Common options include JFrame.EXIT_ON_CLOSE, JFrame.HIDE_ON_CLOSE, and JFrame.DO_NOTHING_ON_CLOSE.
- setVisible(boolean b): Sets the visibility of the JFrame. Pass true to make it visible and false to hide it.
- setLayout(LayoutManager manager): Sets the layout manager for the JFrame, which controls how components are arranged within the frame.
- add(Component comp): Adds a Swing component to the JFrame.
- remove(Component comp): Removes a component from the JFrame.

Simple Swing Frame Example

```
import javax.swing.JFrame;
import javax.swing.JLabel;

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

        // Create a label
        JLabel label
        = new JLabel("NAST College Computer Engineering 4th Semester");

        // Add the label to the frame
        frame.add(label);

        // Set frame properties
        frame.setSize(400,200); // Set the size of the frame

        // Close operation
        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE);

        // Make the frame visible
        frame.setVisible(true);
    }
}
```

3.6 Event handling and Listener Interfaces

Event

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen. Events may also occur that are not directly caused by interactions with a user interface. For example, an event may be generated when a timer expires, software or hardware failure occurs, or an operation is completed. Most of the events relating to the GUI for a program are represented by classes defined in the package `java.awt.event`. This package also defines the listener interfaces for the various kinds of events that it defines. The package `javax.swing.event` defines classes for events that are specific to Swing components.

Types of Events (Event Categories)

The events can be broadly classified into two categories:

- **Foreground Events** - Those events which require the direct interaction of user are known as foreground events. They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.
- **Background Events** - Those events that do not require the interaction of end user are known as background events. For example: Operating system interrupts, hardware or software failure, if timer expires, an operation completion, etc.

The table below shows most of the important event classes and their description:

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or losses keyboard focus.
ItemEvent	Generated when a checkbox or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.
KeyEvent	Generated when input is received from the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.

The Event Delegation Model

The modern approach that is used to handle an event is delegation event model. It is a standard and consistent mechanism to generate and process event. It implements the following concept:

Source - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to its handler. A source generates an event and sends it to one or more listeners.

Listener (Event Handler) - It is also known as event handler. Listener is an object that is responsible for generating response to an event. Listener waits until it receives an event. Once the event is received, the listeners process the event and then returns.

The advantage of this design is that the application logic that processes events is clearly separated from the user interface logic that generates those events. This is an efficient way of handling the event because the event notifications are sent only to those listeners that want to receive them.

Event source

A source is an object that generates an event. This occurs when the internal state of that object changes. Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method.

The general form of registration method is:

```
public void addTypeListener(TypeListener el)
```

Here type is the name of event and el is the reference to the event listener. For example, the method that registers a keyboard event listener is called addKeyListener(KyeListener el). The method that register a mouse motion listener is called addMouseMotionListener(MouseMotionListener el). When an event occurs, all registered listeners are notified and receive a copy of the event object. This is known as multicasting the event.

The table below shows most of the important event sources and their description:

Event Sources	Description
Button	Generates action events when the button is pressed.
Checkbox	Generates item events when the checkbox is selected or deselected.
Choice	Generates item events when the choice is changed.
Menu Item	Generates action events when the menu is selected; generates item events when a checkable menu item is selected or deselected.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Scrollbar	Generates adjustment events when the scroll bar is manipulated.
Text Components	Generates text events when the user enters the character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

Event Listener Interfaces

Listeners are created by implementing one or more of the interfaces defined by the java.awt.event package. To react to an event, we implement appropriate event listener interfaces. A listener is an object that is notified when an event occurs. It has two requirements. First, it must have been registered with one or more sources to receive notifications about specific types of events. Second, it must implement methods to receive and process these notifications. We use `addTypeListener()` method to register the event listener with the source. Similarly, we can also unregister a listener by using `removeTypeListener()` method. The table below shows most of the important event listener and their description:

Interface	Description
ActionListener	Declares one method to receive action events. <code>void actionPerformed(ActionEvent e)</code>
AdjustmentListener	Declares one method when a scrollbar is manipulated. <code>void adjustmentValueChanged (AdjustmentEvent e)</code>
ComponentListener	Declares four methods to recognize when a component is hidden, moved, resized, or shown. <code>void componentResized(ComponentEvent e)</code> <code>void componentMoved(ComponentEvent e)</code> <code>void componentShown(ComponentEvent e)</code> <code>void componentHidden(ComponentEvent e)</code>
ContainerListener	Declares two methods to recognize when a component is added to or removed from a container. <code>void componentAdded(ContainerEvent e)</code> <code>void componentRemoved(ContainerEvent e)</code>
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus. <code>void focusGained(FocusEvent e)</code> <code>void focusLost(FocusEvent e)</code>
ItemListener	Defines one method to recognize when a check box or list item is clicked, when a choice selection is made, or when a checkable menu item is selected or deselected. <code>void itemStateChanged(ItemEvent e)</code>
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed. <code>void keyPressed(KeyEvent e)</code> <code>void keyReleased(KeyEvent e)</code> <code>void keyTyped(KeyEvent e)</code>

MouseListener	Defines five methods to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released. void mouseClicked(MouseEvent e) void mouseEntered(MouseEvent e) void mouseExited(MouseEvent e) void mousePressed(MouseEvent e) void mouseReleased(MouseEvent e)
MouseMotionListener	Defines two methods to recognize when the mouse is dragged or moved. void mouseDragged(MouseEvent e) void mouseMoved(MouseEvent e)
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved. void mouseWheelMoved(MouseWheelEvent e)
TextListener	Defines one method to recognize when a text value changes. void textChanged(TextEvent e)
WindowFocusListener	Defines two methods to recognize when a window gains or loses input focus. void windowGainedFocus(WindowEvent e) void windowLostFocus(WindowEvent e)
WindowListener	Defines seven methods to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit. void windowActivated(WindowEvent e) void windowDeactivated(WindowEvent e) void windowClosed(WindowEvent e) void windowClosing(WindowEvent e) void windowOpened(WindowEvent e) void windowIconified(WindowEvent e) void windowDeiconified(WindowEvent e)

3.7 Handling Action Events

Example of event handling

Example1:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
public class EventExample extends JFrame implements ActionListener{
    public static void main(String args[]) {
        JFrame f1=new JFrame("Event handling");
        JButton b1=new JButton("Click me");
```

```
f1.add(b1);
b1.addActionListener(new EventExample());
f1.setVisible(true);
f1.setSize(400,450);
}
public void actionPerformed(ActionEvent e) {
    JOptionPane.showMessageDialog(null, "The button is clicked");
}
}
```

Example 2:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
public class EventExample extends JFrame{
    public static void main(String args[]) {
        JFrame f1=new JFrame("Event handling");
        JButton b1=new JButton("Click me");
        f1.add(b1);
        f1.setVisible(true);
        f1.setSize(400,450);
        b1.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                String firstNumber=
                JOptionPane.showInputDialog( "Enter first integer" );
                String secondNumber=
                JOptionPane.showInputDialog( "Enter second integer"
                );
                // convert String inputs to int values for use in a
                calculation
                int number1 = Integer.parseInt( firstNumber );
                int number2 = Integer.parseInt( secondNumber );
                int sum = number1 + number2; // add numbers
                // display result in a JOptionPane message dialog
                JOptionPane.showMessageDialog( null, "The sum is " +
sum,
                "Sum of Two Integers", JOptionPane.PLAIN_MESSAGE );
            }
        });
    }
}
```

3.8 JavaFX vs Swing

JavaFX

JavaFX is a modern UI toolkit for Java applications which is designed to replace Swing as the standard GUI library. A rich set is offered by JavaFX for creating cutting-edge, visually attractive user interfaces (UI). Unlike Swing, JavaFX is built entirely in Java and offers extensive support for modern UI elements, multimedia, 2D and 3D graphics, and animation. JavaFX is a flexible option for cross-platform development because its programs may be run on PCs, mobile devices, and browsers. JavaFX has generally become the preferred choice for many Java developers when creating next-generation program due of its focus on rich user experiences and innovative design principles.

Key Differences Between Java Swing and Java FX

Swing is the standard toolkit for Java developer in creating GUI, whereas JavaFX provides platform support for creating desktop applications.

Swing has a more sophisticated set of GUI components, whereas JavaFX has a decent number of UI components available but lesser than what Swing provides.

Swing is a legacy library that fully features and provide pluggable UI components, whereas JavaFX has UI components that are still evolving with a more advanced look and feel.

Swing can provide UI components with a decent look and feel, whereas JavaFX can provide rich internet application having a modern UI.

Swing related classes can be found in the Java API guide with complete documentation, whereas JavaFX doc is available in various format with comprehensive detailing and file support.

Swing, since its advent, can create UI component using standard Java component classes, whereas Java FX initially uses a declarative language called JavaFX Script.

Swing has a UI component library and act as a legacy, whereas JavaFX has several components built over Swing.

Swing has support for MVC, but it is not consistent across a component, whereas JavaFX support is very friendly with MVC.

Swing has various IDEs, which offer a tool for rapid development, whereas JavaFX has also support from various IDEs as well, but it is not as mature as Swing.

A swing was renamed from Java Foundation Classes, and sun microsystems announced it in the year 1997, whereas JavaFX was initially released in December 2008 by Sun microsystem and now acquired by Oracle.

3.9 JavaFX Layouts

Layouts are the top-level container classes that define the UI styles for scene graph objects. Layout can be seen as the parent node to all the other nodes. JavaFX provides various layout panes that support different styles of layouts.

We have several built-in layout panes in JavaFX that are HBox, VBox, StackPane, FlowBox, AnchorPane, etc.

All these classes belong to `javafx.scene.layout` package. `javafx.scene.layout.Pane` class is the base class for all the built-in layout classes in JavaFX.

Layout Classes

`javafx.scene.layout` Package provides various classes that represents the layouts. The classes are described in the table below.

Class	Description
BorderPane	Organizes nodes in top, left, right, centre and the bottom of the screen.
FlowPane	Organizes the nodes in the horizontal rows according to the available horizontal spaces. Wraps the nodes to the next line if the horizontal space is less than the total width of the nodes
GridPane	Organizes the nodes in the form of rows and columns.
HBox	Organizes the nodes in a single row.
Pane	It is the base class for all the layout classes.
StackPane	Organizes nodes in the form of a stack i.e. one onto another
VBox	Organizes nodes in a vertical column.

Steps to create layout

In order to create the layouts, we need to follow the following steps.

1. Instantiate the respective layout class, for example, `HBox root = new HBox();`
2. Setting the properties for the layout, for example, `root.setSpacing(20);`
3. Adding nodes to the layout object, for example, `root.getChildren().addAll(<NodeObjects>);`

3.10 JavaFX UI Controls

The graphical user interface of every desktop application mainly considers UI elements, layouts and behaviors. The UI elements are the one which are actually shown to the user for interaction or information exchange. Layout defines the organization of the UI elements on the screen. Behavior is the reaction of the UI element when some event is occurred on it.

However, the package `javafx.scene.control` provides all the necessary classes for the UI components like Button, Label, etc. Every class represents a specific UI control and defines some methods for their styling.

SN	Control	Description
1	Label	Label is a component that is used to define a simple text on the screen. Typically, a label is placed with the node, it describes.
2	Button	Button is a component that controls the function of the application. Button class is used to create a labelled button.
3	RadioButton	The Radio Button is used to provide various options to the user. The user can only choose one option among all. A radio button is either selected or deselected.
4	CheckBox	Check Box is used to get the kind of information from the user which contains various choices. User marked the checkbox either on (true) or off(false).

5	TextField	Text Field is basically used to get the input from the user in the form of text. <code>javafx.scene.control.TextField</code> represents TextField
6	PasswordField	PasswordField is used to get the user's password. Whatever is typed in the passwordfield is not shown on the screen to anyone.
7	HyperLink	HyperLink are used to refer any of the webpage through your application. It is represented by the class <code>javafx.scene.control.HyperLink</code>
8	Slider	Slider is used to provide a pane of options to the user in a graphical form where the user needs to move a slider over the range of values to select one of them.
9	ProgressBar	Progress Bar is used to show the work progress to the user. It is represented by the class <code>javafx.scene.control.ProgressBar</code> .
10	ProgressIndicator	Instead of showing the analogue progress to the user, it shows the digital progress so that the user may know the amount of work done in percentage.
11	ScrollBar	JavaFX Scroll Bar is used to provide a scroll bar to the user so that the user can scroll down the application pages.
12	Menu	JavaFX provides a Menu class to implement menus. Menu is the main component of any application.
13	ToolTip	JavaFX ToolTip is used to provide hint to the user about any component. It is mainly used to provide hints about the text fields or password fields being used in the application.

Introduction to JavaFX

JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms.

- With JavaFX, you can build many types of applications.

- Typically, they are network-aware applications that are deployed across multiple platforms and display information in a high-performance modern user interface that features audio, video, graphics, and animations.

The JavaFX GUI Programming

- GUI of an application is made up of Windows (Called Stage in JavaFX)
- A window has a container (called Scene) to host the UI root layout container
- UI components are first added to a root layout container (such as VBox) then placed in the Scene
- UI Components raise Events when the user interacts with them (such as a MouseClicked event is raised when a button is clicked)
- Programmer writes Event Handlers to respond to the UI events

We must have to know about the following terminologies

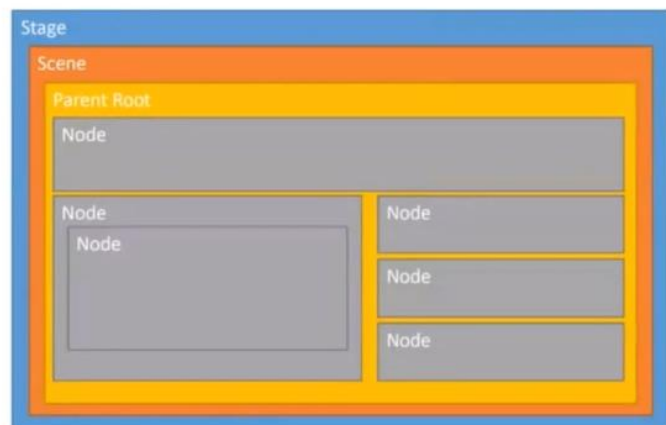
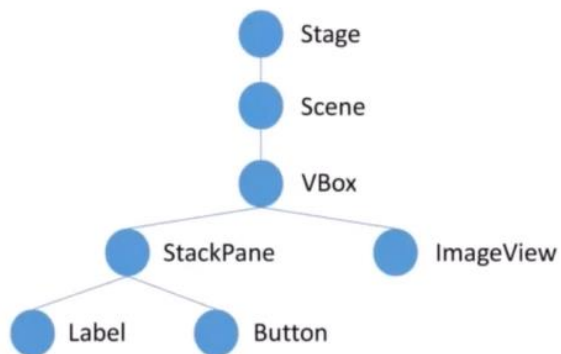
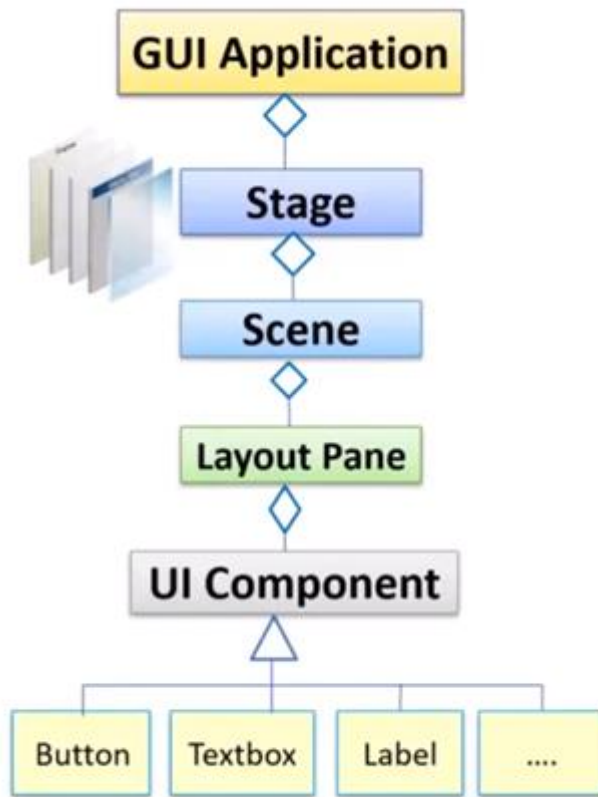
- **Stage**
- **Layout**
- **Scene**
- **Controls**
- **Events**

Stage = Windows where a Scene is displayed

Scene = Container to host the UI root layout container

Stage Methods

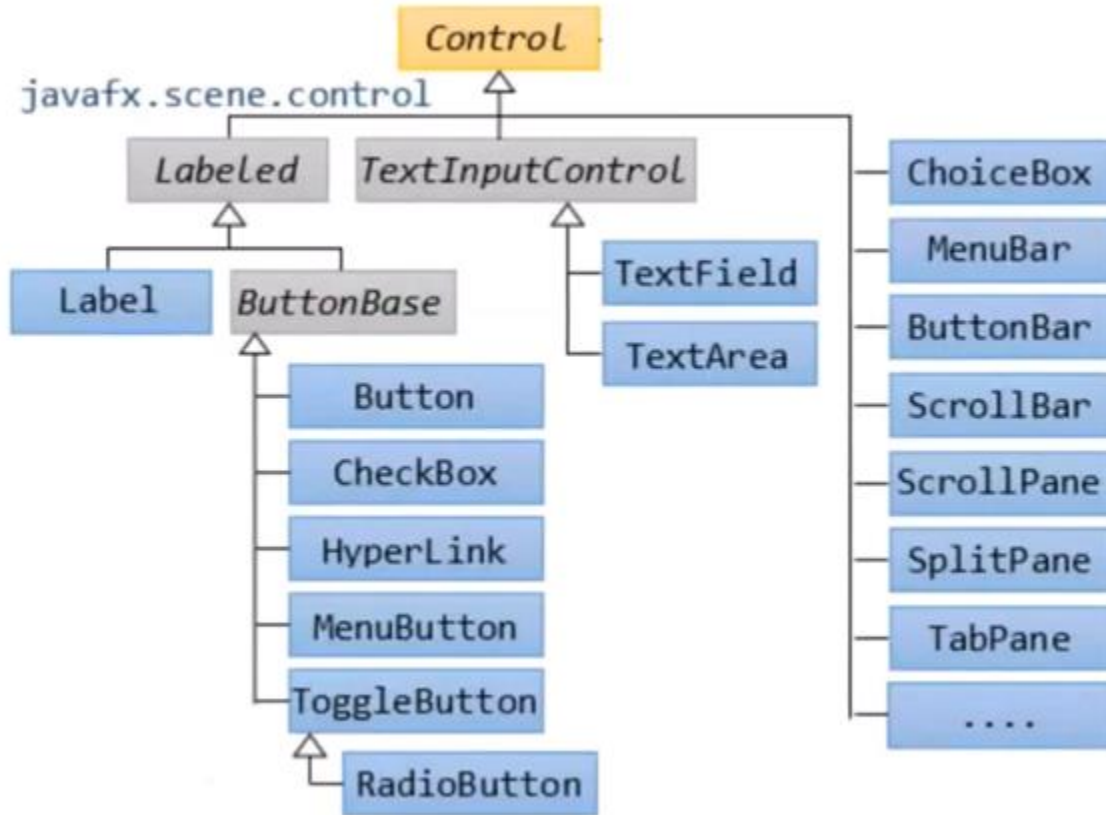
```
primaryStage.show()
primaryStage.setScene(sc)
primaryStage.setTitle(value)
primaryStage.getTitle()
primaryStage.setWidth(value)
primaryStage.getWidth()
primaryStage.setHeight(value)
primaryStage.getHeight()
primaryStage.setFullScreen(true)
primaryStage.isFullScreen()
primaryStage.close()
```

What Makes up JavaFX?

- UI Components
 - Set of pre-built UI components that can be composed to create GUI
 - Ex. Buttons, text-fields, menu, tables, lists, etc.
- Layout containers
 - Control placement/positioning of components in the form of (e.g VBox and HBox)

JavaFX UI Components Hierarchy

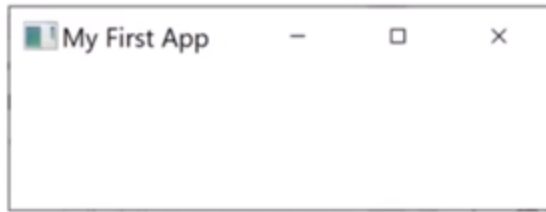


Creating JavaFX GUI Application

- Import `javafx.application.Application` class
- Create a class that extends `javafx.application.Application`
- Override the `start` method in the `Application` class

For example,

```
import javafx.application.Application;
import javafx.stage.Stage;
public class App extends Application{
    @Override
    public void start(Stage stage){
        stage.setTitle("My First App");
        stage.show();
    }
    public static void main(String args[]){
        launch(args);
    }
}
```



JavaFX program to display a button

```
import javafx.application.Application;
import javafx.scene.layout.*;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.stage.Stage;
public class ButtonApp extends Application{
    @Override
    public void start(Stage stage){
        stage.setTitle("Button App");
        //creating controls
        Button btn1= new Button("Click Me");

        //defining layouts
        HBox h = new HBox();
        //adding components to layout
        h.getChildren().add(btn1);

        //creating scene
        Scene sc = new Scene(h);

        //setting scene to stage
        stage.setScene(sc);

        //display stage
        stage.show();
    }
    public static void main(String args[]){
        launch(args);
    }
}
```

Layout

Layout containers or panels can be used to allow for flexible and dynamic arrangements of the UI controls within a scene graph of a JavaFX application. The JavaFX Layout API includes the following container classes that automate common layout models:

- The **HBox** class arranges its content nodes horizontally in a single row.
- The **VBox** class arranges its content nodes vertically in a single column.
- The **StackPane** class places its content nodes in a back-to-front single stack.

- The **FlowPane** class arranges its content nodes in either horizontal or vertical “flow”, wrapping at the specified width (for horizontal) or height (for vertical) boundaries.
- The **GridPane** class enables that developer to create a flexible grid of rows and columns in which to lay out content nodes.
- The **BorderPane** class lays out its content nodes in the top, bottom, right, left, or center region.

HBox



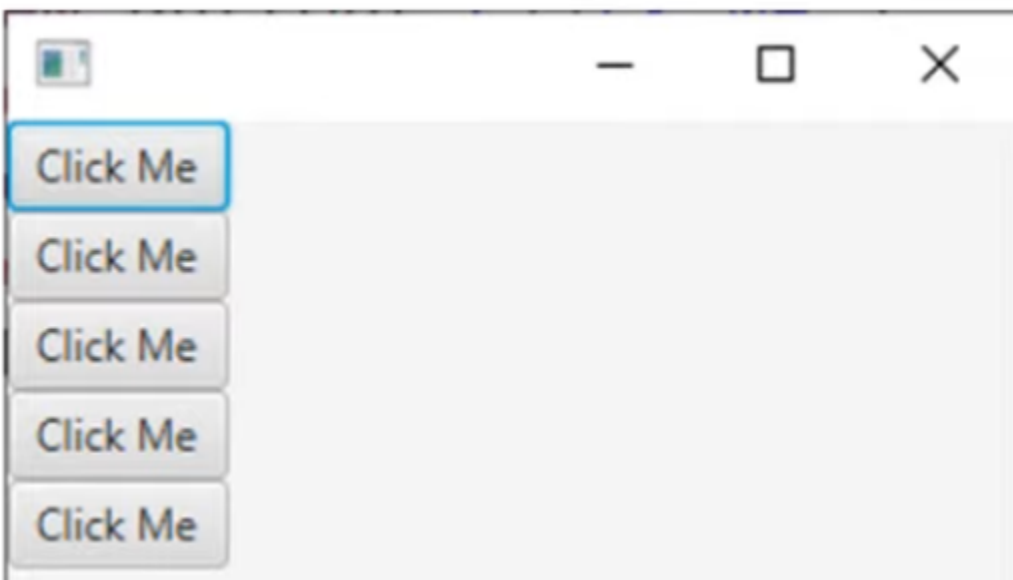
```
// create a HBox
HBox hBox = new HBox();

// create a label
Label label = new Label("this is HBox example");

// add label to hBox
hBox.getChildren().add(label);

// add buttons to hBox
for (int i = 0; i < 5; i++) {
    hBox.getChildren().add(new Button("Button " + (i + 1)));
}
```

VBox



```
// create a VBox
VBox vbox = new VBox();

// create a label
Label label = new Label("this is VBox example");

// add label to vbox
vbox.getChildren().add(label);

// add buttons to vbox
for (int i = 0; i < 5; i++) {
    vbox.getChildren().add(new Button("Button " + (i + 1)));
}
```

StackPane

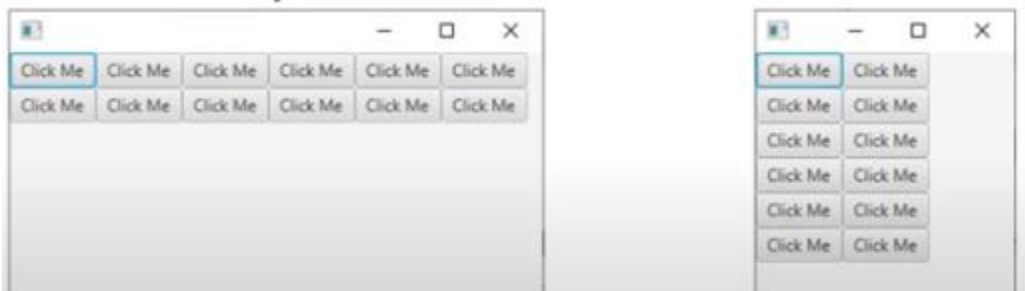
The StackPane layout pane places all the nodes into a single stack where every new node gets placed on the top of the previous node.



```
Button btn1 = new Button("Button 1 on bottom ");
Button btn2 = new Button("Button 2 on top");
StackPane root = new StackPane();
root.getChildren().addAll(btn1, btn2);
```

FlowPane

FlowPane layout panel organizes the nodes in a flow that are wrapped at the flow pane's boundary

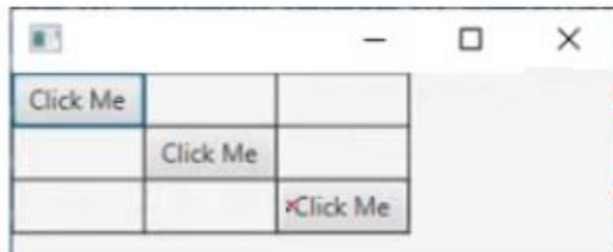
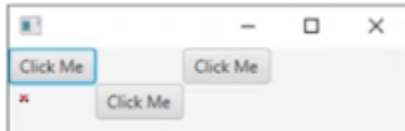


```
FlowPane root = new FlowPane();
root.setVgap(6);
root.setHgap(5);
root.setPrefWrapLength(250);
```

```
root.getChildren().add(new Button("Add"));
root.getChildren().add(new Button("Sub"));
root.getChildren().add(new Button("Mul"));
root.getChildren().add(new Button("Div"));
root.getChildren().add(new Button("Mod"));
```

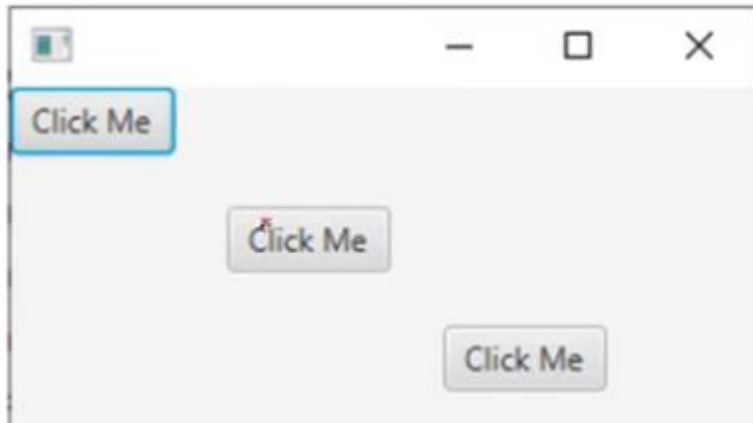
GridPane

GridPane Layout pane allow us to add the multiple nodes in multiple rows and columns. It is seen as a flexible grid of rows and columns where nodes can be placed in any cell of the grid.



```
Label first_name=new Label("First Name");
Label last_name=new Label("Last Name");
TextField tf1=new TextField();
TextField tf2=new TextField();
Button Submit=new Button ("Submit");
GridPane root=new GridPane();
//root.setGridLinesVisible(true);
root.setVgap(10);
root.setHgap(10);
root.addRow(0, first_name,tf1);
root.addRow(1, last_name,tf2);
root.addRow(2, Submit);
```

hGap and VGap



BorderPane

Type	Property	Setter Methods	Description
Node	Bottom	[*] setBottom()	Add the node to the bottom of the screen
Node	Centre	setCentre()	Add the node to the centre of the screen
Node	Left	setLeft()	Add the node to the left of the screen
Node	Right	setRight()	Add the node to the right of the screen
Node	Top	setTop()	Add the node to the top of the screen

```
BorderPane bPane = new BorderPane();
```

```
//Setting the top, bottom, center, right and left nodes to the pane
bPane.setTop(new TextField("Top"));
bPane.setBottom(new TextField("Bottom"));
bPane.setLeft(new TextField("Left"));
bPane.setRight(new TextField("Right"));
bPane.setCenter(new TextField("Center"));
```

JavaFX UI Controls

- Label
- Button
- RadioButton
- CheckBox
- ComboBox
- ListView
- TextField
- PasswordField
- Menu
- Table

Label

Label is a non-editable text control. A Label is useful for displaying text that is required to fit within a specific space.

Label Constructor

Label () Created an empty label

Label (String text) Create Label with supplied text



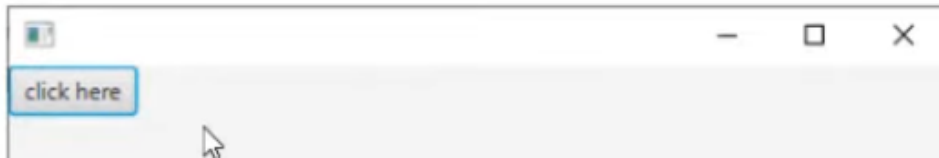
Label Method

```
Label lbl = new Label();  
lbl.setFont(new Font("Times New Roman",32));  
lbl.setText("This is simple text");  
lbl.setTextAlignment(TextAlignment.CENTER);  
//LEFT | CENTER | RIGHT | JUSTIFY  
lbl.setTextFill(Color.RED);  
lbl.setWrapText(true);
```

Setting an image for your Label

```
FileInputStream fis = new FileInputStream("pathofimage");  
Image img = new Image(fis);  
ImageView iv = new ImageView(img);  
Label lbl = new Label("Enter Your Name here",iv);
```

Button



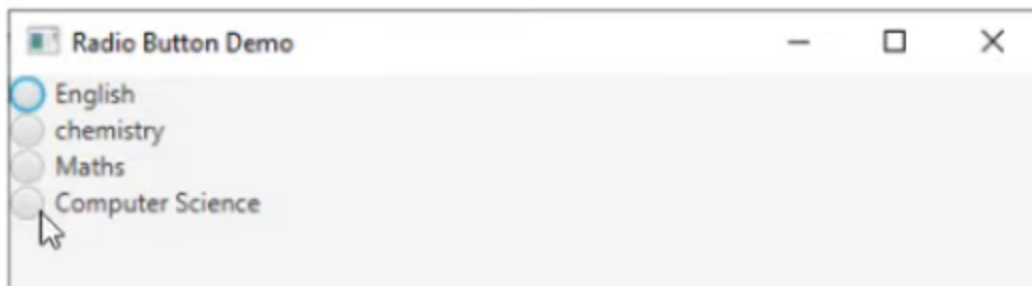
```
Button btn1= new Button("Click Me");
btn1.setText("This is text");
btn1.setWrapText(true);
btn1.setDisable(true);
btn1.setOnAction((event)->{
//code here
});
```

//Setting a button with image

```
FileInputStream fis = new FileInputStream("pathofimage");
Image img = new Image(fis);
ImageView iv = new ImageView(img);

Button btn1= new Button("Click Me",iv);
```

RadioButton

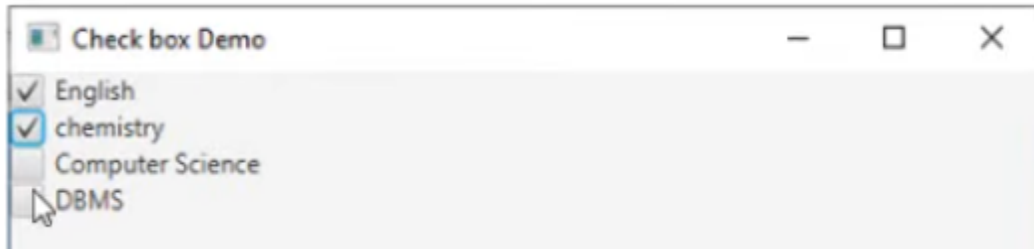


```
RadioButton rdo1= new RadioButton("English");
RadioButton rdo2= new RadioButton("Chemistry");
RadioButton rdo3= new RadioButton("Mathematics");
RadioButton rdo4= new RadioButton("Computer Science");
ToggleGroup group = new ToggleGroup();
rdo1.setToggleGroup(group);
rdo2.setToggleGroup(group);
rdo3.setToggleGroup(group);
rdo4.setToggleGroup(group);

//defining layouts
VBox h = new VBox();

//adding components to layout
h.getChildren().addAll(rdo1,rdo2,rdo3,rdo4);
```

CheckBox



///coding of checkbox

```
CheckBox chk1= new CheckBox("English");  
CheckBox chk2= new CheckBox("Chemistry");  
CheckBox chk3= new CheckBox("Mathematics");  
CheckBox chk4= new CheckBox("Computer Science");
```

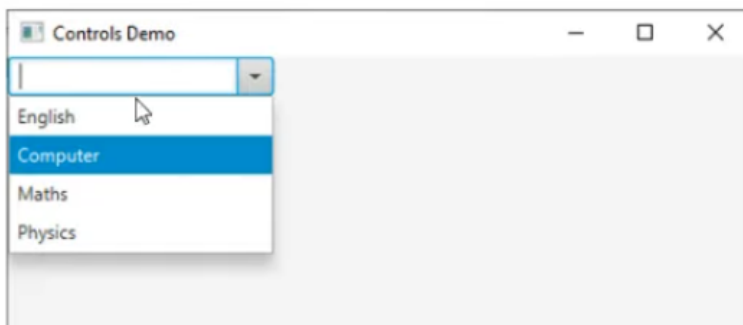
//defining layouts

```
VBox h = new VBox();
```

//adding components to layout

```
h.getChildren().addAll(chk1,chk2,chk3,chk4);
```

ComboBox



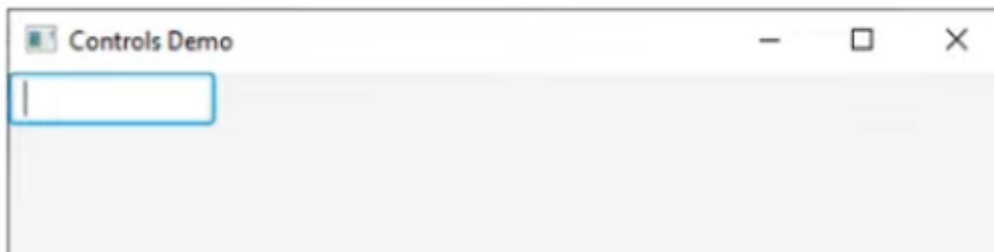
```
ComboBox<String> cbo1= new ComboBox<>();  
cbo1.getItems().add("English");  
cbo1.getItems().add("Physics");  
cbo1.getItems().add("Math");  
cbo1.getItems().add("Java");  
cbo1.setEditable(true);
```

ListView



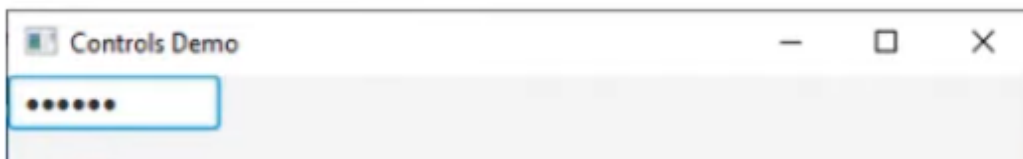
```
ListView<String> cbo1= new ListView<>();  
cbo1.getItems().add("English");  
cbo1.getItems().add("Physics");  
cbo1.getItems().add("Chemistry");  
cbo1.getItems().add("History");  
cbo1.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
```

TextField



```
TextField txt= new TextField();  
txt.setMaxWidth(20);
```

PasswordField



```
PasswordField txtPass= new PasswordField();  
txtPass.setMaxWidth(20);
```

Menu

```
//menu bar  
MenuBar menuBar = new MenuBar();  
  
//menu  
Menu file = new Menu("File");
```

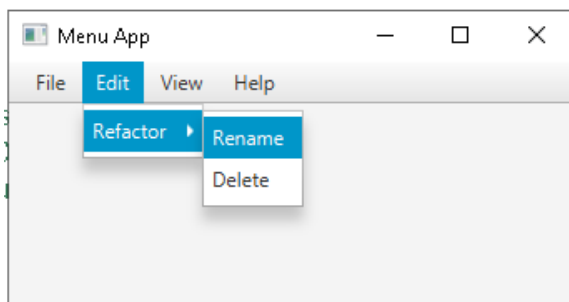
```
Menu edit = new Menu("Edit");
Menu view = new Menu("View");
Menu help = new Menu("Help");

//binding menu to menuBar
menuBar.getMenus().add(file);
menuBar.getMenus().add(edit);
menuBar.getMenus().add(view);
menuBar.getMenus().add(help);

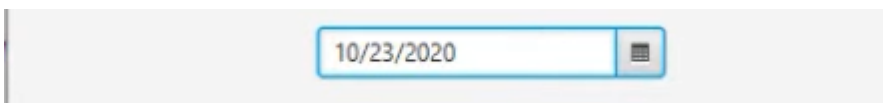
//menu item
file.getItems().add(new MenuItem("Open"));
file.getItems().add(new MenuItem("Save"));
file.getItems().add(new MenuItem("Close"));

//submenu
Menu refactor = new Menu("Refactor");
MenuItem rename = new MenuItem("Rename");
MenuItem delete = new MenuItem("Delete");
refactor.getItems().add(rename);
refactor.getItems().add(delete);
edit.getItems().add(refactor);

//add to layout
BorderPane root = new BorderPane();
root.setTop(menuBar);
```

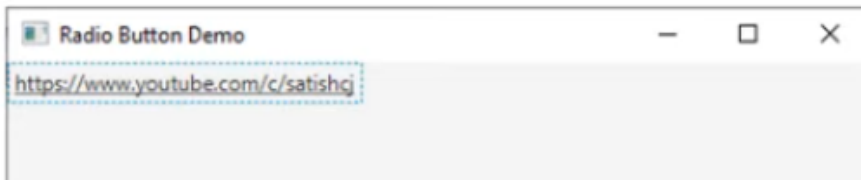


DatePicker



```
DatePicker picker = new DatePicker();
```

Hyperlink



```
Hyperlink youtube= new Hyperlink("https://www.youtube.com/");
```

JavaFX Event Handling

To handle JavaFX event you need to **import javafx.event package**

```
Button btn1= new Button("Click Me");
btn1.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Clicked");
    }
});
```

```
// lambda expression
btn1.setOnAction((event) -> {
    btn1.setText("You Clicked!");
});
```

For details:

https://docs.oracle.com/javafx/2/ui_controls/jfxpub-ui_controls.htm