ADVANCED JAVA PROGRAMMING

BCA-V SEM

NOTES

Prepared By

Prof. Prabhu Kichadi, M.Tech
KLES SSMS BCA College Athani

UNIT – II

CONTENTS

JFC: *Java Foundation Classes*

(JFC) are a comprehensive set of GUI components and services which dramatically simplify the development and deployment of commercial-quality desktop and Internet/Intranet applications.

Swing:

Swing is a part of JFC (Java Foundation Classes) which provides a set of libraries to create GUI (Graphical User Interface) in a platform independently in java.

- Provides many Java GUI Controls.

The **javax. swing** package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

AWT:

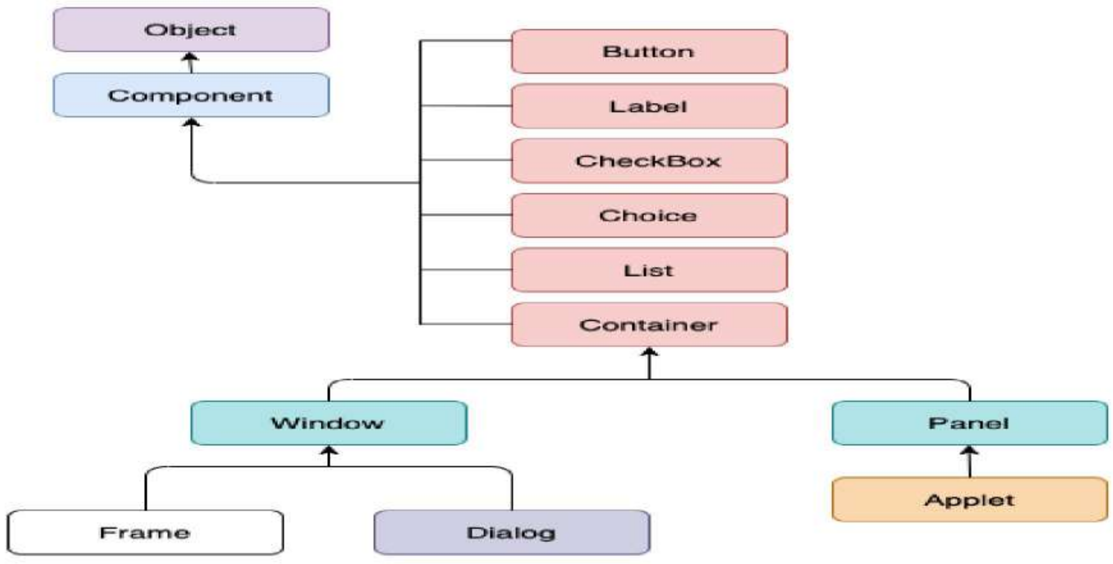
"AWT stands for Abstract window Toolkit is an Application programming interface (API) for creating Graphical User Interface (GUI) in Java. It allows Java programmers to develop window-based applications".

The **java.awt** package provides GUI components in the form of classes.

AWT Features:

- ➢ AWT Components are Platform Dependent.
- ➢ AWT Components are heavy weight.
- ➢ Requires more usage of logic and system resources.

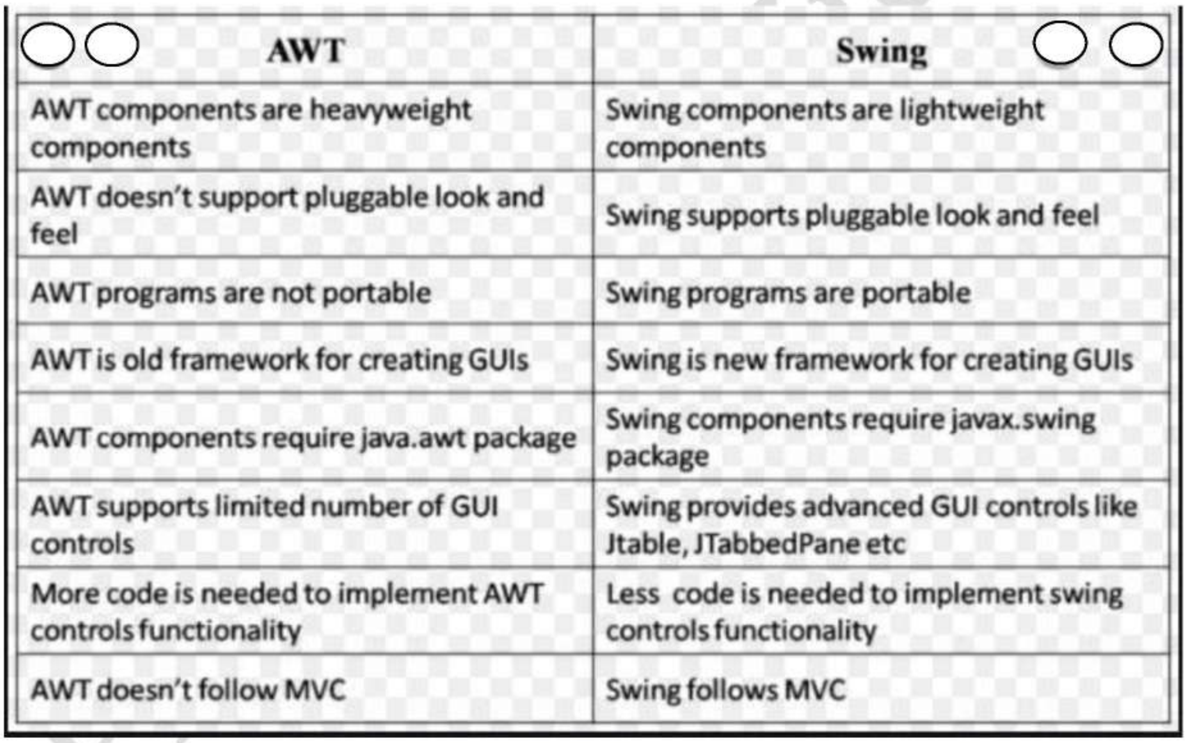
The following image represents the hierarchy for Java AWT



Swing Features:

- **Light Weight** – Swing components are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.
- **Rich Controls** – Swing provides a rich set of advanced controls like Tree, JTabbedPane, JSlider, Jcolorpicker, and JTable controls.
- **Highly Customizable** – Swing controls can be customized in a very easy way as visual appearance is independent of internal representation.
- Pluggable look-and-feel – SWING based GUI Application look and feel can be changed at run-time, based on available values.
- Swing follows **MVC**.
- Java swing components are **platform-independent.**

Difference Between swing and AWT



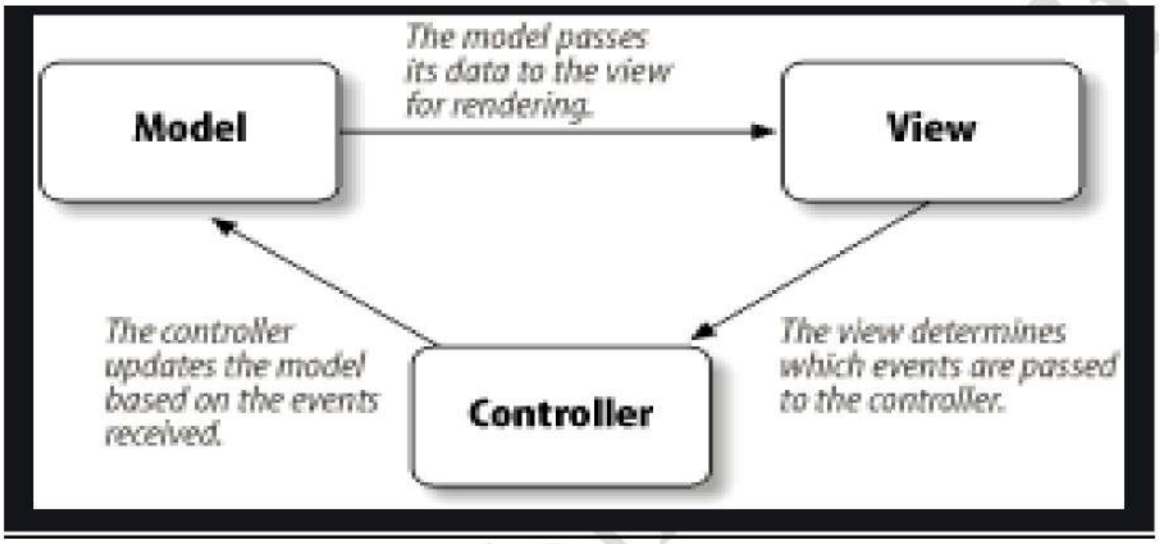
AWT	Swing
AWT components are heavyweight components	Swing components are lightweight components
AWT doesn't support pluggable look and feel	Swing supports pluggable look and feel
AWT programs are not portable	Swing programs are portable
AWT is old framework for creating GUIs	Swing is new framework for creating GUIs
AWT components require java.awt package	Swing components require javax.swing package
AWT supports limited number of GUI controls	Swing provides advanced GUI controls like Jtable, JTabbedPane etc
More code is needed to implement AWT controls functionality	Less code is needed to implement swing controls functionality
AWT doesn't follow MVC	Swing follows MVC

MVC Architecture or MVC Model

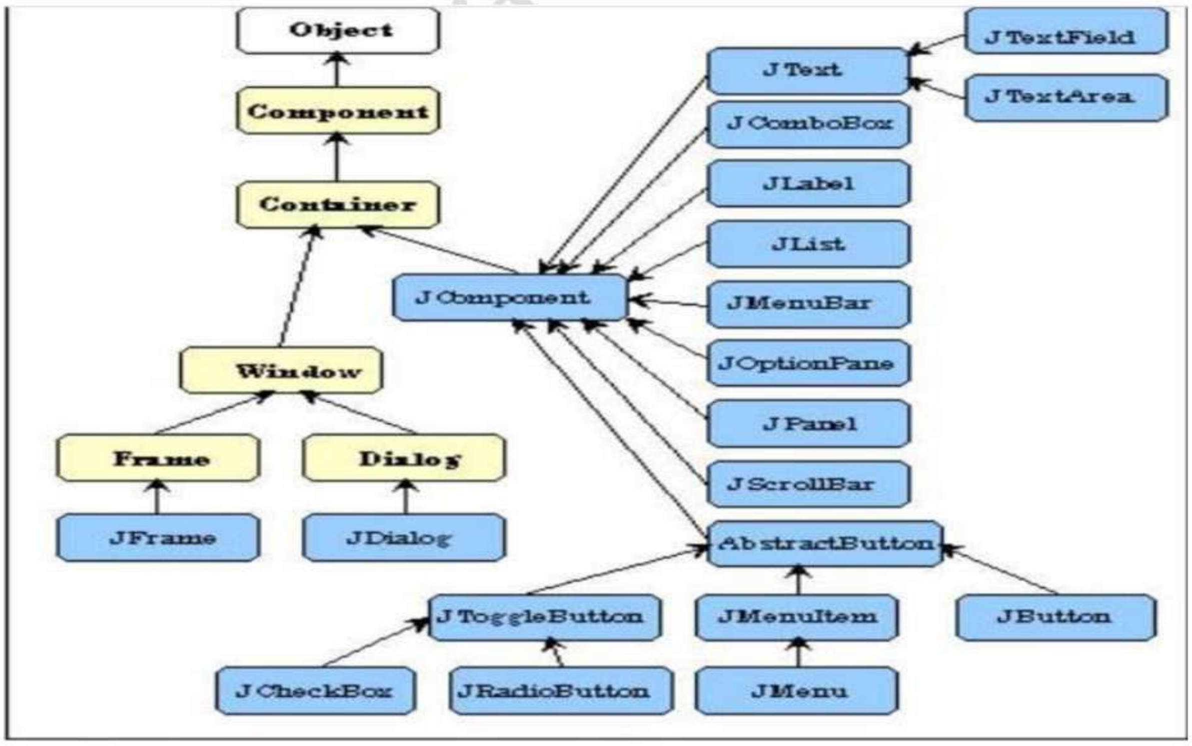
MVC Architecture: Model–view–controller (usually known as MVC) is a software design pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements.

Swing API architecture follows loosely based MVC architecture in the following manner.

- Model represents component's data.
- View represents visual representation of the component's data.
- Controller takes the input from the user on the view and reflects the changes in Component's data.
- Swing component has Model as a separate element, while the View and Controller part are clubbed in the User Interface elements. Because of which, Swing has a pluggable look-and-feel architecture.


Model
The model passes its data to the view for rendering.
View
The controller updates the model based on the events received.
Controller
The view determines which events are passed to the controller.

Hierarchy of Java Swing Classes:



Containers & Components

Containers are an integral part of SWING GUI components. A container provides a space where a component can be located, Container is a top-level component it can add other components like JButton, JRadioButton, etc,

SWING Containers

Following is the list of commonly used containers while designed GUI using SWING.

Note: Child classes of ***Container*** class are all containers in swings.

1. **Panel: *JPanel*** is the simplest container. It provides space in which any other component can be placed, including other panels.
2. **Frame**: A ***JFrame*** is a top-level window with a title and a border.
3. **Window:** A ***JWindow*** object is a top-level window with no borders and no menu bar, no management buttons(close, minimize, maximize).
4. **Dialog:** A ***JDialog*** The JDialog control represents a top-level window with a border and a title used to take some form of input from the user. It inherits the Dialog class.

   Unlike JFrame, it doesn't have maximize and minimize buttons.
5. **Applet:** A ***JApplet*** object is used to run dynamic content on the web page, runs on embedded in html tag <applet code=" ">

Containers hierarchy:

JFrame->JPanel->JDialog->JApplet,

Container creation steps:/GUI creation steps.

1. To create any GUI using swing components, first we need a top-level container to hold components.

2. Create Container first using container classes.

3. Set appearance properties to container like,

Size, layout, visibility, close operation, title, etc.

Ex :

```
JFrame fr = new JFrame();
fr.setLayout(null);
fr.setBounds(200, 200, 600, 600);
frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
fr.setVisible(true);
```

4. Then create components and set appearance properties for components.

5. Now add components (JButton) to container(JFrame).

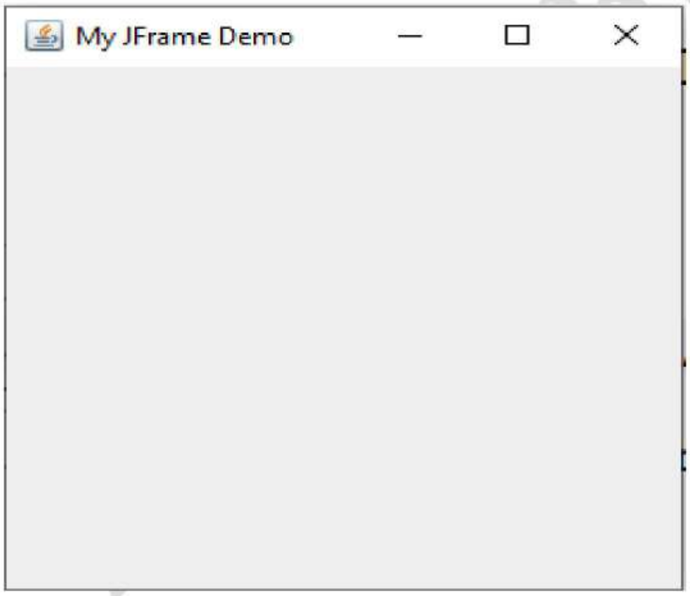
6. Register listeners with event sources.

7. Handle events using listener interfaces.

Programs: JFrame creation – Top-level container

```
import javax.swing.JFrame;
public class MyConatiner
{
	public static void main(String[] args)
	{
		JFrame frm = new JFrame("My JFrame Demo");
		//Appearance properties
		frm.setLayout(null);
		frm.setBounds(100, 100, 600, 600);
		frm.setVisible(true);
		frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

Output:



SWING Components:

Swing components are the basic building blocks of an GUI, these are atomic in nature, Components are UI elements that can be added into containers.

Ex: ***JButton*** component can be added into ***JFrame*** container.

Note: Child classes of ***JComponent*** class are all components in swings.

Some of SWING components are:

1. ***JLabel:*** *A* ***JLabel*** object is a component for placing text in a container.
2. ***JButton:*** This class creates a labelled button.
3. ***JTextField***: A ***JTextField*** object is a text component that allows for the editing of a single line of text.
4. ***JCheckBox:*** A ***JCheckBox*** is a graphical component that can be in either an on (true) or off (false) state.
5. ***JRadioButton:*** The ***JRadioButton*** class is a graphical component that can be in either an on (true) or off (false) state. in a group.
6. ***JList:*** A ***JList*** component presents the user with a scrolling list of text items.
7. ***JTextArea***: A ***JTextArea*** object is a text component that allows editing of a multiple lines of text.
8. ***ImageIcon:*** A ***ImageIcon*** control is an implementation of the Icon interface that paints Icons from Images.
9. ***JScrollbar:*** A ***JScrollbar*** control represents a scroll bar component in order to enable the user to select from range of value's.
10. ***JOptionPane: JOptionPane*** provides set of standard dialog boxes that prompt users for a value or informs them of something.
11. ***JSlider***: A ***JSlider*** lets the user graphically select a value by sliding a knob within a bounded interval.
12. ***JComboBox:*** A ***JComboBox*** component presents the user with a to show up menu of choices.
13. ***JTable:*** The ***JTable*** class is used to display data in tabular form. It is composed of rows and columns.
14. ***JList:*** The object of ***JList*** class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

15. JtabbedPane: The ***JTabbedPane*** class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.

16. JScrollPane: A ***JScrollPane*** is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

Containers:

1. JComponent:

The JComponent class is the base class of all Swing components except top-level containers. Swing components whose names begin with "J" are descendants of the JComponent class.

For example, JButton, JScrollPane, JPanel, JTable etc. But, JFrame and JDialog don't inherit JComponent class because they are the child of top-level containers.

The JComponent class extends the Container class which itself extends Component. The Container class has support for adding components to the container.

Constructors:

JComponent() : Default JComponent constructor.

2. JFrame:

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

Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

Constructors

Constructor	Description
JFrame()	It constructs a new frame that is initially invisible.
JFrame(GraphicsConfiguration gc)	It creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.
JFrame(String title)	It creates a new, initially invisible Frame with the specified title.
JFrame(String title, GraphicsConfiguration gc)	It creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

Methods:

protected void addImpl(Component comp, Object constraints, int index)

Adds the specified child Component.

protected JRootPane createRootPane()

Called by the constructor methods to create the default rootPane.

protected void frameInit()

Called by the constructors to init the JFrame properly.

AccessibleContext getAccessibleContext()

Gets the AccessibleContext associated with this JFrame.

Container getContentPane()

Returns the contentPane object for this frame.

int getDefaultCloseOperation()

Returns the operation that occurs when the user initiates a "close" on this frame.

Component getGlassPane()

Returns the glassPane object for this frame.

Graphics getGraphics()

Creates a graphics context for this component.

JMenuBar getJMenuBar()

Returns the menubar set on this frame.

JLayeredPane getLayeredPane()

Returns the layeredPane object for this frame.

JRootPane getRootPane()

Returns the rootPane object for this frame.

protected String paramString()

Returns a string representation of this JFrame.

void remove(Component comp)

Removes the specified component from the container.

void setContentPane(Container contentPane)

Sets the contentPane property.

A Sample Java Swing Desktop Application:

General steps for java swing application creation:

1. import javax.swing.* & java.awt.event.*;
2. Create a container by extending JFrame class or Create an Object of JFrame.
3. Set the appearance properties for frame like size, visibility, close operation etc.
    i. setSize(int width, int height);
    ii. setBounds(int x, int y, int width, int height);
    iii. setVisible(Boolean val);
    iv. setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
4. Create swing components like labels, buttons, tables etc using respective component classes. Ex: JLabel lab = new JLabel();

5. Set the appearance properties like alignment, size, additional logic if required.
6. Add the component into a container by using add(Component) method of JFrame. Ex: fr.add(lab);
7. Register Listeners if we want to handle events for sources.

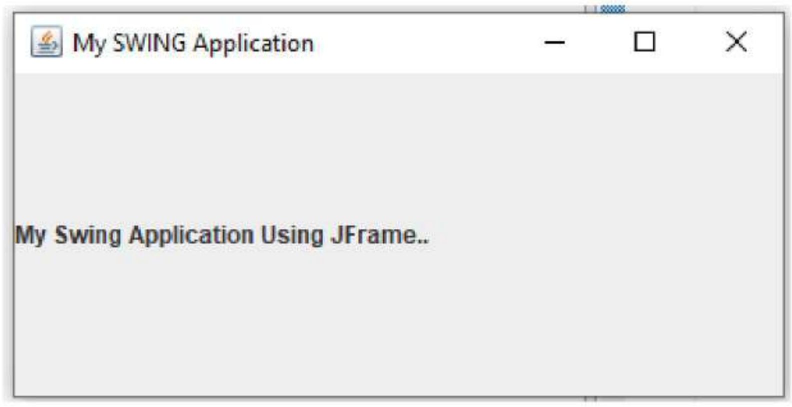
Examples:

1. Creating a Swing application by extending JFrame class.

(Creating a Frame and Adding label component in it)

```
package com.kle.bca;

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

public class MyFrame extends JFrame
{
	JLabel lab1;

	MyFrame()
	{
		setSize(600, 600);
		setTitle("My SWING Application");
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setVisible(true);
		lab1 = new JLabel("My Swing Application Using JFrame..");
		add(lab1);
	}


	public static void main(String[] args)
	{
		MyFrame fr = new MyFrame();
	}
}
```

Output:

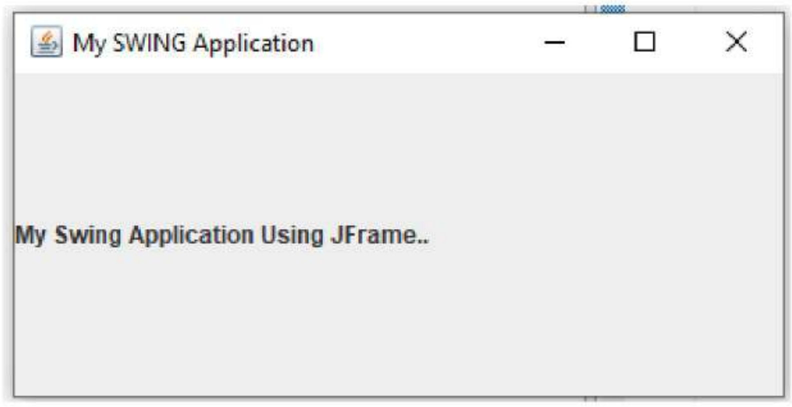


My Swing Application Using JFrame..

2. Creating a Swing application by creating JFrame class object.

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

public class MyFrame1
{
	JFrame fr;
	JLabel lab1;

	public MyFrame1()
	{
		fr = new JFrame("My SWING Application");
		fr.setSize(400, 200);
		fr.setTitle("My SWING Application");
		fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		fr.setVisible(true);
		lab1 = new JLabel("My Swing Application Using JFrame..");
		fr.add(lab1);
	}


	public static void main(String[] args)
	{
		MyFrame1 fr = new MyFrame1();
	}
}
```

Output:



My Swing Application Using JFrame..

JPanel

JPanel, a part of Java Swing package, is a container that can store a group of components.

The main task of ***JPanel*** is to organize components, various layouts can be set in JPanel which provide better organization of components; however, it does not have a title bar.

Constructors:

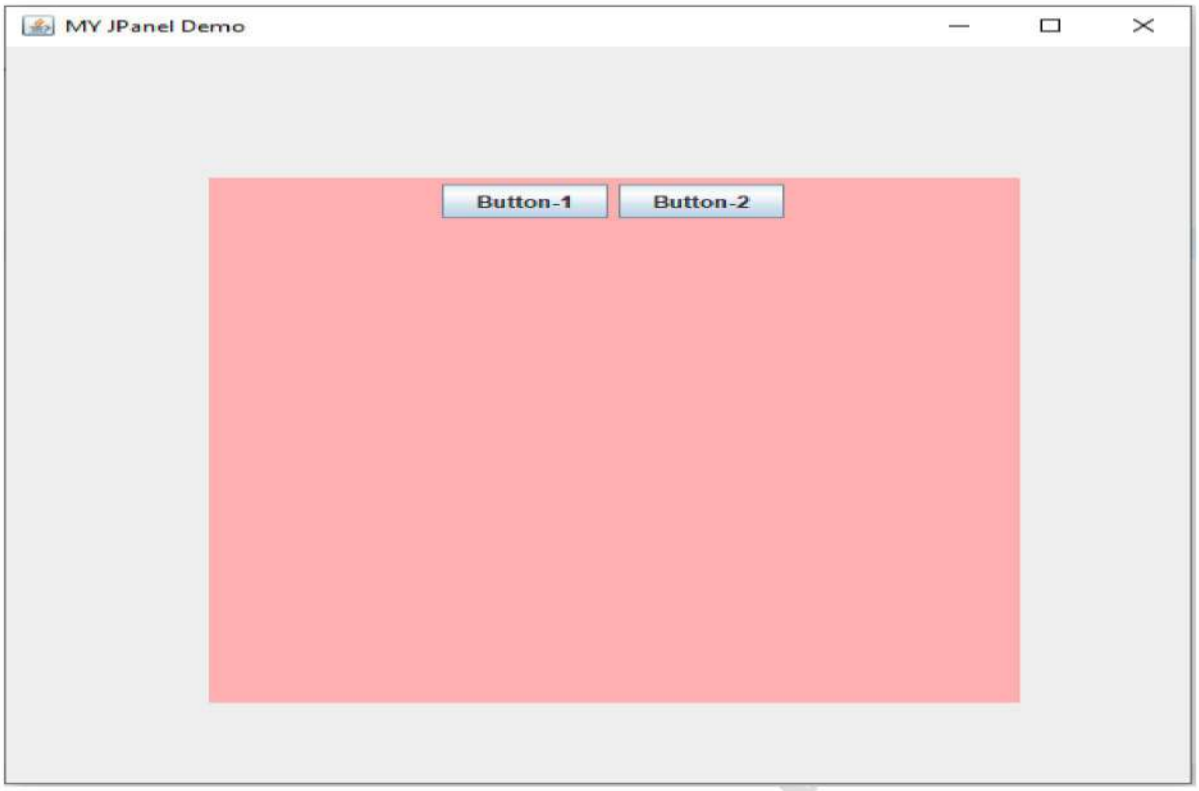
1	**JPanel()** Creates a new JPanel with a double buffer and a flow layout.
2	**JPanel(boolean isDoubleBuffered)** Creates a new JPanel with FlowLayout and the specified buffering strategy.
3	**JPanel(LayoutManager layout)** Creates a new buffered JPanel with the specified layout manager.
4	**JPanel(LayoutManager layout, boolean isDoubleBuffered)** Creates a new JPanel with the specified layout manager and buffering strategy.

Java JPanel Example

Creating JFrame, Adding JPanel in it, intern JPanel contains 2 Buttons:

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

public class MyJPanel
{
	JFrame fr;
	JPanel jp;
	JButton btn1, btn2;

	public MyJPanel()
	{
		fr = new JFrame("MY JPanel Demo");
		fr.setBounds(100,100, 600, 600);
		fr.setLayout(null);
		fr.setDefaultCloseOperation(fr.EXIT_ON_CLOSE);
		fr.setVisible(true);

		jp = new JPanel();
		jp.setBackground(Color.pink);
		jp.setBounds(100,100,400,400);

		fr.add(jp);

		btn1 = new JButton("Button-1");
		btn1.setBounds(100, 100, 80, 60);
		jp.add(btn1);

		btn2 = new JButton("Button-2");
		btn2.setBounds(100, 100, 80, 60);
		jp.add(btn2);
	}
	public static void main(String[] args)
	{
		MyJPanel mp = new MyJPanel();
	}
}
```

Output:

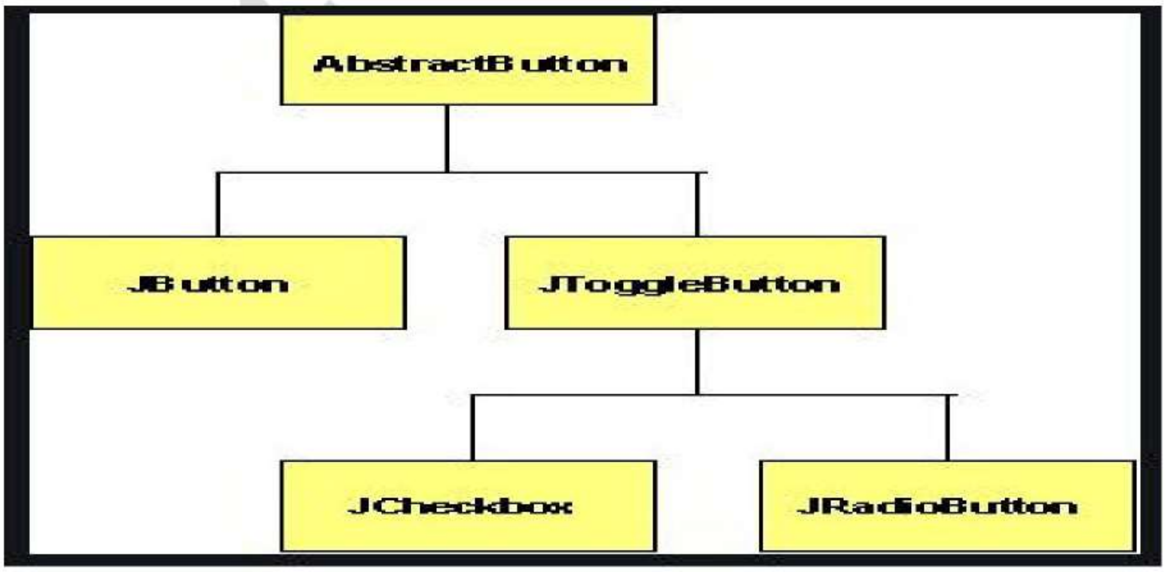


SWING Buttons

javax. swing package provides many action-oriented UI buttons, AbstractButton is the base-class for all buttons in swing.

Generally, there are three types of buttons available in java swing GUI.

1. JButton
2. JCheckbox
3. JRadioButton



1. JButton:

- The ***JButton*** class is used to create a labeled button that has platform independent implementation.
- The application result in some action when the button is pushed. It inherits AbstractButton class.
- It is present in ***javax.swing*** package.

JButton class declaration:

```
public class JButton extends AbstractButton implements Accessible
```

Constructors:

JButton()	- It creates a button with no text and icon.
JButton(String s)	- It creates a button with the specified text.
JButton(Icon i)	- It creates a button with the specified icon object.

Methods:

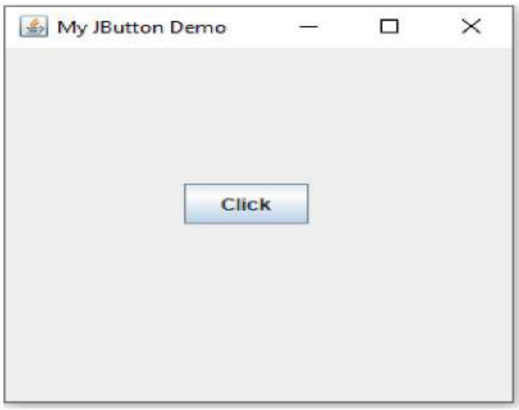
void setText(String s)	- It is used to set specified text on button
String getText()	- It is used to return the text of the button.
void setEnabled(boolean b)	- It is used to enable or disable the button.
void setIcon(Icon b)	- It is used to set the specified Icon on the button.
Icon getIcon()	- It is used to get the Icon of the button.
void setMnemonic(int a)	- It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	- It is used to add the ActionListener to this object.

JButton Example

```
import javax.swing.JButton;
import javax.swing.JFrame;

public class MyJButton1 extends JFrame
{
        JButton btn;
        public MyJButton1()
        {
                setTitle("My JButton Demo");
                setSize(600, 600);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
```

```
            setLayout(null);
            setVisible(true);
            btn = new JButton("Click");
            btn.setBounds(100,100, 70, 30);
            add(btn);
      }
      public static void main(String[] args)
      {
            new MyJButton1();
      }
}
```

Output:



1. JCheckbox:

The ***JCheckBox*** class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ". It inherits ***JToggleButton*** class.

JCheckbox class declaration:

```
public class JCheckBox extends JToggleButton implements Accessible
```

Constructors:

JCheckBox()	- Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	- Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	- Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	- Creates a check box where properties are taken from the Action supplied.

Methods:

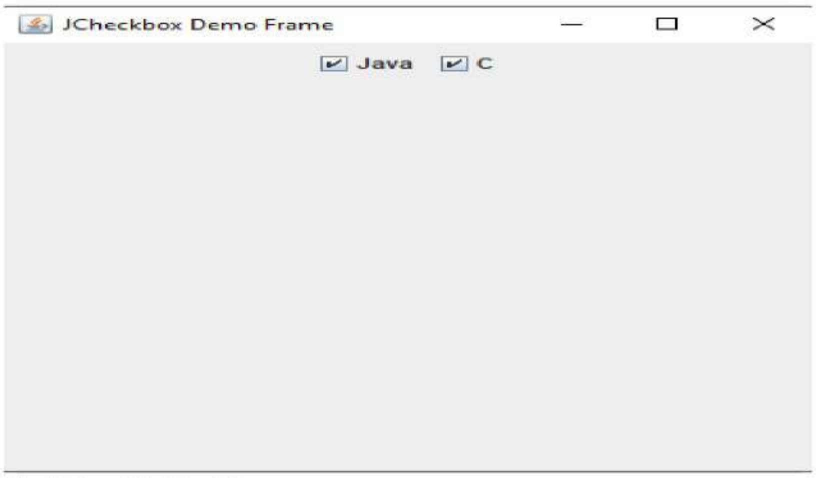
AccessibleContext getAccessibleContext() - It is used to get the

	AccessibleContext associated with this JCheckBox.
protected String paramString()	- It returns a string representation of this JCheckBox.

JCheckbox Example

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

public class MyCheckbox1
{

        JFrame fr;
        JCheckBox cb1,cb2;
        JLabel jb;

        public MyCheckbox1()
        {
                fr = new JFrame("JCheckbox Demo Frame");
                fr.setSize(400, 400);
                fr.setDefaultCloseOperation(fr.EXIT_ON_CLOSE);
                fr.setLayout(new FlowLayout());
                fr.setVisible(true);

                cb1 = new JCheckBox("Java", true);
                cb2 = new JCheckBox("C", true);

                cb1.setBounds(100,100, 50,50);
                cb1.setBounds(100,180, 50,50);

                fr.add(cb1);
                fr.add(cb2);

        }
        public static void main(String[] args)
        {
                new MyCheckbox1();
        }
}
```

Output:



3. JRadioButton:

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

Note: It should be added in ButtonGroup to select one radio button only.

JRadioButton class declaration:

```
public class JRadioButton extends JToggleButton implements Accessible
```

Constructors:

JRadioButton()	- Creates an unselected radio button with no text.
JRadioButton(String s)	- Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	- Creates a radio button with the specified text and selected status.

Methods:

Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.

void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

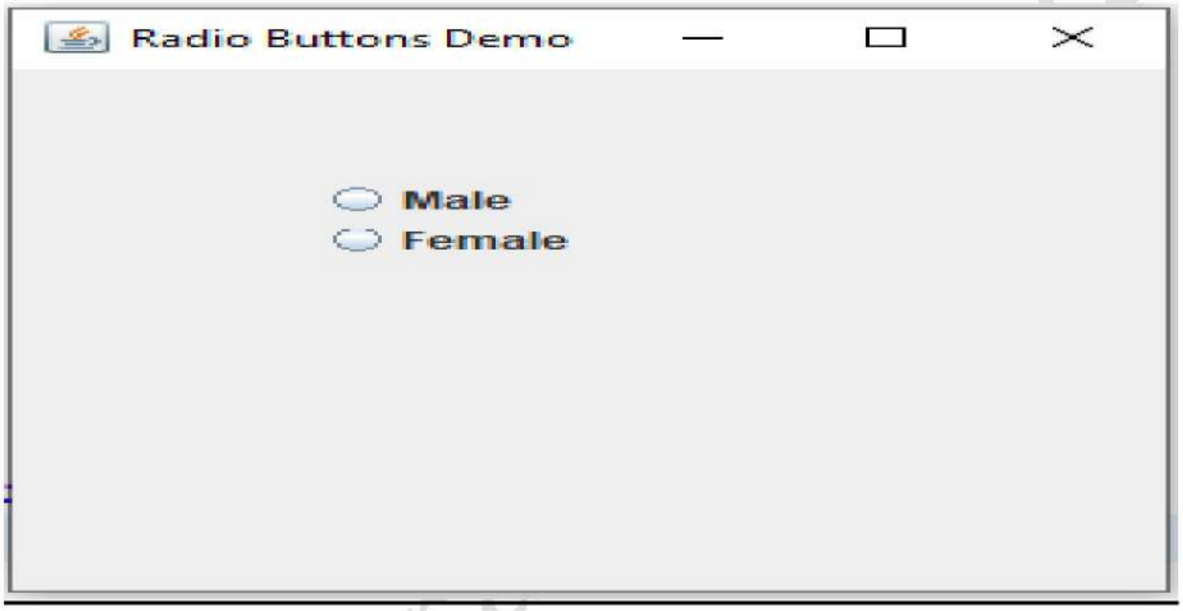
JRadioButton Example:

We need add JRadioButton in ButtonGroup, hence we have to create ButtonGroup Object.

```
import javax.swing.*;

public class MyJRadioButton
{
	JFrame fr;
	ButtonGroup bg;
	JRadioButton b1,b2;

	public MyJRadioButton()
	{
		fr = new JFrame("Radio Buttons Demo");
		fr.setBounds(100,100, 600, 600);
		fr.setLayout(null);
		fr.setDefaultCloseOperation(fr.EXIT_ON_CLOSE);
		fr.setVisible(true);

		bg = new ButtonGroup();

		b1 = new JRadioButton("Male");
		b2 = new JRadioButton("Female");

		b1.setBounds(75,50,100,30);
		b2.setBounds(75,70,100,30);

		bg.add(b1);
		bg.add(b2);
```

```
            fr.add(b1);
            fr.add(b2);
        }

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

Output:



JTabbedPane

The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.

JTabbedPane class declaration:

```
public class JTabbedPane extends JComponent implements Serializable, Accessib
le, SwingConstants
```

Constructors:

JTabbedPane() - Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.

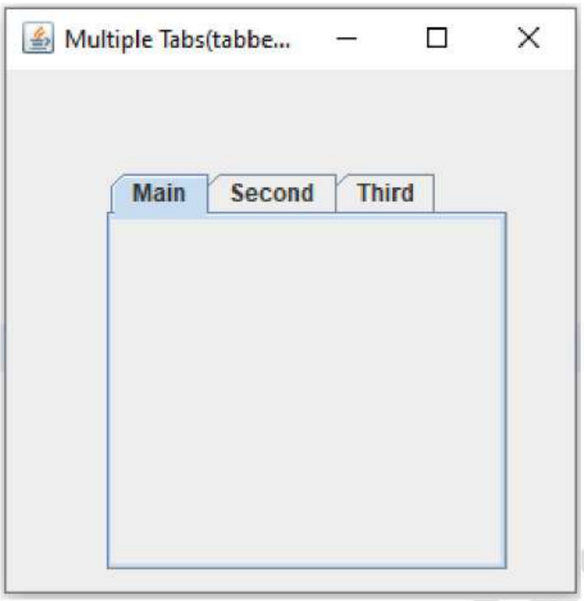
JTabbedPane(int tabPlacement)	- Creates an empty TabbedPane with a specified tab placement.
JTabbedPane(int tabPlacement, int tabLayoutPolicy)	- Creates an empty TabbedPane with a specified tab placement and tab layout policy.

JTabbedPane Example:

- Generally create panels, add panels to tabbedpane object, then add tabbedpane to frames.

```
import javax.swing.*;

public class MyJTabbedPane
{
	JFrame fr;
	JTabbedPane tabs;
	JPanel p1, p2,p3;

	public MyJTabbedPane()
	{
		fr = new JFrame("Multiple Tabs(tabbed_pane) Demo");
		fr.setBounds(100,100, 300, 300);
		fr.setLayout(null);
		fr.setDefaultCloseOperation(fr.EXIT_ON_CLOSE);
		fr.setVisible(true);

		tabs = new JTabbedPane();
		tabs.setBounds(50,50,200,200);

		p1 = new JPanel();
		p2 = new JPanel();
		p3 = new JPanel();

		tabs.add("Main", p1);
		tabs.add("Second", p2);
		tabs.add("Third", p3);

		fr.add(tabs);

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

Output:



JScrollPane

A ***JscrollPane*** is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component **whose size can change dynamically.**

Constructors:

JScrollPane()
JScrollPane(Component) It creates a scroll pane. The

JScrollPane(int, int)
JScrollPane(Component, int, int)
	Component parameter, when present, sets the scroll pane's client. The two int parameters, when present, set the vertical and horizontal scroll bar policies (respectively).

Methods:

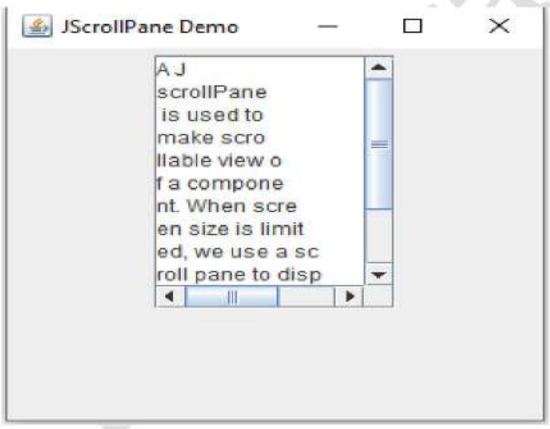
Methods	Description
void setColumnHeaderView(Component)	It sets the column header for the scroll pane.
void setRowHeaderView(Component)	It sets the row header for the scroll pane.
void setCorner(String, Component) Component getCorner(String)	It sets or gets the specified corner. The int parameter specifies which corner and must be one of the following constants defined in ScrollPaneConstants: UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER, LOWER_LEFT_CORNER, LOWER_RIGHT_CORNER, LOWER_LEADING_CORNER, LOWER_TRAILING_CORNER, UPPER_LEADING_CORNER, UPPER_TRAILING_CORNER.
void setViewportView(Component)	Set the scroll pane's client.

JscrollPane Example:

- In this example we are creating a Scrollable Component, if the size of the window is a matter, we can resize dynamically.
- Creating TextArea adding into JScrollPane object.

```
import java.awt.*;
import javax.swing.*;
public class MyJScrollPane
{
	JFrame fr;
	JTextArea txarea;
	JScrollPane jsp;
	public MyJScrollPane()
	{
		fr = new JFrame("JScrollPane Demo");
		fr.setBounds(100,100, 300, 300);
		fr.setLayout(new FlowLayout());
		fr.setDefaultCloseOperation(fr.EXIT_ON_CLOSE);
		fr.setVisible(true);
		txarea = new JTextArea(10,10);
		jsp = new JScrollPane(txarea);

	jsp.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROL
LBAR_ALWAYS);

	jsp.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR
_ALWAYS);
		fr.add(jsp);
	}
	public static void main(String[] args)
	{
		new MyJScrollPane();
	}
}
```

Output:



```
A J
scrollPane
 is used to
make scro
llable view o
f a compone
nt. When scre
en size is limit
ed, we use a sc
roll pane to disp
```

JList

The object of ***JList*** class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

JList class declaration syntax:

```
	public class JList extends JComponent implements Scrollable, Accessible
```

Constructors:

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

Methods:

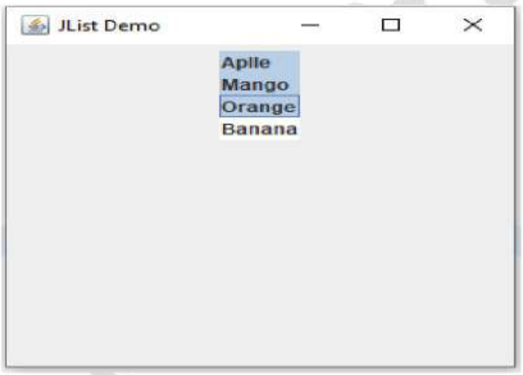
Methods	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return the data model that holds a list of items displayed by the JList component.
void setListData(Object[] listData)	It is used to create a read-only ListModel from an array of objects.

JList Example:

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

```
public class MyJList
{
      JFrame fr;
      DefaultListModel<String> dfm;
      JList<String> j;
      public MyJList()
      {
            fr = new JFrame("JList Demo");
            fr.setBounds(100,100, 300, 300);
            fr.setLayout(new FlowLayout());
            fr.setDefaultCloseOperation(fr.EXIT_ON_CLOSE);
            fr.setVisible(true);
            dfm = new DefaultListModel<>();
            dfm.addElement("Apple");
            dfm.addElement("Mango");
            dfm.addElement("Orange");
            dfm.addElement("Banana");
            j = new JList<>(dfm);
            fr.add(j);
      }
      public static void main(String[] args)
      {
            new MyJList();
      }
}
```

Output:



JComboBox

The object of ***JComboBox*** class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

JComboBox class declaration:

```
public class JComboBox extends JComponent implements ItemSelectable, ListDataListener, ActionListener, Accessible
```

Constructors:

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array.
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

Methods:

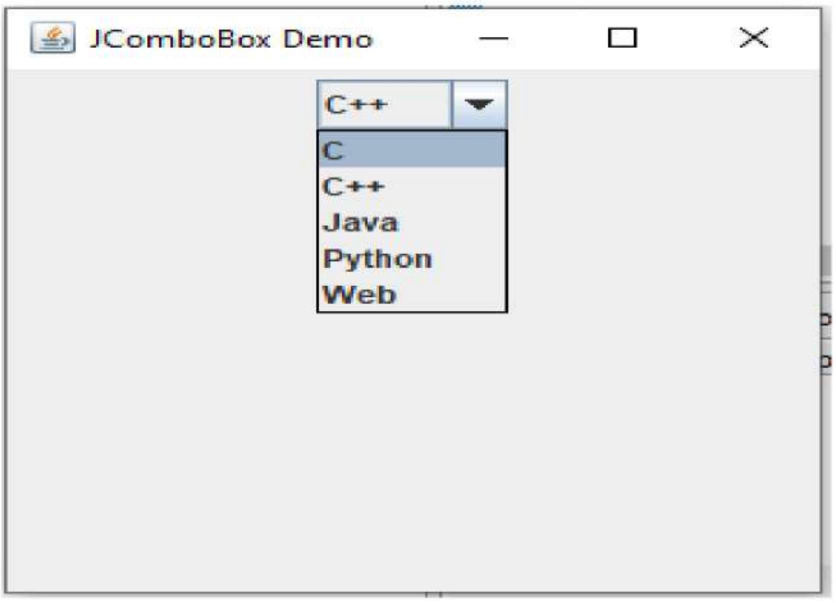
Methods	Description
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox is editable.
void addActionListener(ActionListener a)	It is used to add the ActionListener.

void addItemListener(ItemListener i)	It is used to add the ItemListener.

JComboBox Example:

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

public class MyJComboBox
{

	JFrame fr;
	JComboBox jb;

	public MyJComboBox()
	{
		fr = new JFrame("JComboBox Demo");
		fr.setBounds(100,100, 300, 300);
		fr.setLayout(new FlowLayout());
		fr.setDefaultCloseOperation(fr.EXIT_ON_CLOSE);
		fr.setVisible(true);

		String subjets[] = {"C", "C++", "Java","Python","Web"};

		jb =  new JComboBox(subjets);

		fr.add(jb);
	}

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

Output:



JOptionPane

The ***JOptionPane*** class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box.
These dialog boxes are used to display information or get input from the user.
The ***JOptionPane*** class inherits ***JComponent*** class.

JOptionPane class is used to display four types of dialog boxes

- **MessageDialog** - dialog box that displays a message making it possible to add icons to alert the user.
- **ConfirmDialog** - dialog box that besides sending a message, enables the user to answer a question.
- **InputDialog** - dialog box that besides sending a message, allows entry of a text.
- **OptionDialog** - dialog box that covers the three previous types.

JOptionPane class declaration:

```
public class JOptionPane extends JComponent implements Accessible
```

Constructors:

Constructors	Description
JOptionPane( )	It is used to create a JOptionPane with a test message.
JOptionPane(Object message)	It is used to create an instance of JOptionPane to display a message.
JOptionPane(Object message, int messageType)	It is used to create an instance of JOptionPane to display a message with specified message type and default options.

Methods:

Methods	Description
JDialog createDialog(String title)	It is used to create and return a new parentless JDialog with the specified title.

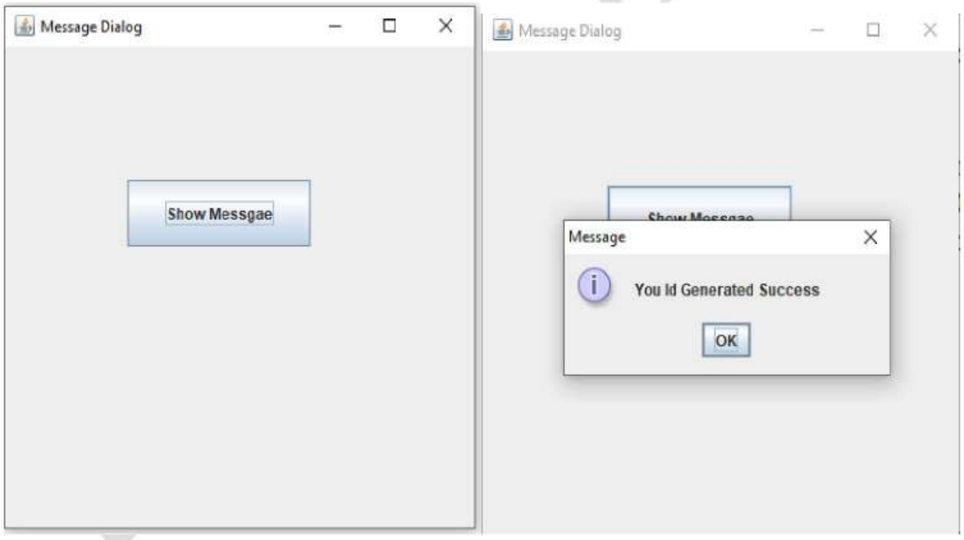
static void showMessageDialog(Component parentComponent, Object message)	It is used to create an information-message dialog titled "Message".
static void showMessageDialog(Component parentComponent, Object message, String title, int messageType)	It is used to create a message dialog with given title and messageType.
static int showConfirmDialog(Component parentComponent, Object message)	It is used to create a dialog with the options Yes, No and Cancel; with the title, Select an Option.
static String showInputDialog(Component parentComponent, Object message)	It is used to show a question-message dialog requesting input from the user parented to parentComponent.
void setInputValue(Object newValue)	It is used to set the input value that was selected or input by the user.

JOptionPane Example1:

1. Message Dialog:

```
package com.kle.bca1;

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

public class MyMessageDialog implements ActionListener
{

	JFrame frm;
	JButton btn;

	public MyMessageDialog()
	{
		frm = new JFrame("Message Dialog");
		frm.setBounds(100, 100, 400, 400);
		frm.setVisible(true);
		frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frm.setLayout(null);

		btn = new JButton("Show Messgae");
		btn.setBounds(100, 100, 150, 50);
		btn.addActionListener(this);

		frm.add(btn);

	}
	public static void main(String[] args)
	{
		new MyMessageDialog();
	}
	public void actionPerformed(ActionEvent arg0)
	{
		JOptionPane.showMessageDialog(frm, "You Id Generated
Success");
	}

}
```

Output:



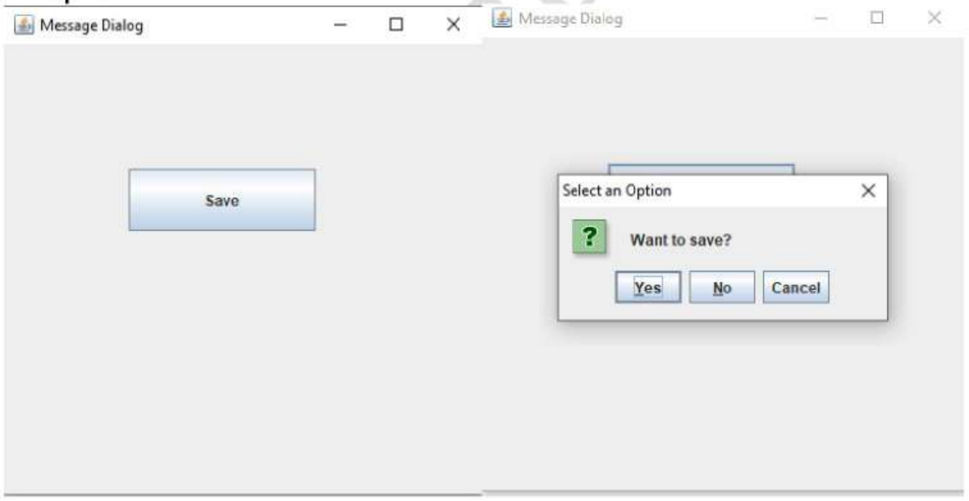
You Id Generated Success

2. Confirm Dialog:

```
package com.kle.bca1;
import java.awt.event.*;
import javax.swing.*;
public class MyConfirmDialog implements ActionListener
{
	JFrame frm;
```

```
        JButton btn;

        public MyConfirmDialog()
        {
                frm = new JFrame("Message Dialog");
                frm.setBounds(100, 100, 400, 400);
                frm.setVisible(true);
                frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frm.setLayout(null);
                btn = new JButton("Save");
                btn.setBounds(100, 100, 150, 50);
                btn.addActionListener(this);
                frm.add(btn);

        }
        public static void main(String[] args)
        {
                new MyConfirmDialog();
        }
        public void actionPerformed(ActionEvent arg0)
        {
                JOptionPane.showConfirmDialog(frm, "Want to save?");
        }

}
```

Output:

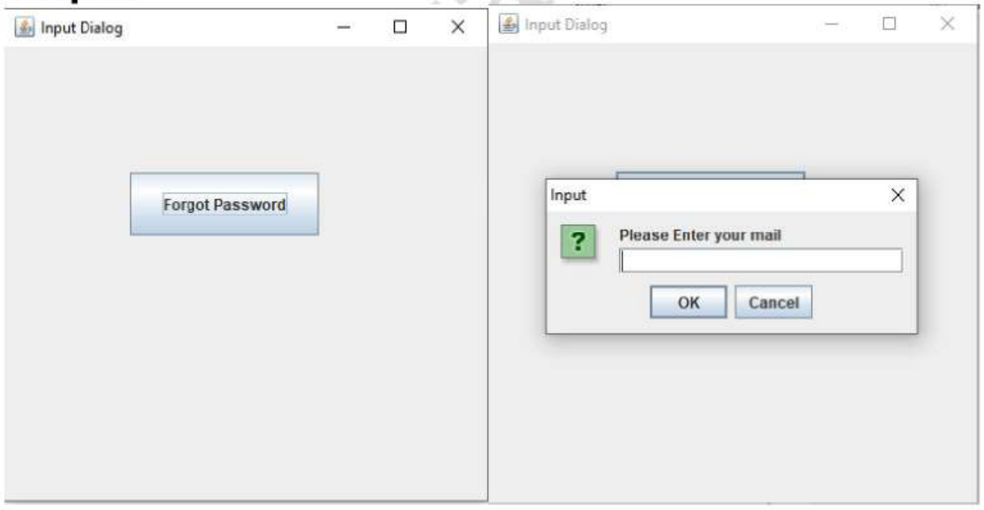


3. Input Dialog:

```
package com.kle.bca1;

import java.awt.event.*;
import javax.swing.*;
public class MyInputDialog implements ActionListener
```

```
{
        JFrame frm;
        JButton btn;

        public MyInputDialog()
        {
                frm = new JFrame("Input Dialog");
                frm.setBounds(100, 100, 400, 400);
                frm.setVisible(true);
                frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frm.setLayout(null);
                btn = new JButton("Forgot Password");
                btn.setBounds(100, 100, 150, 50);
                btn.addActionListener(this);
                frm.add(btn);
        }
        public static void main(String[] args)
        {
                new MyInputDialog();
        }
        public void actionPerformed(ActionEvent arg0)
        {
                JOptionPane.showInputDialog(frm, "Please Enter your mail");
        }

}
```

Output:



LayoutManager:

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers.

- Most LayoutManagers are present in **java.awt** package

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc

1. BorderLayout:

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window.

The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Constructors:

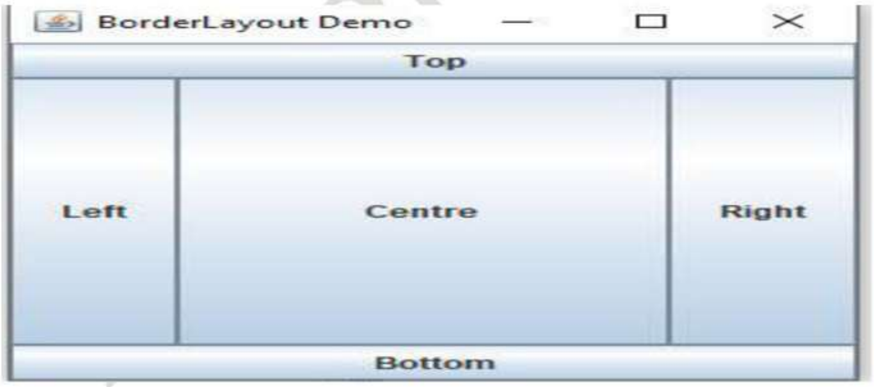
- **BorderLayout():** creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

BorderLayout Example:

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

public class MyBorderLayout1
{
            JFrame fr;
            JButton b1,b2,b3,b4,b5;
```

```
        public MyBorderLayout1()
        {
              fr = new JFrame("BorderLayout Demo");
              fr.setBounds(100,100, 300, 300);
              fr.setDefaultCloseOperation(fr.EXIT_ON_CLOSE);
              fr.setVisible(true);

              b1 = new JButton("Right");
              b2 = new JButton("Left");
              b3 = new JButton("Top");
              b4 = new JButton("Bottom");
              b5 = new JButton("Centre");

              fr.add(b1, BorderLayout.EAST);
              fr.add(b2, BorderLayout.WEST);
              fr.add(b3, BorderLayout.NORTH);
              fr.add(b4, BorderLayout.SOUTH);
              fr.add(b5, BorderLayout.CENTER);

        }
        public static void main(String[] args)
        {
              new MyBorderLayout1();
        }
}
```

Output:


BorderLayout Demo
Top
Left
Centre
Right
Bottom