

Unit IV Swing Part A

Swing

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

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

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

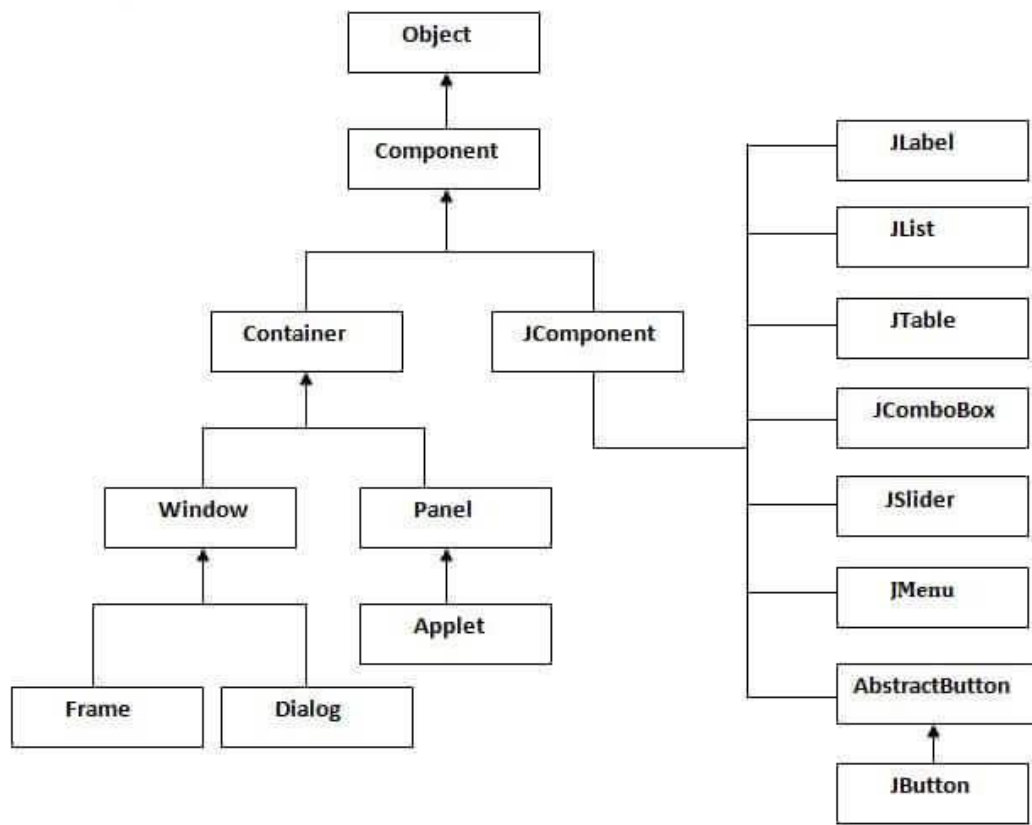
No.	Java AWT	Java Swing
1)	AWT components are platform-dependent .	Java swing components are platform-independent .
2)	AWT components are heavyweight .	Swing components are lightweight .
3)	AWT doesn't support pluggable look and feel .	Swing supports pluggable look and feel .
4)	AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT doesn't follows MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC .

What is JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

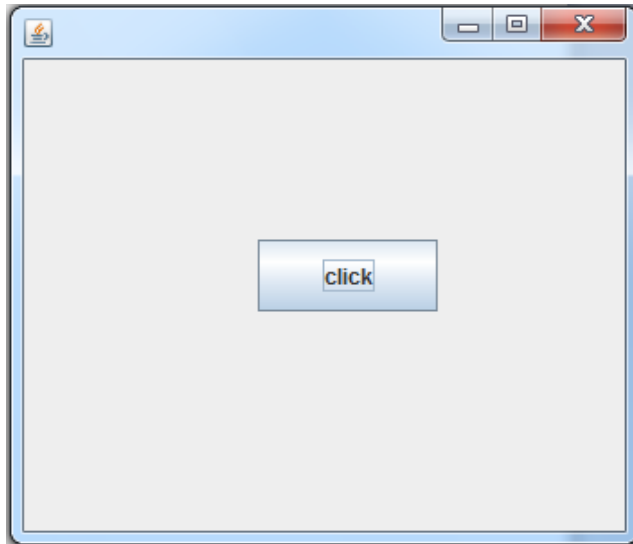
Simple Java Swing Example

```

import javax.swing.*;

public class FirstSwingExample
{
    public static void main(String[] args)
    {
        JFrame f=new JFrame();//creating instance of JFrame
        JButton b=new JButton("click");//creating instance of JButton
        b.setBounds(130,100,100, 40);//x axis, y axis, width, height
        f.add(b);//adding button in JFrame
        f.setSize(350,300);//350 width and 300 height
        f.setLayout(null);//using no layout managers
        f.setVisible(true);//making the frame visible
    }
}
  
```

Output:



Swing by Association inside constructor

```
import javax.swing.*;
public class SwingExample1
{
    JFrame f;
    SwingExample1()
    {
        f=new JFrame();//creating instance of JFrame
        JButton b=new JButton("click");//creating instance of JButton
        b.setBounds(130,100,100, 40);
        f.add(b);//adding button in JFrame
        f.setSize(400,500);//400 width and 500 height
        f.setLayout(null);//using no layout managers
        f.setVisible(true);//making the frame visible
    }
    public static void main(String[] args)
    {
        new SwingExample1();
    }
}
```

Example of Swing by inheritance

```
import javax.swing.*;
public class SwingExample2 extends JFrame//inheriting JFrame
{
    JFrame f;
    SwingExample2()
    {
```

```

        JButton b=new JButton("click");//create button
        b.setBounds(130,100,100, 40);
        add(b);//adding button on frame
        setSize(400,500);
        setLayout(null);
        setVisible(true);
    }
    public static void main(String[] args)
    {
        new SwingExample2();
    }
}

```

Swing Applet

A Swing applet extends **JApplet** rather than **Applet**. **JApplet** is derived from **Applet**. Thus, **JApplet** includes all of the functionality found in **Applet** and adds support for Swing. **JApplet** is a top-level Swing container. Therefore, it includes the various panes described earlier. As a result, all components are added to **JApplet**'s content pane in the same way that components are added to **JFrame**'s content pane.

Example of EventHandling in JApplet:

```

import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
public class EventJApplet extends JApplet implements ActionListener
{
    JButton b;
    JTextField tf;
    public void init()
    {
        tf=new JTextField();
        tf.setBounds(30,40,150,20);
        b=new JButton("Click");
        b.setBounds(80,150,70,40);
        add(b);add(tf);
        b.addActionListener(this);
        setLayout(null);
    }
    public void actionPerformed(ActionEvent e)
    {
        tf.setText("Welcome");
    }
}

```

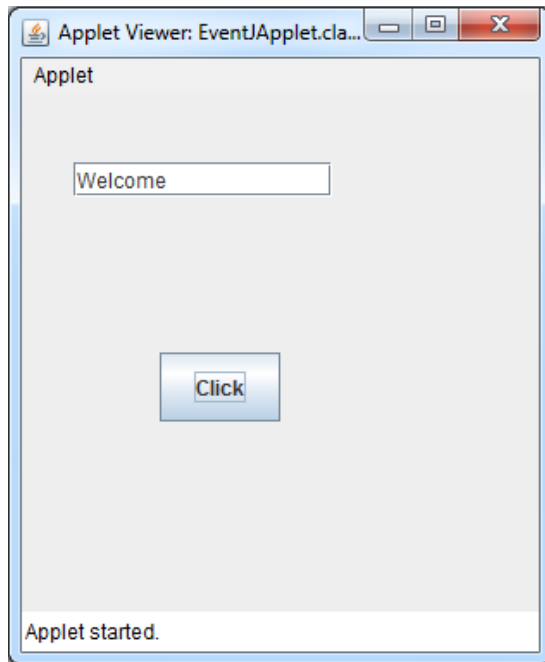
```

}

/*
<applet code="EventJApplet.class" width=300 height=300>
</applet>
*/

```

Output:



TitleBar icon in Java AWT and Swing

The `setIconImage()` method of `Frame` class is used to change the icon of `Frame` or `Window`. It changes the icon which is displayed at the left side of `Frame` or `Window`. The `Toolkit` class is used to get instance of `Image` class in AWT and Swing. `Toolkit` class is the abstract super class of every implementation in the `Abstract Window Toolkit (AWT)`. Subclasses of `Toolkit` are used to bind various components. It inherits `Object` class.

Example to change TitleBar icon in Java Swing

```

import javax.swing.*;
import java.awt.*;
class IconExample
{
    IconExample()
    {
        JFrame f=new JFrame();
        Image icon = Toolkit.getDefaultToolkit().getImage("icon1.png");
        f.setIconImage(icon);
        f.setLayout(null);
    }
}

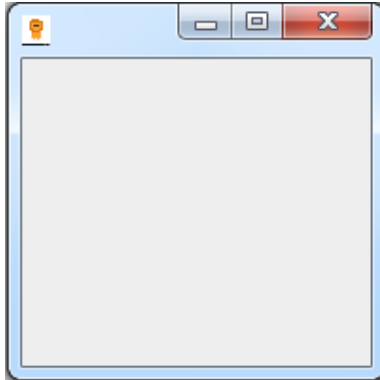
```

```

        f.setSize(200,200);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new IconExample();
    }
}

```

Output:



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.

JButton class declaration

public class JButton extends AbstractButton implements Accessible

Commonly used Constructors:

Constructor	Description
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.

Commonly used Methods of AbstractButton class:

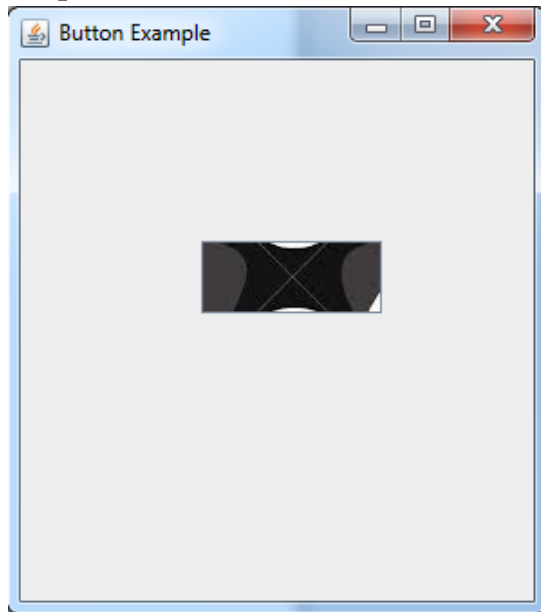
Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	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 <u>action listener</u> to this object.

Example of displaying image on the button:

```
import javax.swing.*;
public class ButtonExample
{
    ButtonExample()
    {
        JFrame f=new JFrame("Button Example");
        JButton b=new JButton(new ImageIcon("icon.jpg"));
        b.setBounds(100,100,100, 40);
        f.add(b);
        f.setSize(300,400);
        f.setLayout(null);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args)
    {
        new ButtonExample();
    }
}
```

Output:



JLabel

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

JLabel class declaration

public class JLabel extends JComponent implements SwingConstants, Accessible

Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

Commonly used Methods:

Methods	Description
String getText()	It returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

Java JTextField

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

JTextField class declaration

public class JTextField extends JTextComponent implements SwingConstants

Commonly used Constructors:

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.

<code>JTextField(int columns)</code>	Creates a new empty TextField with the specified number of columns.
--------------------------------------	---

Commonly used Methods:

Methods	Description
<code>void addActionListener(ActionListener l)</code>	It is used to add the specified action listener to receive action events from this textfield.
<code>Action getAction()</code>	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
<code>void setFont(Font f)</code>	It is used to set the current font.
<code>void removeActionListener(ActionListener l)</code>	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

JTextField Example with ActionListener

```

import javax.swing.*;
import java.awt.event.*;
public class TextFieldExample implements ActionListener
{
    JTextField tf1,tf2,tf3;
    JButton b1,b2;
    TextFieldExample()
    {
        JFrame f= new JFrame();
        tf1=new JTextField();
        tf1.setBounds(50,50,150,20);
        tf2=new JTextField();
        tf2.setBounds(50,100,150,20);
        tf3=new JTextField();
        tf3.setBounds(50,150,150,20);
        tf3.setEditable(false);
        b1=new JButton("+");
        b1.setBounds(50,200,50,50);
        b2=new JButton("-");
        b2.setBounds(120,200,50,50);
        b1.addActionListener(this);
        b2.addActionListener(this);
        f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}

```

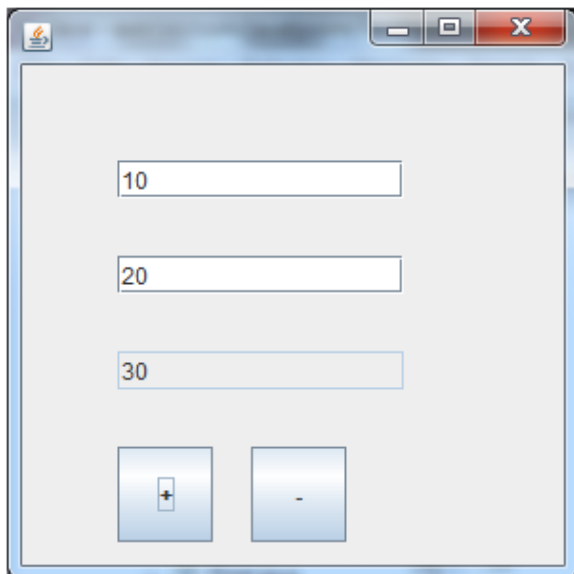
```

public void actionPerformed(ActionEvent e)
{
    String s1=tf1.getText();
    String s2=tf2.getText();
    int a=Integer.parseInt(s1);
    int b=Integer.parseInt(s2);
    int c=0;
    if(e.getSource()==b1)
    {
        c=a+b;
    }
    else if(e.getSource()==b2)
    {
        c=a-b;
    }
    String result=String.valueOf(c);
    tf3.setText(result);
}
public static void main(String[] args)
{
    new TextFieldExample();
}
}

```

Output:

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



JTextArea

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

JTextArea class declaration

public class JTextArea extends JTextComponent

Commonly used Constructors:

Constructor	Description
JTextArea()	Creates a text area that displays no text initially.
JTextArea(String s)	Creates a text area that displays specified text initially.
JTextArea(int row, int column)	Creates a text area with the specified number of rows and columns that displays no text initially.
JTextArea(String s, int row, int column)	Creates a text area with the specified number of rows and columns that displays specified text.

Commonly used Methods:

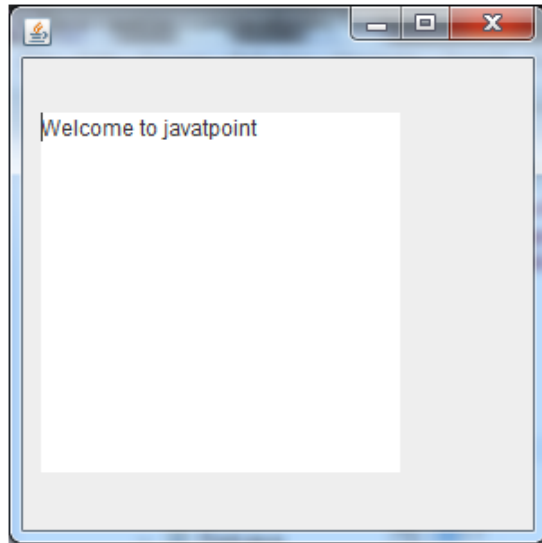
Methods	Description
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

JTextArea Example

```
import javax.swing.*;
public class TextAreaExample
{
    TextAreaExample()
    {
        JFrame f= new JFrame();
        JTextArea area=new JTextArea("Welcome to javatpoint");
        area.setBounds(10,30, 200,200);
        f.add(area);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new TextAreaExample();
    }
}
```

```
}
}
```

Output:



JPasswordField

The object of a JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text. It inherits JTextField class.

JPasswordField class declaration

```
public class JPasswordField extends JTextField
```

Commonly used Constructors:

Constructor	Description
JPasswordField()	Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
JPasswordField(int columns)	Constructs a new empty JPasswordField with the specified number of columns.
JPasswordField(String text)	Constructs a new JPasswordField initialized with the specified text.
JPasswordField(String text, int columns)	

JPasswordField Example with ActionListener

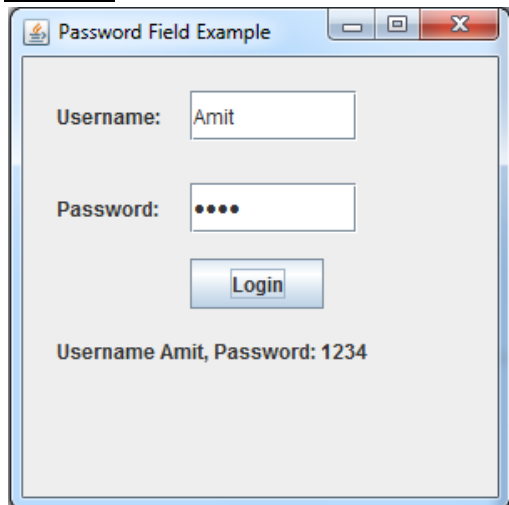
```
import javax.swing.*;
import java.awt.event.*;
public class PasswordFieldExample
{
    public static void main(String[] args)
    {
```

```

JFrame f=new JFrame("Password Field Example");
final JLabel label = new JLabel();
label.setBounds(20,150, 200,50);
final JPasswordField value = new JPasswordField();
value.setBounds(100,75,100,30);
JLabel l1=new JLabel("Username:");
l1.setBounds(20,20, 80,30);
JLabel l2=new JLabel("Password:");
l2.setBounds(20,75, 80,30);
JButton b = new JButton("Login");
b.setBounds(100,120, 80,30);
final JTextField text = new JTextField();
text.setBounds(100,20, 100,30);
f.add(value); f.add(l1); f.add(label); f.add(l2); f.add(b); f.add(text);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
b.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        String data = "Username " + text.getText();
        data += ", Password: " + new String(value.getPassword());
        label.setText(data);
    }
});
}
}

```

Output:



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

Commonly used Constructors:

Constructor	Description
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.

Commonly used Methods:

Methods	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.
protected String paramString()	It returns a <u>string</u> representation of this JCheckBox.

JCheckBox Example: Food Order

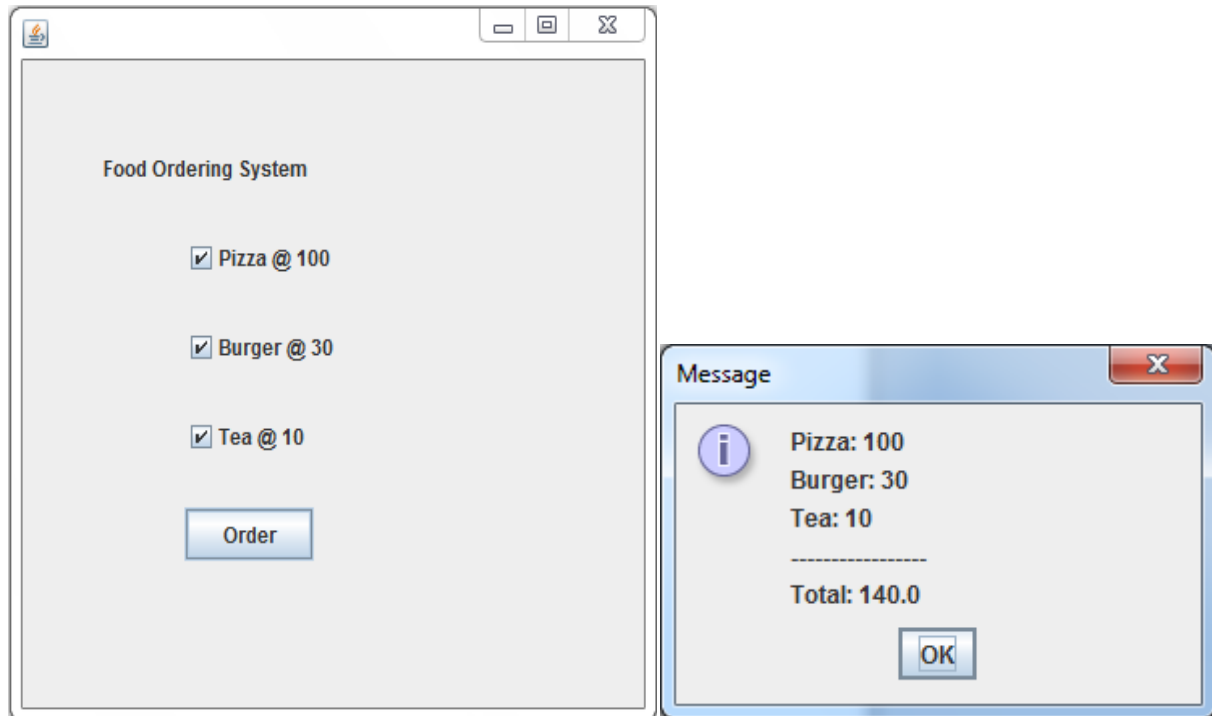
```
import javax.swing.*;
import java.awt.event.*;
public class CheckBoxExample extends JFrame implements ActionListener
{
    JLabel l;
    JCheckBox cb1,cb2,cb3;
    JButton b;
    CheckBoxExample()
    {
        l=new JLabel("Food Ordering System");
        l.setBounds(50,50,300,20);
        cb1=new JCheckBox("Pizza @ 100");
        cb1.setBounds(100,100,150,20);
        cb2=new JCheckBox("Burger @ 30");
        cb2.setBounds(100,150,150,20);
        cb3=new JCheckBox("Tea @ 10");
```

```

        cb3.setBounds(100,200,150,20);
        b=new JButton("Order");
        b.setBounds(100,250,80,30);
        b.addActionListener(this);
        add(l);add(cb1);add(cb2);add(cb3);add(b);
        setSize(400,400);
        setLayout(null);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
    public void actionPerformed(ActionEvent e)
    {
        float amount=0;
        String msg="";
        if(cb1.isSelected())
        {
            amount+=100;
            msg="Pizza: 100\n";
        }
        if(cb2.isSelected())
        {
            amount+=30;
            msg+="Burger: 30\n";
        }
        if(cb3.isSelected())
        {
            amount+=10;
            msg+="Tea: 10\n";
        }
        msg+="-----\n";
        JOptionPane.showMessageDialog(this,msg+"Total: "+amount);
    }
    public static void main(String[] args)
    {
        new CheckBoxExample();
    }
}

```

Output:



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.

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

JRadioButton class declaration

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

Commonly used Constructors:

Constructor	Description
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.

Commonly used 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.

<code>void setMnemonic(int a)</code>	It is used to set the mnemonic on the button.
<code>void addActionListener(ActionListener a)</code>	It is used to add the action listener to this object.

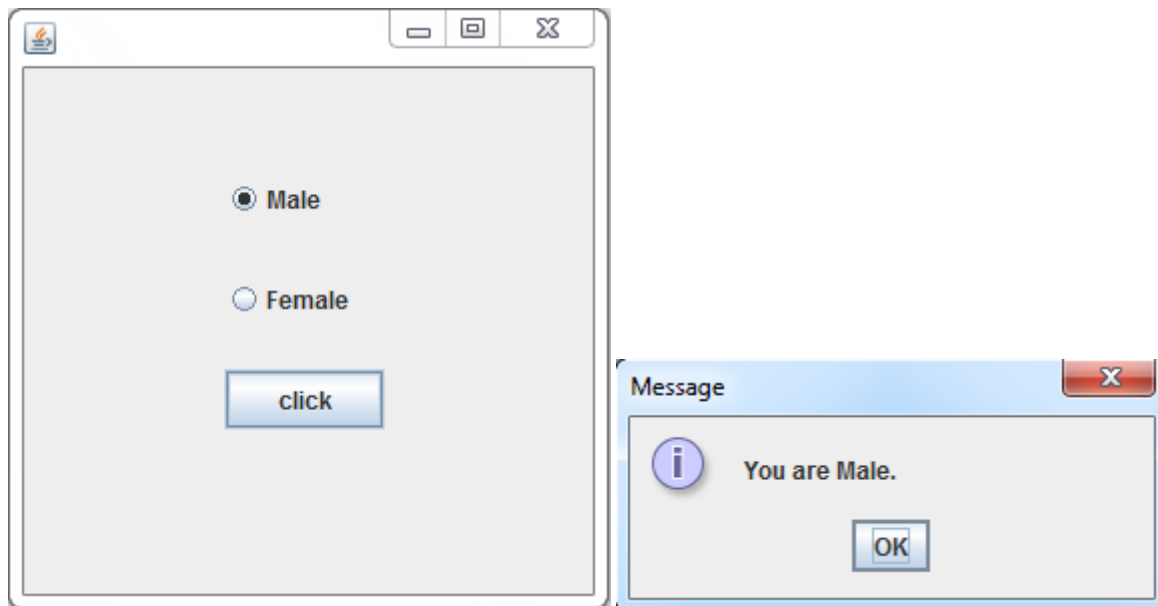
JRadioButton Example with ActionListener

```

import javax.swing.*;
import java.awt.event.*;
class RadioButtonExample extends JFrame implements ActionListener
{
    JRadioButton rb1,rb2;
    JButton b;
    RadioButtonExample()
    {
        rb1=new JRadioButton("Male");
        rb1.setBounds(100,50,100,30);
        rb2=new JRadioButton("Female");
        rb2.setBounds(100,100,100,30);
        ButtonGroup bg=new ButtonGroup();
        bg.add(rb1);bg.add(rb2);
        b=new JButton("click");
        b.setBounds(100,150,80,30);
        b.addActionListener(this);
        add(rb1);add(rb2);add(b);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e)
    {
        if(rb1.isSelected())
        {
            JOptionPane.showMessageDialog(this,"You are Male.");
        }
        if(rb2.isSelected())
        {
            JOptionPane.showMessageDialog(this,"You are Female.");
        }
    }
    public static void main(String args[])
    {
        new RadioButtonExample();
    }
}

```

Output:



JComboBox

The object of Choice 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

Commonly used 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 <u>array</u> .
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified <u>Vector</u> .

Commonly used 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 <u>ActionListener</u> .

```
void addItemListener(ItemListener i)
```

```
It is used to add the ItemListener.
```

JComboBox Example

```
import javax.swing.*;
```

```
public class ComboBoxExample
```

```
{
```

```
    JFrame f;
```

```
    ComboBoxExample()
```

```
    {
```

```
        f=new JFrame("ComboBox Example");
```

```
        String country[]={"India","Aus","U.S.A","England","Newzealand"};
```

```
        JComboBox cb=new JComboBox(country);
```

```
        cb.setBounds(50, 50,90,20);
```

```
        f.add(cb);
```

```
        f.setLayout(null);
```

```
        f.setSize(400,500);
```

```
        f.setVisible(true);
```

```
    }
```

```
    public static void main(String[] args)
```

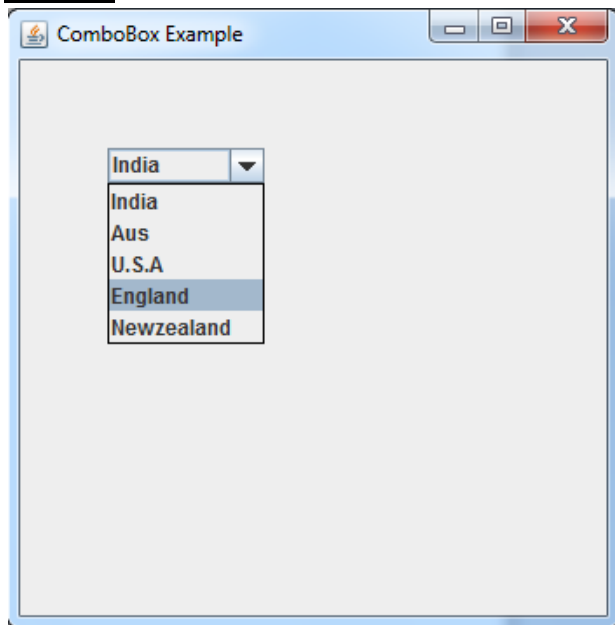
```
    {
```

```
        new ComboBoxExample();
```

```
    }
```

```
}
```

Output:



JComboBox Example with ActionListener

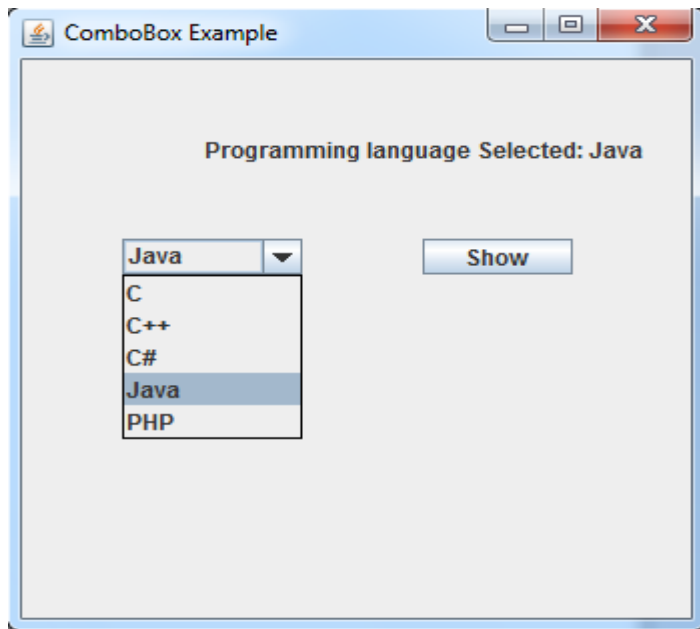
```
import javax.swing.*;
```

```

import java.awt.event.*;
public class ComboBoxExample1
{
    JFrame f;
    ComboBoxExample1()
    {
        f=new JFrame("ComboBox Example");
        final JLabel label = new JLabel();
        label.setHorizontalAlignment(JLabel.CENTER);
        label.setSize(400,100);
        JButton b=new JButton("Show");
        b.setBounds(200,100,75,20);
        String languages[]={"C","C++","C#","Java","PHP"};
        final JComboBox cb=new JComboBox(languages);
        cb.setBounds(50, 100,90,20);
        f.add(cb); f.add(label); f.add(b);
        f.setLayout(null);
        f.setSize(350,350);
        f.setVisible(true);
        b.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                String data = "Programming language Selected: " +
cb.getItemAt(cb.getSelectedIndex());
                label.setText(data);
            }
        });
    }
    public static void main(String[] args)
    {
        new ComboBoxExample1();
    }
}

```

Output:



Java JTable

The JTable class is used to display data in tabular form. It is composed of rows and columns.

Commonly used Constructors:

Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

JTable Example

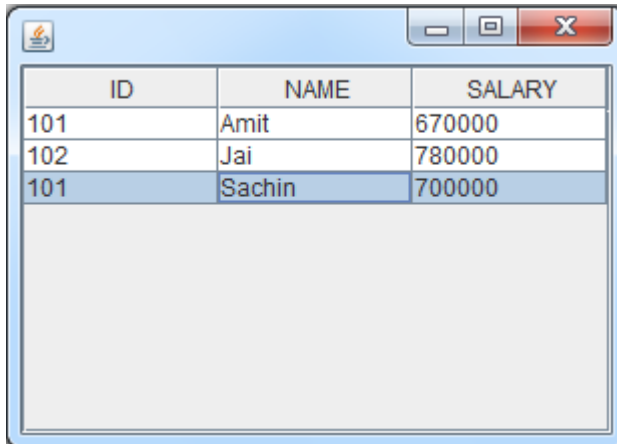
```
import javax.swing.*;
public class TableExample
{
    JFrame f;
    TableExample()
    {
        f=new JFrame();
        String data[][]={ {"101","Amit","670000"},
                           {"102","Jai","780000"},
                           {"101","Sachin","700000"} };
        String column[]={"ID","NAME","SALARY"};
        JTable jt=new JTable(data,column);
        jt.setBounds(30,40,200,300);
        JScrollPane sp=new JScrollPane(jt);
        f.add(sp);
        f.setSize(300,400);
        f.setVisible(true);
    }
}
```

```

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

```

Output:



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

JTable Example with ListSelectionListener

```

import javax.swing.*;
import javax.swing.event.*;
public class TableExample1
{
    public static void main(String[] a)
    {
        JFrame f = new JFrame("Table Example");
        String data[][]={ {"101","Amit","670000"},
            {"102","Jai","780000"},
            {"101","Sachin","700000"} };
        String column[]={"ID","NAME","SALARY"};
        final JTable jt=new JTable(data,column);
        jt.setCellSelectionEnabled(true);
        ListSelectionModel select= jt.getSelectionModel();
        select.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        select.addListSelectionListener(new ListSelectionListener()
        {
            public void valueChanged(ListSelectionEvent e)
            {
                String Data = null;
                int[] row = jt.getSelectedRows();
                int[] columns = jt.getSelectedColumns();
                for (int i = 0; i < row.length; i++)
                {

```

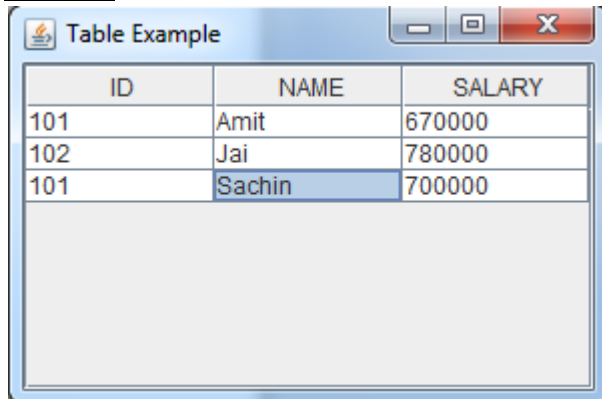
```

        for (int j = 0; j < columns.length; j++)
        {
            Data = (String) jt.getValueAt(row[i],
columns[j]);
        }
        System.out.println("Table element selected is: " + Data);
    }
}

});
JScrollPane sp=new JScrollPane(jt);
f.add(sp);
f.setSize(300, 200);
f.setVisible(true);
}
}

```

Output:



Java 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

public class JList extends JComponent implements Scrollable, Accessible

Commonly used 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.

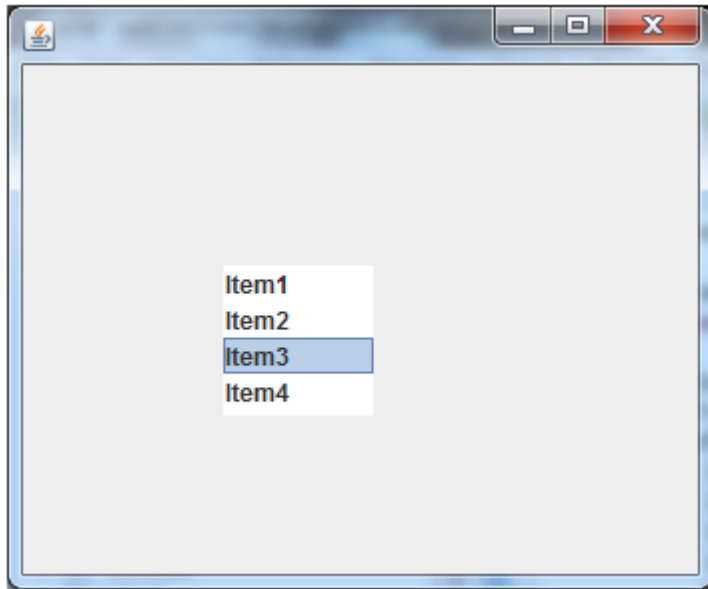
Commonly used Methods:

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 javax.swing.*;
public class ListExample
{
    ListExample()
    {
        JFrame f= new JFrame();
        DefaultListModel<String> l1 = new DefaultListModel<>();
        l1.addElement("Item1");
        l1.addElement("Item2");
        l1.addElement("Item3");
        l1.addElement("Item4");
        JList<String> list = new JList<>(l1);
        list.setBounds(100,100, 75,75);
        f.add(list);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new ListExample();
    }
}
```

Output:



AKV