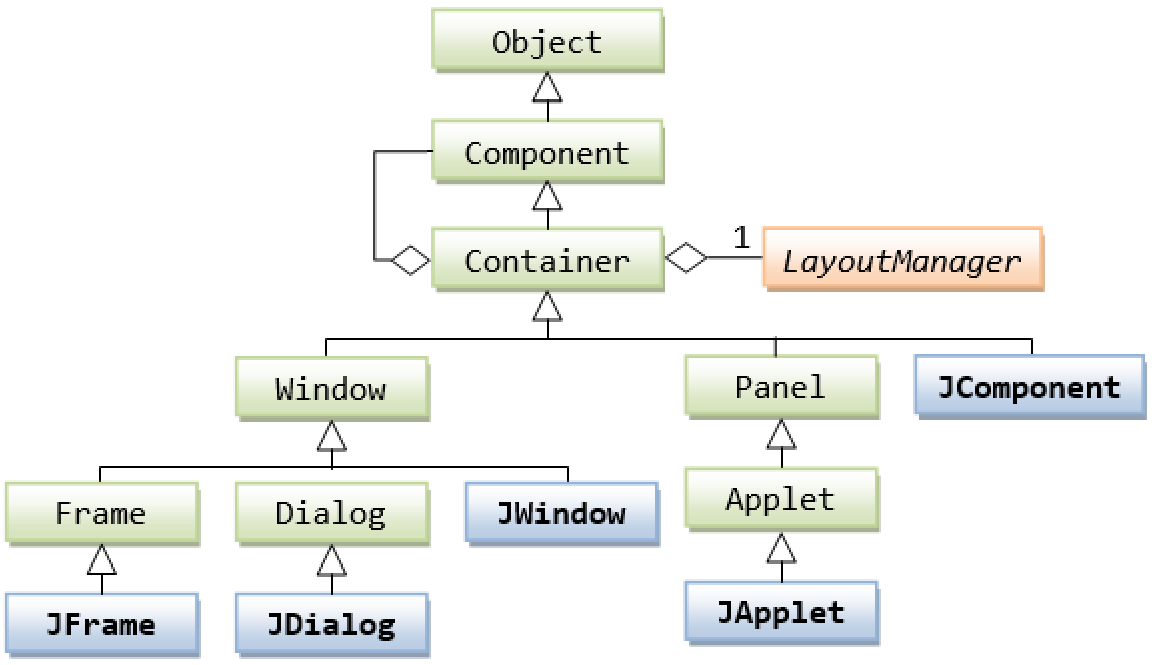
Java Swing

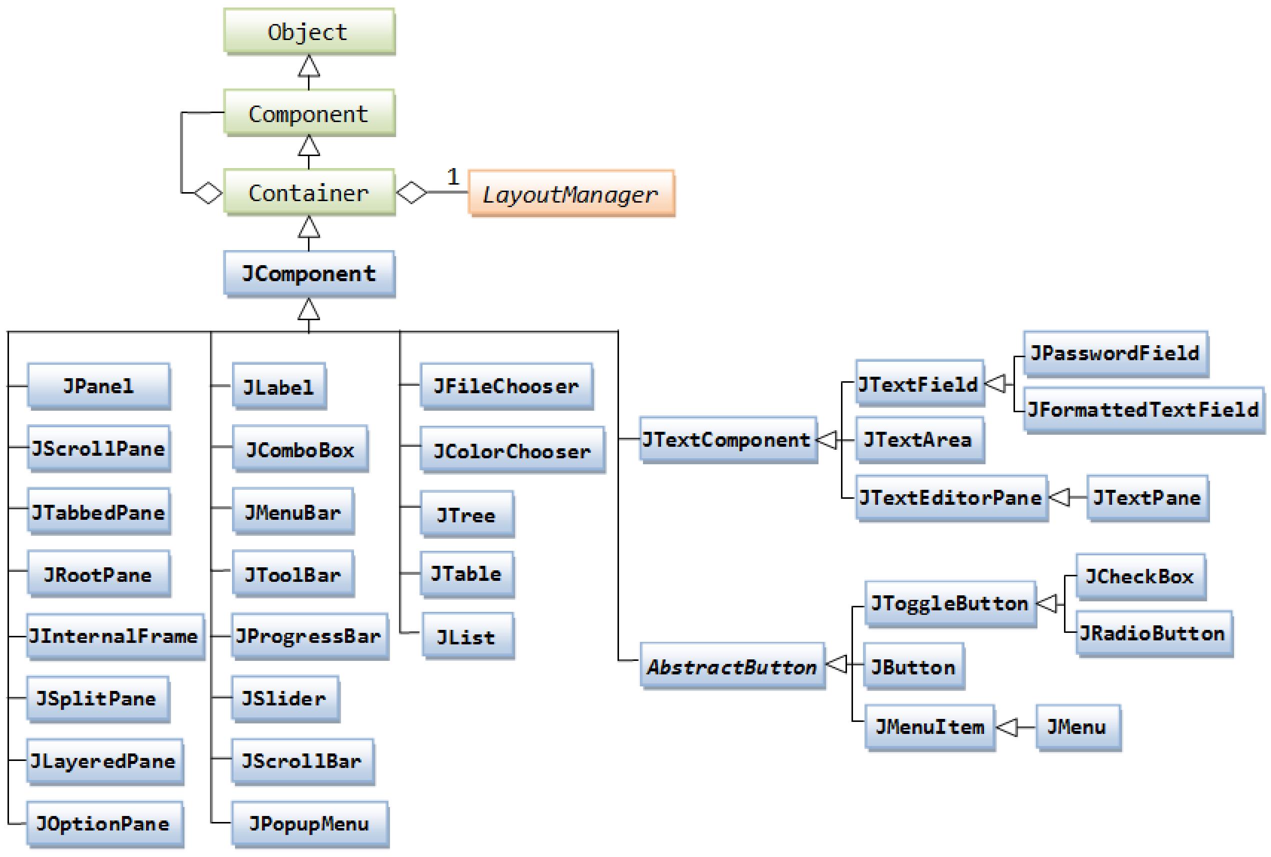
Java Swing is a part of **Java Foundation Classes** (JFC) that is *used to* ***create window-based applications***.
It is built on the top of AWT (Abstract Windowing Toolkit) API and **entirely written in java**.
Swing is a **set of classes** that provides **more powerful and flexible components** than are possible with the **AWT**.
In addition to the familiar components, such as buttons, check boxes, and labels, Swing supplies several exciting additions, including **tabbed panes, scroll panes, trees, and tables.**
Even familiar components such as **buttons** have more **capabilities** in **Swing**. For example, a **button** may have both an **image** and **a text string** associated with it.
Also, the **image** can be **changed** as the **state of the button changes.**
Unlike AWT components, **Swing components** are **not implemented** by **platform-specific code**.
Instead, they are written entirely in Java and, therefore, **are platform-independent**.
The term **lightweight** is used to describe such elements.
The **javax.swing** package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

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





Commonly used Methods of Component class

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Containers

There are two types of containers :

Top-Level Containers

A top-level container, as the name suggests, lies at the top of the containment hierarchy. The top-level containers are **JFrame**, **JApplet**, and **JDialog**. These containers **do not inherit JComponent class** but inherit the **AWT classes' Component and Container**. These containers are **heavyweight** components. The most commonly used containers are **JFrame** and **JApplet**.

Lightweight Containers

Lightweight containers lie next to the top-level containers in the containment hierarchy. **They inherit JComponent.** One of the examples of lightweight container is **JPanel**.

As lightweight container can be contained within another container, they can be used to organize and manage groups of related components.

Java Swing Examples

There are two ways to create a frame:

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

Simple Java Swing Example

We are creating one button and adding it on the JFrame object inside the main() method.

```
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(400,500);                 //400 width and 500 height
f.setLayout(null);            //using no layout managers
f.setVisible(true);           //making the frame visible
}
}
```

Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

```
import javax.swing.*;
public class Simple2 extends JFrame   //inheriting JFrame
{
Simple2()
{
JLabel lb = new JLabel("First Program of Swing");
JButton b=new JButton("click");                        //create button
lb.setBounds(130,50,200,40);
b.setBounds(130,100,100, 40);
add(b);//adding button on frame
add(lb);
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args)
{
Simple2 f = new Simple2();
}
}
```

Java 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.

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 addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JButton Example

```
import javax.swing.*;
public class ButtonExample1
{
public static void main(String[] args)
{
   JFrame f=new JFrame("Button Example");
   JButton b=new JButton("Click Here");
   b.setBounds(50,100,95,30);
   f.add(b);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
}
```

Java JButton Example with ActionListener

```
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample extends JFrame implements ActionListener
{
   JTextField tf;
   public ButtonExample()
   {
   tf=new JTextField();
   tf.setBounds(50,50,200,20);
   JButton b=new JButton("Click Here");
   b.setBounds(50,100,95,30);
   b.addActionListener(this);
   add(b);
   add(tf);
   }
   public void actionPerformed(ActionEvent e)
      {
         tf.setText("Welcome to Arrow Academy");
      }

public static void main(String[] args)
{
   ButtonExample f = new ButtonExample();
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
}
```

Example of displaying image on the button:

```
import javax.swing.*;
public class ButtonExample
{
public static void main(String[] args)
{
JFrame f=new JFrame("Button Example");
ImageIcon i = new ImageIcon("D:\\icon.png");
JButton b=new JButton(i);
b.setBounds(100,100,150, 40);
f.add(b);
f.setSize(500,400);
f.setLayout(null);
f.setVisible(true);
// f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

Java 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.

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. (JLabel.LEFT, JLabel.CENTER, JLabel.RIGHT)

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 JLabel Example

```
import javax.swing.*;
class LabelExample
{
public static void main(String args[])
   {
   JFrame f= new JFrame("Label Example");
   JLabel l1,l2;
   l1=new JLabel("First Label.");
   l1.setBounds(50,50, 100,30);
   l2=new JLabel("Second Label.");
   l2.setBounds(50,100, 100,30);
   f.add(l1);
   f.add(l2);
   f.setSize(300,300);
   f.setLayout(null);
   f.setVisible(true);
   }
   }
```

Create a JLabel with an image icon

```
import javax.swing.*;
class LabelExample
{
public static void main(String args[])
   {
   JFrame f= new JFrame("Label Example");
   ImageIcon i = new ImageIcon("D:\\icon.png");
   JLabel l1,l2;
   l1=new JLabel(i);
   l1.setBounds(50,50,150,30);
   l2=new JLabel("Second Label.");
   l2.setBounds(50,100, 100,30);
   f.add(l1); f.add(l2);
   f.setSize(300,300);
   f.setLayout(null);
   f.setVisible(true);
   }
   }
```

JTextField

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

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.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

Commonly used Methods:

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

Java JTextField Example

```
import javax.swing.*;
class TextFieldExample
{
public static void main(String args[])
   {
   JFrame f= new JFrame("TextField Example");
   JTextField t1,t2;
   t1=new JTextField("Welcome to Arrow World.");
   t1.setBounds(50,100, 200,30);
   t2=new JTextField("Swing Tutorial");
   t2.setBounds(50,150, 200,30);
   f.add(t1);
   f.add(t2);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
   }
}
```

Java JTextField Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
class Sample extends JFrame implements ActionListener
{
   JTextField tf1,tf2,tf3;
    JButton b1,b2,b3,b4;
   Sample()
   {

      tf1 = new JTextField();
      tf2 = new JTextField();
      tf3 = new JTextField();
      tf3.setEditable(false);
      b1 = new JButton("+");
      b2 = new JButton("-");
      b3 = new JButton("*");
      b4 = new JButton("/");

      tf1.setBounds(50,50,100,30);
      tf2.setBounds(50,100,100,30);
      tf3.setBounds(50,150,100,30);

      b1.setBounds(50,200,50,50);
      b2.setBounds(110,200,50,50);
      b3.setBounds(170,200,50,50);
      b4.setBounds(230,200,50,50);

      b1.addActionListener(this);
      b2.addActionListener(this);
      b3.addActionListener(this);
      b4.addActionListener(this);

      add(tf1);
      add(tf2);
      add(tf3);
      add(b1);
      add(b2);
      add(b3);
      add(b4);
   }

   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;
        }
         else if(e.getSource()==b3)
        {
           c=a*b;
        }
         else if(e.getSource()==b4)
        {
           c=a/b;
        }

        String str = String.valueOf(c);
        tf3.setText(str);
    }
public static void main(String[] args)
{
    Sample fr = new Sample();
    fr.setSize(500,500);
    fr.setLayout(null);
    fr.setVisible(true);
}
}
```

Java 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

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 append(String s)	It is used to append the given text to the end of the document.

Java JTextArea Example

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

Java JTextArea Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class TextAreaExample extends JFrame implements ActionListener
{
JTextArea area;
JLabel l1,l2;
JButton b;
TextAreaExample()
{
   l1=new JLabel();
   l1.setBounds(50,25,100,30);
   l2=new JLabel();
   l2.setBounds(160,25,100,30);
   area=new JTextArea();
   area.setBounds(20,75,250,200);
   b=new JButton("Count Words");
   b.setBounds(100,300,120,30);
   b.addActionListener(this);
   add(l1);
   add(l2);
   add(area);
   add(b);
}
public void actionPerformed(ActionEvent e)
{
  String text=area.getText();
   String words[]=text.split("\\s");
   l1.setText("Words: "+words.length);
   l2.setText("Characters: "+text.length());
}
public static void main(String[] args)
{
  TextAreaExample t = new TextAreaExample();
   t.setSize(450,450);
   t.setLayout(null);
   t.setVisible(true);
}
}
```

Java 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.

Constructor	Description
JPasswordField()	Constructs a new JPasswordField, with a default document, and null starting text string
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)	Construct a new JPasswordField initialized with the specified text and columns.

Example:

```
import javax.swing.*;
public class PasswordFieldExample
{
   public static void main(String[] args)
   {
      JFrame f=new JFrame("Password Field Example");
      JPasswordField value = new JPasswordField();
      JLabel l1=new JLabel("Password:");
      l1.setBounds(20,100, 80,30);
      value.setBounds(100,100,100,30);
         f.add(value);
         f.add(l1);
         f.setSize(300,300);
         f.setLayout(null);
         f.setVisible(true);
   }
}
```

```
import javax.swing.*;
import java.awt.event.*;
public class PasswordFieldExample extends JFrame implements ActionListener
   {
        JLabel label;
        JTextField text;
         JPasswordField pass;
PasswordFieldExample ()
{
  label = new JLabel();
    label.setBounds(20,150,300,50);
       JLabel l1=new JLabel("Username:");
       l1.setBounds(20,20, 80,30);
      text= new JTextField();
       text.setBounds(100,20, 100,30);

       JLabel l2=new JLabel("Password:");
       l2.setBounds(20,75, 80,30);
       pass = new JPasswordField();
       pass.setBounds(100,75,100,30);

       JButton b = new JButton("Login");
       b.setBounds(100,120, 80,30);

              add(pass);
              add(l1);
              add(label);
              add(l2);
              add(b);
              add(text);
              b.addActionListener(this);
            }
              public void actionPerformed(ActionEvent e)
              {
                String data1 = text.getText();
                String data2 = new String (pass.getPassword());
                String data3 = "Username :" +data1+ ", Password :"+data2;
                label.setText(data3);
              }

     public static void main(String[] args)
     {
     PasswordFieldExample f = new PasswordFieldExample();
             f.setSize(300,300);
              f.setLayout(null);
              f.setVisible(true);

 }
}
```

Java 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.

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.

Example 1:

```
import javax.swing.*;
public class CheckBoxExample
{
    public static void main(String args[])
    {
      JFrame f= new JFrame("CheckBox Example");
      JCheckBox checkBox1 = new JCheckBox("C++");
      checkBox1.setBounds(100,100,100,50);
      JCheckBox checkBox2 = new JCheckBox("Java", true);
      checkBox2.setBounds(100,150,100,50);
      f.add(checkBox1);
      f.add(checkBox2);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
  }
```

Java 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.

Method:-
static void showMessageDialog(Component parentComponent, Object message)
It is used to create an information-message dialog titled "Message".

Java JCheckBox Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
public class CheckBoxExample extends JFrame implements ActionListener{
  JLabel l;
  JCheckBox cb1,cb2,cb3;
  JButton b;
  CheckBoxExample f ;
  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);
     }
  public void actionPerformed(ActionEvent e)
{
    float amount=0;
    String msg="";
    if(cb1.isSelected())
       {
      amount+=100;              // amount = 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(f,msg+"Total: "+amount);
    }
    public static void main(String[] args)
    {
      CheckBoxExample f = new CheckBoxExample();
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
  }
```

Java 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.

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.

Java JRadioButton Example

```
import javax.swing.*;
public class RadioButtonExample
{
public static void main(String[] args)
{
JFrame f=new JFrame();
JRadioButton r1=new JRadioButton("A) Male");
JRadioButton r2=new JRadioButton("B) Female");
r1.setBounds(75,50,100,30);
r2.setBounds(75,100,100,30);
ButtonGroup bg=new ButtonGroup();
bg.add(r1);
bg.add(r2);
f.add(r1);
f.add(r2);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
}
```

Java JRadioButton Example with ActionListener

```
import javax.swing.*;
import java.awt.event.*;
class RadioButtonExample extends JFrame implements ActionListener
{
        JRadioButton rb1,rb2,rb3;
        JButton b1;
        RadioButtonExample f;
        RadioButtonExample()
        {
        rb1 = new JRadioButton("Male",true);
        rb2 = new JRadioButton("Female",true);
        rb3 = new JRadioButton("Other",true);

        b1 = new JButton("Submit");

        ButtonGroup bg = new ButtonGroup();
        bg.add(rb1);
        bg.add(rb2);
        bg.add(rb3);

        rb1.setBounds(50,50,100,50);
        rb2.setBounds(50,100,100,50);
        rb3.setBounds(50,150,100,50);
        b1.setBounds(50,220,100,50);
```

```
		b1.addActionListener(this);
		add(rb1);
		add(rb2);
		add(rb3);
		add(b1);
		}

		public void actionPerformed(ActionEvent e)
		{
			if(rb1.isSelected())
			{
				JOptionPane.showMessageDialog(f,"You have Selected male");
			}
			if(rb2.isSelected())
			{
				JOptionPane.showMessageDialog(f,"You have Selected female");
			}
			if(rb3.isSelected())
			{
				JOptionPane.showMessageDialog(f,"You have Selected other");
			}
		}

		public static void main(String[] args)
		{
		RadioButtonExample f = new RadioButtonExample();
		f.setSize(500,500);
		f.setLayout(null);
		f.setVisible(true);
		}
	}
```

Java JComboBox

JComboBox is a part of Java Swing package. JComboBox inherits JComponent class . JComboBox shows a popup menu that shows a list and the user can select a option from that specified list .

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

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.

Example 1:

```
import javax.swing.*;
public class ComboBoxExample
{
public static void main(String[] args)
{
   JFrame 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);
}   }
```

Example 2

```
import javax.swing.*;
import java.awt.event.*;
public class ComboBoxExample extends JFrame implements ActionListener
{
   JComboBox cb;
   JLabel label;
ComboBoxExample()
{
   label = new JLabel();
   label.setBounds(50,50,400,30);
   JButton b=new JButton("Show");
   b.setBounds(200,100,75,20);
   String languages[]={"C","C++","C#","Java","PHP"};
   cb=new JComboBox(languages);
   cb.setBounds(20,100,90,20);
   add(cb);
   add(label);
   add(b);
   b.addActionListener(this);
}
public void actionPerformed(ActionEvent e)
{
String data = "Programming language Selected: " + cb.getItemAt(cb.getSelectedIndex());
label.setText(data);
}
public static void main(String[] args)
{
  ComboBoxExample f = new ComboBoxExample();
   f.setLayout(null);
   f.setSize(350,350);
   f.setVisible(true);
}
}
```

Tabbed Panes

A tabbed pane is a component that appears as a group of folders in a file cabinet. **Each folder** has a **title**. When a user selects a folder, its contents become visible. Only one of the folders may be selected at a time. Tabbed panes are commonly used for **setting configuration options**.
Tabbed panes are encapsulated by the **JTabbedPane** class, which extends **JComponent.**
There are three constructors of JtabbedPane

1. JTabbedPane ( )
The constructor JTabbedPane ( ) creates an **empty TabbedPane**. When we use the constructor JTabbedPane ( ), it creates the pane without any specification for its placement. So the tab is placed on its default place that is TOP as discussed above using JTabbedPane.TOP.

2. JTabbedPane (int tabPlacement)
This constructor creates an empty TabbedPane. It provides the privilege to decide the direction or place for the pane. The placement is done when the direction is specified using JTabbedPane.TOP, JTabbedPane.BOTTOM, JTabbedPane.LEFT, or JTabbedPane.RIGHT.

3. JTabbedPane (int tabPlacement, int tabLayoutPolicy)
An empty tabbed pane is created when this constructor is used. This provides the privilege to decide the placement for the tab-pane the same as the JTabbedPane (int tabPlacement) constructor.
It also lets the programmer decide the tab layout policy. This allows the programmer to control how the tabs will be displayed.
The tab layout policy is either set to **JTabbedPane.SCROLL_TAB_LAYOUT** or **JTabbedPane.WRAP_TAB_LAYOUT**. By default, the policy is set to **WRAP_TAB_LAYOUT**.

With the policy set to SCROLL_TAB_LAYOUT, the tabs become scroll-able and a button for scrolling the tabs, left-right or up-down, is displayed in your tabbed pane.

Example:
```
import javax.swing.*;
class AddJTabbedPane
{
  public static void main(String[] args)
  {
  JTabbedPane tp;
  JLabel lab1, lab2, lab3, lab4, lab5, lab6, lab7, lab8;
  JPanel fruit, vegetable;
  JFrame frame=new JFrame("JTabbedPane Example");
    fruit = new JPanel();
    lab1=new JLabel("Apple");
    lab2=new JLabel("Orange");
    lab3=new JLabel("Papaya");
    lab4=new JLabel("Pine Apple");
    fruit.add(lab1);
    fruit.add(lab2);
```

```
        fruit.add(lab3);
        fruit.add(lab4);

        vegetable = new JPanel();
        lab5=new JLabel("Cauliflower");
        lab6=new JLabel("Brinjal");
        lab7=new JLabel("Peas");
        lab8=new JLabel("Lady finger");
        vegetable.add(lab5);
        vegetable.add(lab6);
        vegetable.add(lab7);
        vegetable.add(lab8);

        tp=new JTabbedPane();
        tp.addTab("Fruit",fruit);
        tp.addTab("Vegetable",vegetable);
        tp.setBounds(50,50,200,200);
        frame.add(tp);
        frame.setSize(200,200);
        frame.setLayout(null);
        frame.setVisible(true);

    }

}
```

JScrollPane

A scroll pane is a component that presents a rectangular area in which a component may be viewed. Horizontal and/or vertical scroll bars may be provided if necessary. Scroll panes are implemented in Swing by the **JScrollPane** class, which extends **JComponent.**

Some of its constructors are shown here:
JScrollPane()
JScrollPane(Component comp)
JScrollPane(int vsb, int hsb)
JScrollPane(Component comp, int vsb, int hsb)

Here, comp is the component to be added to the scroll pane. **vsb and hsb** are **int constants** that define **when vertical** and **horizontal scroll bars** for this scroll pane are shown. These constants are defined by the **ScrollPaneConstants** interface. Some examples of these constants are described as follows:

HORIZONTAL_SCROLLBAR_ALWAYS
Always provide horizontal scroll bar

HORIZONTAL_SCROLLBAR_AS_NEEDED
Provide horizontal scroll bar, if needed

VERTICAL_SCROLLBAR_ALWAYS
Always provide vertical scroll bar

VERTICAL_SCROLLBAR_AS_NEEDED
Provide vertical scroll bar, if needed

Here are the steps that you should follow to use a scroll pane in an applet:
1. Create a JComponent object.
2. Create a JScrollPane object. (The arguments to the constructor specify the component and the policies for vertical and horizontal scroll bars.)
3. Add the scroll pane to the content pane of the applet.

Example

```
import javax.swing.*;
import java.awt.*;
public class JScrollPaneDemo
{
public static void main(String[] args)
{
JFrame frame = new JFrame("Scroll Pane Example");
JPanel jp = new JPanel();
jp.setLayout(new GridLayout(20, 20));
int b = 0;
for(int i = 0; i < 20; i++)
{
for(int j = 0; j < 20; j++)
{
jp.add(new JButton("Button " + b));
++b;
}
}
int v = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int h = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
JScrollPane jsp = new JScrollPane(jp, v, h);
frame.add(jsp,BorderLayout.CENTER);
frame.setSize(500, 500);
frame.setVisible(true);
}
}
```

Java JMenuBar, JMenu and JMenuItem

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

```
import javax.swing.*;
import java.awt.event.*;
public class MenuExample extends JFrame implements ActionListener
{
JMenuBar mb;
JMenu file,edit,help;
JMenuItem cut,copy,paste,selectAll;
JTextArea ta;
MenuExample()
{
cut=new JMenuItem("cut");
copy=new JMenuItem("copy");
paste=new JMenuItem("paste");
selectAll=new JMenuItem("selectAll");
cut.addActionListener(this);
copy.addActionListener(this);
paste.addActionListener(this);
selectAll.addActionListener(this);
mb=new JMenuBar();
file=new JMenu("File");
edit=new JMenu("Edit");
help=new JMenu("Help");
edit.add(cut);
edit.add(copy);
edit.add(paste);
edit.add(selectAll);
mb.add(file);
mb.add(edit);
mb.add(help);
ta=new JTextArea();
ta.setBounds(5,5,360,320);
add(ta);
setJMenuBar(mb);
}
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)
{
   MenuExample f = new MenuExample();
   f.setLayout(null);
   f.setSize(400,400);
   f.setVisible(true);
}
}
```

Java JTree

The JTree class is used to display the tree structured data or hierarchical data.
JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree.
Trees are implemented in Swing by the JTree class, which extends JComponent.

Commonly used Constructors:

Constructor	Description
JTree()	Creates a JTree with a sample model.
JTree(Object[] value)	Creates a JTree with every element of the specified array as the child of a new root node.
JTree(TreeNode root)	Creates a JTree with the specified TreeNode as its root, which displays the root node.

The **DefaultMutableTreeNode** class implements the **MutableTreeNode** interface.
It represents a node in a tree. One of its constructors is shown here:

DefaultMutableTreeNode(Object obj)

Here, obj is the object to be enclosed in this tree node.
The new tree node doesn’t have a parent or children.
To create a hierarchy of tree nodes, the add( ) method of DefaultMutableTreeNode can be used.

```
import javax.swing.*;
import java.awt.*;
import javax.swing.tree.*;
public class Tree2
{
public static void main(String args[])
{
JFrame jf = new JFrame("JTree");
JLabel label1 = new JLabel("Displaying tree");

DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode("Nutritious Food");

//Creating 3 children of the root node, Nutritious Food.
DefaultMutableTreeNode fruits = new DefaultMutableTreeNode("Fruits");
DefaultMutableTreeNode vegetables = new DefaultMutableTreeNode("Vegetables");
DefaultMutableTreeNode dryFruits = new DefaultMutableTreeNode("Dry Fruits");

//Adding 3 children - to Nutritious Food
rootNode.add(fruits);
rootNode.add(vegetables);
rootNode.add(dryFruits);

//Creating 3 children of fruits
DefaultMutableTreeNode fruit1 = new DefaultMutableTreeNode("Apple");
DefaultMutableTreeNode fruit2= new DefaultMutableTreeNode("Mango");
DefaultMutableTreeNode fruit3= new DefaultMutableTreeNode("Grapes");

//Adding 3 children to fruits
fruits.add(fruit1);
fruits.add(fruit2);
fruits.add(fruit3);


//Creating three children of vegetables
DefaultMutableTreeNode vegetable1 = new DefaultMutableTreeNode("Cabbage");
DefaultMutableTreeNode vegetable2= new DefaultMutableTreeNode("Tomato");
DefaultMutableTreeNode vegetable3 = new DefaultMutableTreeNode("Potato");

//Adding three child of vegetables
vegetables.add(vegetable1);
vegetables.add(vegetable2);
vegetables.add(vegetable3);

//Creating and adding one child dry fruits
DefaultMutableTreeNode dryFruit1 = new DefaultMutableTreeNode("Almonds");
DefaultMutableTreeNode dryFruit2 = new DefaultMutableTreeNode("Walnuts");
dryFruits.add(dryFruit1);
dryFruits.add(dryFruit2);
JTree tree = new JTree(rootNode);

//Adding JTree to JScrollPane
int v = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
int h = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
JScrollPane jsp = new JScrollPane(tree,v,h);

jf.add(label1);
jf.add(jsp,BorderLayout.CENTER);
jf.setSize(300,200);
jf.setVisible(true);
}
}
```

JTable

In Java, JTable is used to edit or display 2-D data which consists of rows and columns. It is almost similar to a spreadsheet that contains data in a tabular form.
JTable can be created by instantiating the class javax.swing.JTable.

Syntax of JTable in Java:

```
JTable jt=new JTable();
```

Constructors of JTable in Java

- **JTable():** A new table will be created with empty cells.
- **JTable(int r, int c):** A table will be created with the size as r*c.
- **JTable(Object[ ][ ] d, Object [ ]col):** A table will be created with the specified data where []col describes the names of column.

00	01	02
10	11	12
20	21	22
30	31	32

Methods of JTable in Java

The following are the most common methods of JTable in Java:

- **addColumn (Table Columnc):** A column c will be added to the column array end of the JTable column model.
- **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.
- **clearSelection ():** The columns and rows which are selected will be deselected.
- **getSelectedColumn ():** The index of the selected column which is selected first will be returned. It no column is selected, -1 will be returned.
- **getSelectedColumnCount ():** A count of selected columns will be returned.
- **getSelectedRow ():** The index of the selected row which is selected first will be returned. It no row is selected, -1 will be returned.
- **getSelectedRowCount ():** Count of selected rows will be returned.
- **isCellEditable (int r, int c):** If the cell in the specified row and column is editable, true will be returned.
- **removeColumn (TableColumnc):** Column c will be removed from the table's column array.
- **isCellSelected (int R, int C):** If the mentioned index is in the valid range of columns and rows and also, that position is selected, true will be returned.
- **isRowSelected (int r):** If the mentioned index is in the valid range of rows and also, that row is selected, true will be returned.

- **isColumnSelected (int c):** If the mentioned index is in the valid range of columns and also, that row is selected, true will be returned.

Example:

```
import javax.swing.*;
import java.awt.*;
public class TableExample
{
   public static void main(String[] args)
   {
   JFrame f=new JFrame();
   String data[][]={ {"101","Amit","670000"},
                {"102","Jai","780000"},
                {"103","Sachin","700000"},
                {"104","Amol","512000"},
                {"105","Seema","450000"},
                {"106","Gaurav","750000"}
               };
   String column[]={"ID","NAME","SALARY"};
   JTable jt=new JTable(data,column);
   jt.setBounds(30,40,300,300);
   int v = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
   int h = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
   JScrollPane sp=new JScrollPane(jt,v,h);
   f.add(sp,BorderLayout.CENTER);
   f.setSize(300,300);
   f.setVisible(true);
}
}
```

Java JProgressBar

The JProgressBar class is used to display the progress of the task. It inherits JComponent class.

JProgressBar class declaration

Commonly used Constructors:

Constructor	Description
JProgressBar()	It is used to create a horizontal progress bar but no string text.
JProgressBar(int min, int max)	It is used to create a horizontal progress bar with the specified minimum and maximum value.
JProgressBar(int orient)	It is used to create a progress bar with the specified orientation, it can be either Vertical or Horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
JProgressBar(int orient, int min, int max)	It is used to create a progress bar with the specified orientation, minimum and maximum value.

Commonly used Methods:

Method	Description
void setStringPainted(boolean b)	It is used to determine whether string should be displayed.
void setString(String s)	It is used to set value to the progress string.
void setOrientation(int orientation)	It is used to set the orientation, it may be either vertical or horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
void setValue(int value)	It is used to set the current value on the progress bar.

Example:

```
import javax.swing.*;
class Sample extends JFrame
{
  JProgressBar jpb;
  int i = 0;
  Sample()
```

```
 {
  jpb = new JProgressBar(0,2000);
  jpb.setBounds(40,40,120,40);
  jpb.setValue(0);
  jpb.setStringPainted(true);
  jpb.setString("Loading");
  add(jpb);
  setSize(500,500);
  setLayout(null);
 }
 void iterate()
 {
  while(i<=2000)
  {
   jpb.setValue(i);
   i=i+20;

   try
   {
    Thread.sleep(150);
   }
   catch(Exception e)
   {

   }
  }

 }

 public static void main(String[] args)
 {
  Sample s1 = new Sample();
  s1.setVisible(true);
  s1.iterate();
 }
}
```