

What is GUI(graphical user interface)

- A GUI (pronounced “GOO-ee”) gives an application a distinctive “look and feel.”
- A graphical user interface is a visual interface to a program. GUIs are built from GUI components (buttons, menus, labels etc). A GUI component is an object with which the user interacts via the mouse or keyboard.
- Together, the appearance and how user interacts with the program are known as the program "look and feel".
- The classes that are used to create GUI components are part of the java.awt or javax.swing package. Both these packages provide a rich set of user interface components.

GUI vs Non-GUI

- The classes present in the awt and swing packages can be classified into two broad categories. GUI classes & Non-GUI Support classes.
- The GUI classes as the name indicates are visible and user can interact with them. Examples of these are JButton, JFrame & JRadioButton etc
- The Non-GUI support classes provide services and perform necessary functions for GUI classes. They do not produce any visual output. Examples of these classes are Layout managers & Event handling

What is AWT

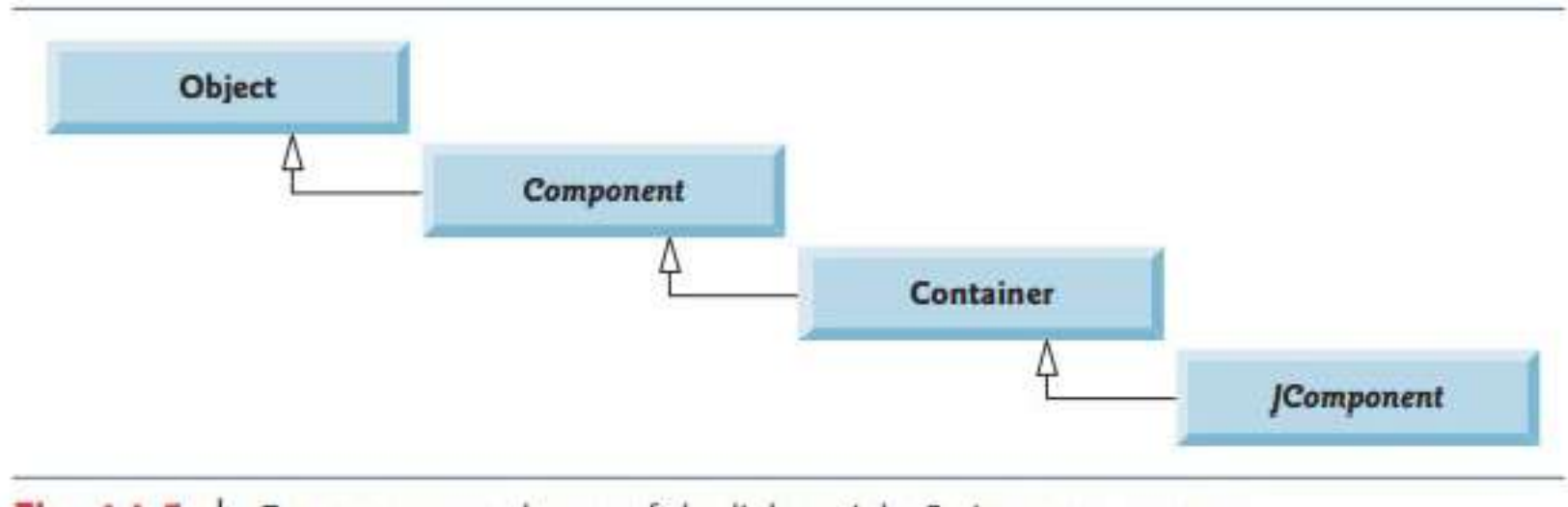
- AWT stands for Abstract Windowing Toolkit contains original but not pure GUI components that came with the first release of JDK.
- These components are tied directly to the local platform's (Windows, Linux, MAC etc) graphical user interface capabilities. Thus results in a java program executing on different java platforms (windows, Linux, Solaris etc) has a different appearance and sometimes even different user interaction on each platform.
- AWT components are often called Heavy Weight Components (HWC) as they rely on the local platform's windowing system.
- AWT component it creates a corresponding process on the operating system.
- In short component of AWT are OS depended

About Swing

- These are the newest GUI components. Swing components are written, manipulated and displayed completely in java, therefore also called pure java components. The swing components allow the programmer to specify a uniform look and feel across all platforms.
- javax.swing package is used to import
- not dependent on operating system
- 99% have lightweight components
- A rich set of classes which contain
- Jpanels, JButton, JTextarea,and so
- Names start from J of swing class

Superclasses of Swing's Lightweight GUI Components

- The Fig. shows an inheritance hierarchy of classes from which lightweight Swing components inherit their common attributes and behaviors.



Swing vs AWT

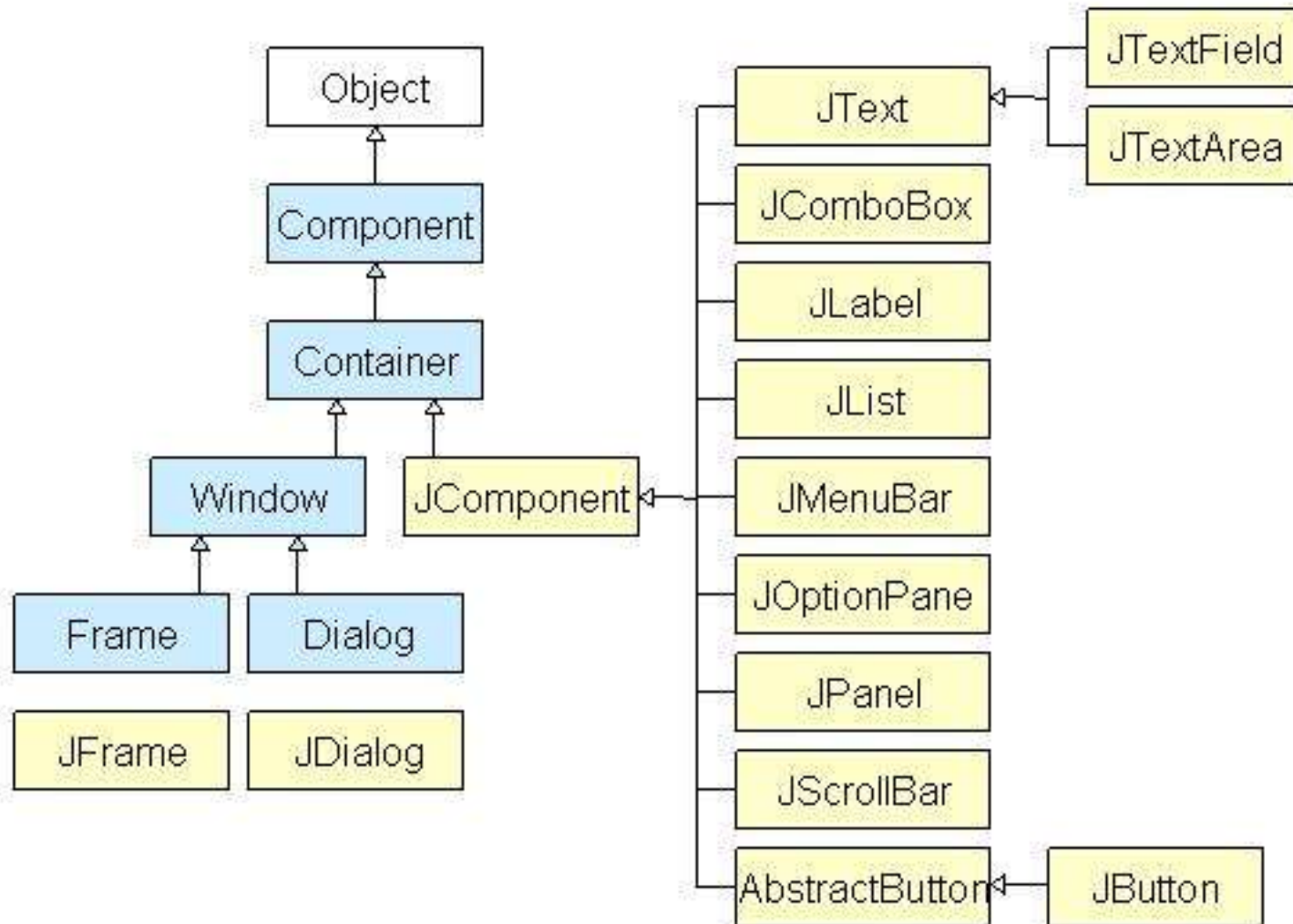
1. AWT

- AWT components are platform-dependent.
- AWT components are heavyweight.
- AWT doesn't support pluggable look and feel.
- AWT provides less components than Swing.
- Does not follow MVC where model represents data, view represents presentation and controller acts as an interface between model and view.

2. Swing

- Java swing components are platform-independent.
- Swing components are lightweight.
- Swing supports pluggable look and feel.
- Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
- Swing follows MVC.

Java Swing class Hierarchy Diagram



All components in swing are JComponent which can be added to container classes.

What is a container class?

- *Container classes are classes that can have other components on it. So for creating a GUI, we need at least one container object.*
- There are 3 types of containers.
 1. **Panel**: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window.
 2. **Frame**: It is a fully functioning window with its title and icons.
 3. **Dialog**: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame.
- .

Methods of Component class

- The methods of Component class are widely used in java swing that are given below.

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

JFrame

- JFrame is a class of javax.swing package extended by java.awt.frame, it adds support for JFC/SWING component architecture. It is the top level window, with border and a title bar. JFrame class has many methods which can be used to customize it.
- **Creating a JFrame**
- JFrame class has many constructors used to create a JFrame. Following is the description.
 1. **JFrame()**: creates a frame which is invisible
 2. **JFrame(GraphicsConfiguration gc)**: creates a frame with a blank title and graphics configuration of screen device.
 3. **JFrame(String title)**: creates a JFrame with a title.
 4. **JFrame(String title, GraphicsConfiguration gc)**: creates a JFrame with specific Graphics configuration and specified title.

JLabel

- JLabel is a class of java Swing .
- JLabel is used to display a short string or an image icon.
- JLabel can display text, image or both .
- JLabel is only a display of text or image and it cannot get focus .
- JLabel is inactive to input events such a mouse focus or keyboard focus.
- By default labels are vertically centered but the user can change the alignment of label.

JLabel

- Constructor of the class are :
 1. **JLabel()** : creates a blank label with no text or image in it.
 2. **JLabel(String s)** : creates a new label with the string specified.
 3. **JLabel(Icon i)** : creates a new label with a image on it.
 4. **JLabel(String s, Icon i, int align)** : creates a new label with a string, an image and a specified horizontal alignment

JLabel

- Commonly used methods of the class are :
 1. `getIcon()` : returns the image that that the label displays
 2. `setIcon(Icon i)` : sets the icon that the label will display to image i.
 3. `getText()` : returns the text that the label will display
 4. `setText(String s)` : sets the text that the label will display to string s creates a blank label with no text or image in it.

JLabel

```
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);
}
}
```

JTextField

- JTextField is a part of javax.swing package.
- The class JTextField is a component that allows editing of a single line of text. JTextField inherits the JTextComponent class and uses the interface SwingConstants.
- The constructor of the class are :
 1. **JTextField()** : constructor that creates a new TextField
 2. **JTextField(int columns)** : constructor that creates a new empty TextField with specified number of columns.
 3. **JTextField(String text)** : constructor that creates a new empty text field initialized with the given string.
 4. **JTextField(String text, int columns)** : constructor that creates a new empty text field with the given string and a specified number of columns .

JTextField

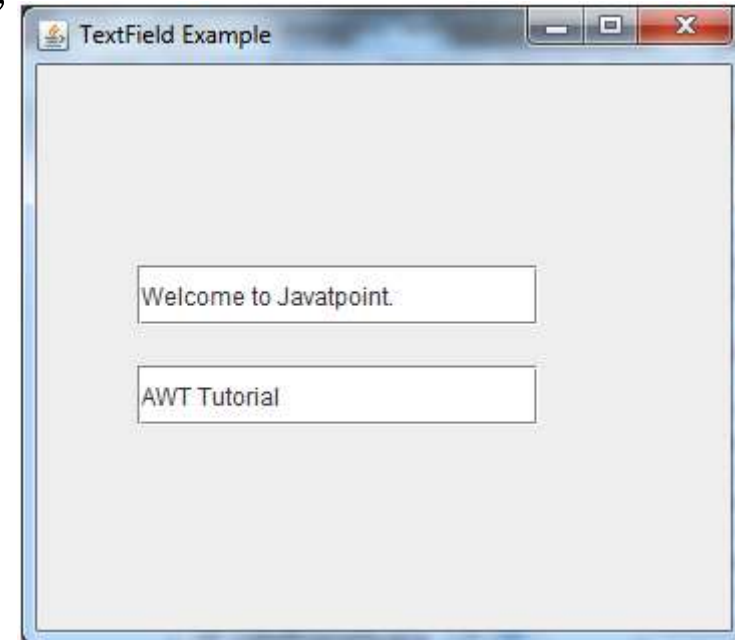
- Methods of the JTextField are:
 1. `setColumns(int n)` :set the number of columns of the text field.
 2. `setFont(Font f)` : set the font of text displayed in text field.
 3. `addActionListener(ActionListener l)` : set an ActionListener to the text field.
 4. `int getColumns()` :get the number of columns in the textfield.

JTextField

- Methods of the JTextField are:
 1. `setColumns(int n)` :set the number of columns of the text field.
 2. `setFont(Font f)` : set the font of text displayed in text field.
 3. `addActionListener(ActionListener l)` : set an ActionListener to the text field.
 4. `int getColumns()` :get the number of columns in the textfield.

JTextField

```
■ 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 Javatpoint.");  
        t1.setBounds(50,100, 200,30);  
        t2=new JTextField("AWT Tutorial");  
        t2.setBounds(50,150, 200,30);  
        f.add(t1); f.add(t2);  
        f.setSize(400,400);  
        f.setLayout(null);  
        f.setVisible(true);  
    }  
}
```



JTextArea

- JTextArea is a part of java Swing package . It represents a multi line area that displays text. It is used to edit the text .
- JTextArea inherits JComponent class. The text in JTextArea can be set to different available fonts and can be appended to new text .
- A text area can be customized to the need of user .

JTextArea

- Constructors of JTextArea are:
 1. `JTextArea()` : constructs a new blank text area .
 2. `JTextArea(String s)` : constructs a new text area with a given initial text.
 3. `JTextArea(int row, int column)` : constructs a new text area with a given number of rows and columns.
 4. `JTextArea(String s, int row, int column)` : constructs a new text area with a given number of rows and columns and a given initial text.

JTextArea

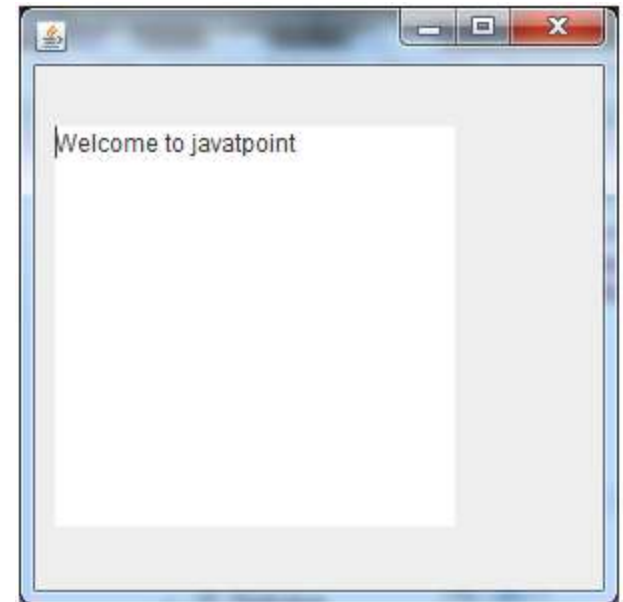
- Commonly used methods :
 1. `append(String s)` : appends the given string to the text of the text area.
 2. `getLineCount()` : get number of lines in the text of text area.
 3. `setFont(Font f)` : sets the font of text area to the given font.
 4. `setColumns(int c)` : sets the number of columns of the text area to given integer.
 5. `setRows(int r)` : sets the number of rows of the text area to given integer.
 6. `getColumns()` : get the number of columns of text area.
 7. `getRows()` : get the number of rows of text area.

JTextArea

- Commonly used methods :
 1. `append(String s)` : appends the given string to the text of the text area.
 2. `getLineCount()` : get number of lines in the text of text area.
 3. `setFont(Font f)` : sets the font of text area to the given font.
 4. `setColumns(int c)` : sets the number of columns of the text area to given integer.
 5. `setRows(int r)` : sets the number of rows of the text area to given integer.
 6. `getColumns()` : get the number of columns of text area.
 7. `getRows()` : get the number of rows of text area.

JTextArea

```
■ 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();
       }
   }
```



JTextArea

- JButton class is used to create a push button control, which can generate an ActionEvent when it is clicked.
- In order to handle a button click event, the ActionListener interface should be implemented.
- JButton is a component which extends the JComponent class and it can be added to a container.

JButton

- Constructors of JButton:

1. `public JButton()` :Creates a button with no text on it..
2. `public JButton(String text)` : Creates a button with a text on it.
3. `public JButton(Icon image)` : Creates a button with an icon on it.
4. `public JButton(String text, Icon image)` : Creates a button with a text and an icon on it.

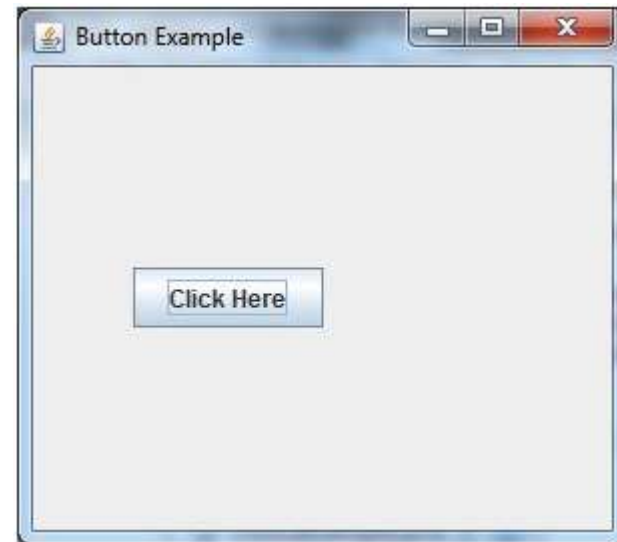
JButton

- Methods of JButton class

1. `public void setText(String text)` : Sets a String message on the JButton.
2. `public String getText()` : Gets a String message of JButton.
3. `public void setIcon(Icon icon)` : Sets an icon or image over the JButton.
4. `public Icon getIcon()` : Gets the icon or image of the JButton.
5. `void setHorizontalTextPosition(int textPosition)` : Sets the button message on the LEFT/RIGHT of its icon or image
6. `void setVerticalTextPosition(int textPosition)` : Sets the button message on the TOP/BOTTOM of its icon or image.

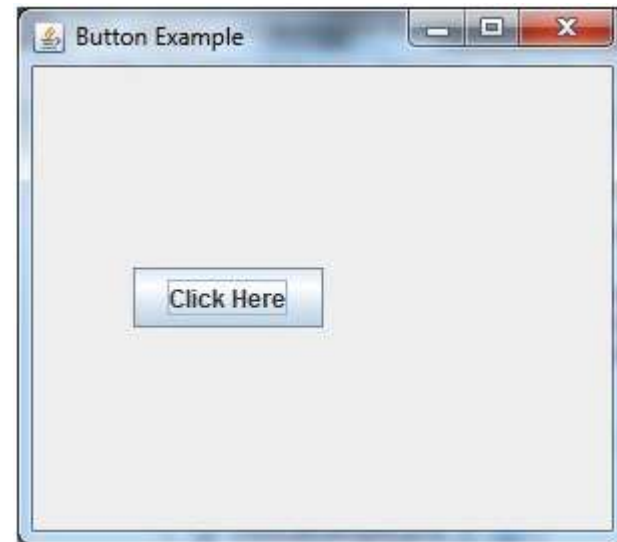
JButton

- ```
import javax.swing.*;
public class ButtonExample {
 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);
 }
}
```



# JButton

- ```
import javax.swing.*;  
public class ButtonExample {  
    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);  
    }  
}
```



JCheckBox

- A check box is a control that consists of a combination of a small box and a label.
- The label provides the description of the box with which it is associated. It is a two-state control having states true (checked) and false (unchecked).
- The state of a
- checkbox can be changed by clicking on it. A checkbox is an object of JCheckBox class.
- A group of JCheckBoxes is used when we need the user to make multiple choices.
- User can change this state by clicking on the check box of the component. Here is a typical JCheckBox component in default Java look and feel:

JCheckBox

■ Constructor of the class are :

1. **JCheckBox()** : creates a new checkbox with no text or icon

```
JCheckBox cb1 = new JCheckBox();
```

2. **JCheckBox(Icon i)** : creates a new checkbox with the icon specified

3. **JCheckBox(Icon icon, boolean s)** : creates a new checkbox with the icon specified and the boolean value specifies whether it is selected or not.

4. **JCheckBox(String t)** : creates a new checkbox with the string specified

5. **JCheckBox(String text, boolean selected)** : creates a new checkbox with the string specified and the boolean value specifies whether it is selected or not.

6. **JCheckBox(String text, Icon icon)** : creates a new checkbox with the string and the icon specified.

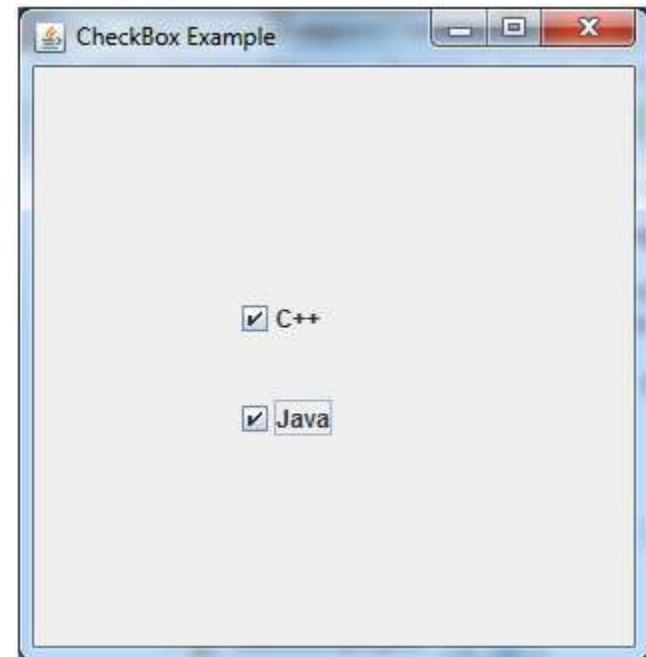
7. **JCheckBox(String text, Icon icon, boolean selected)** : creates a new checkbox with the string and the icon specified and the boolean value specifies whether it is selected or not.

JCheckBox

- Methods to add Item Listener to checkbox.
 1. `addActionListener(ItemListener l)`: adds item listener to the component
 2. `itemStateChanged(ItemEvent e)` : abstract function invoked when the state of the item to which listener is applied changes
 3. `getItem()` : Returns the component-specific object associated with the item whose state changed
 4. `getStateChange()` : Returns the new state of the item. The `ItemEvent` class defines two states: `SELECTED` and `DESELECTED`.
 5. `getSource()` : Returns the component that fired the item event.

JCheckBox

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



JRadioButton

- We use the JRadioButton class to create a radio button. Radio button is use to select one option from multiple options. It is used in filling forms, online objective papers and quiz.
- We add radio buttons in a ButtonGroup so that we can select only one radio button at a time. We use “ButtonGroup” class to create a ButtonGroup and add radio button in a group.
- 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.

JRadioButton

- Commonly used Methods:

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

```
import javax.swing.*;
import java.awt.*;
class RadioButtonExample extends JFrame
{
    RadioButtonExample()
    {
        setLayout(new FlowLayout());
        JLabel lbSex = new JLabel("SEX");
        JRadioButton rdbtMale = new JRadioButton("Male");
        JRadioButton rdbtfemale = new JRadioButton("Female", true);
        ButtonGroup sex = new ButtonGroup();
        sex.add(rdbtMale);
        sex.add(rdbtfemale);
        add(lbSex);
        add(rdbtMale);
        add(rdbtfemale);
    }
}
class JRadioButtonJavaExample
{
    public static void main(String args[])
    {
        RadioButtonExample frame = new RadioButtonExample();
        frame.setTitle ("JRadioButton Java Example");
        frame.setBounds(200,250,250,100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```



JList

- JList is part of Java Swing package . JList is a component that displays a set of Objects and allows the user to select one or more items .
- JList inherits JComponent class. JList is a easy way to display an array of Vectors .
- Constructor for JList are :
 1. **JList()**: creates an empty blank list
 2. **JList(E [])** : creates an new list with the elements of the array.
 3. **JList(ListModel d)**: creates a new list with the specified List Model
 4. **JList(Vector l)** : creates a new list with the elements of the vector

JList

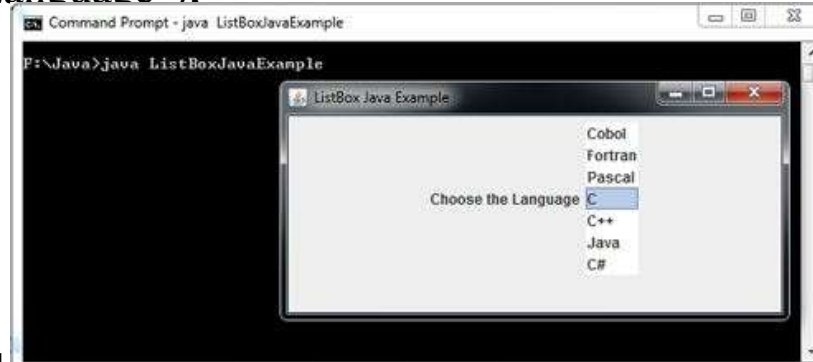
- Commonly used Methods:

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

```
■ import javax.swing.*;
    import java.awt.*;
    class ListBoxExample extends JFrame {
■     ListBoxExample() {
        setLayout(new FlowLayout());
        String[] language = {"Cobol","Fortran","Pascal","C","C++","Java","C#"};
        JLabel lblLanguage = new JLabel("Choose the Language");
        JList LstMonth = new JList(language);
        add(lblLanguage); add(LstMonth);
    }
}

class ListBoxJavaExample{
    public static void main(String args[]){
        ListBoxExample frame = new ListBoxExample(),
        frame.setTitle("ListBox Java Example");
        frame.setBounds(200,250,150,200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```



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 . JComboBox can be editable or read- only depending on the choice of the programmer .
- Constructor of the JComboBox are:
 1. `JComboBox()` : creates a new empty JComboBox .
 2. `JComboBox(ComboBoxModel M)` : creates a new JComboBox with items from specified ComboBoxModel
 3. `JComboBox(E [] i)` : creates a new JComboBox with items from specified array
 4. `JComboBox(Vector items)` : creates a new JComboBox with items from the specified vector

JComboBox

■ Commonly used Methods are :

1. `addItem(E item)` : adds the item to the JComboBox
2. `addItemListener(ItemListener l)` : adds a ItemListener to JComboBox
3. `getItemAt(int i)` : returns the item at index I
4. `getItemCount()`: returns the number of items from the list
5. `getSelectedItem()` : returns the item which is selected
6. `removeItemAt(int i)` : removes the element at index I
7. `setEditable(boolean b)` : the boolean b determines whether the combo box is editable or not .If true is passed then the combo box is editable or vice versa.
8. `setSelectedIndex(int i)`: selects the element of JComboBox at index i.
9. `showPopup()` :causes the combo box to display its popup window.
10. `setSelectedItem(Object a)`: sets the selected item in the combo box display area to the object in the argument.
11. `setSelectedIndex(int a)`: selects the item at index anIndex.

JComboBox

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

class ComboBoxExample extends JFrame{
    ComboBoxExample() {
        setLayout(new FlowLayout());
        String[] month = {"Jan","feb","mar","Apr","May","Jun","Jul","Aug","sep","oct","Nov","Dec"};
        JLabel lblDay = new JLabel("Day");
        JLabel lblMonth = new JLabel("Month");
        JLabel lblYear = new JLabel("Year");
        JComboBox cboDay = new JComboBox();
        JComboBox cboMonth = new JComboBox(month);
        JComboBox cboYear = new JComboBox();
        for(int i=1;i<=31;i++)
            cboDay.addItem(i);
        for(int i=2009;i>1970;i--)
            add(lblDay); add(cboDay);
            add(lblMonth); add(cboMonth);
            add(lblYear); add(cboYear);
    }
}

class JComboBoxJavaExample {
    public static void main(String args[]) {
        ComboBoxExample frame = new ComboBoxExample();
        frame.setTitle("ComboBox Java Example");
        frame.setBounds(200,250,350,50);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```



JTable

- The JTable class is a part of Java Swing Package and is generally used to display or edit two-dimensional data that is having both rows and columns. It is similar to a spreadsheet. This arranges data in a tabular form.
- Constructors in JTable:
 1. `JTable()`: A table is created with empty cells.
 2. `JTable(int rows, int cols)`: Creates a table of size rows * cols.
 3. `JTable(Object[][] data, Object []Column)`: A table is created with the specified name where []Column defines the column names.

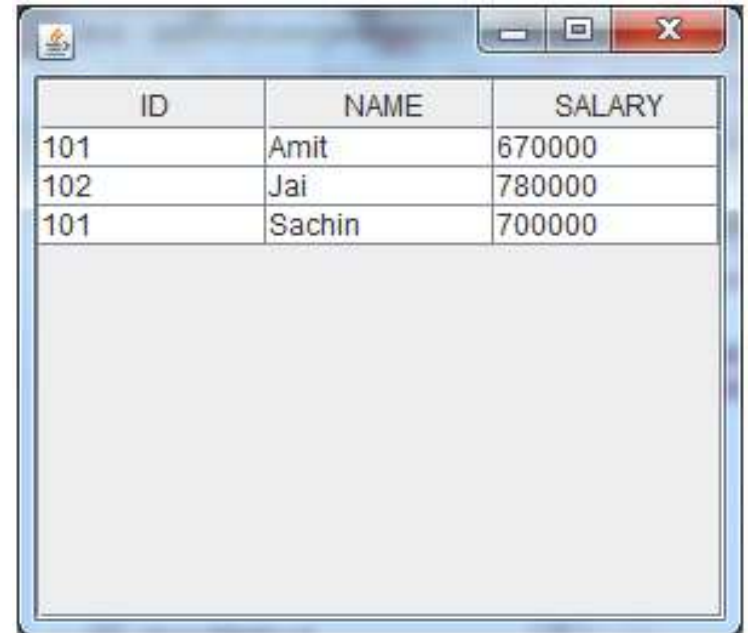
JTable

- Functions in JTable:

1. `addColumn(TableColumn []column)` : adds a column at the end of the JTable.
2. `clearSelection()` : Selects all the selected rows and columns.
3. `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.
4. `setValueAt(Object value, int row, int col)` : Sets the cell value as 'value' for the position row, col in the JTable.

JTable

```
■ 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();
    }
  }
```



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

JTabbedPane

- JTabbedPane in Swing enables you to group several swing component groups on the same container. To view a particular component group, you need to select the tab corresponding to the component group in the tabbed pane.
- To create a tabbed pane, create an instance of JTabbedPane class and add various tabs to it using the addTab() method of JTabbedPane class.
- The various constructors that swing provides to create a JTabbedPane are:
 1. **JTabbedPane():** Creates an empty JTabbedPane with default placement of JTabbedPane.Top. The placement specifies the division of JTabbedPane.
 2. **JTabbedPane(int placement):** Creates a JTabbedPane with specified placement that is passed as an argument to the constructor. The various placements for a JTabbedPane are JTabbedPane.TOP, JTabbedPane.BOTTOM, JTabbedPane.LEFT and JTabbedPane.RIGHT.
 3. **JTabbedPane(int placement, int tabLayoutPolicy):** Creates a JTabbedPane with the placement and tabLayoutPolicy passed as an argument to the constructor.

JTabbedPane

- JTabbedPane in Swing enables you to group several swing component groups on the same container. To view a particular component group, you need to select the tab corresponding to the component group in the tabbed pane.
- To create a tabbed pane, create an instance of JTabbedPane class and add various tabs to it using the addTab() method of JTabbedPane class.
- The various constructors that swing provides to create a JTabbedPane are:
 1. **JTabbedPane():** Creates an empty JTabbedPane with default placement of JTabbedPane.Top. The placement specifies the division of JTabbedPane.
 2. **JTabbedPane(int placement):** Creates a JTabbedPane with specified placement that is passed as an argument to the constructor. The various placements for a JTabbedPane are JTabbedPane.TOP, JTabbedPane.BOTTOM, JTabbedPane.LEFT and JTabbedPane.RIGHT.
 3. **JTabbedPane(int placement, int tabLayoutPolicy):** Creates a JTabbedPane with the placement and tabLayoutPolicy passed as an argument to the constructor.

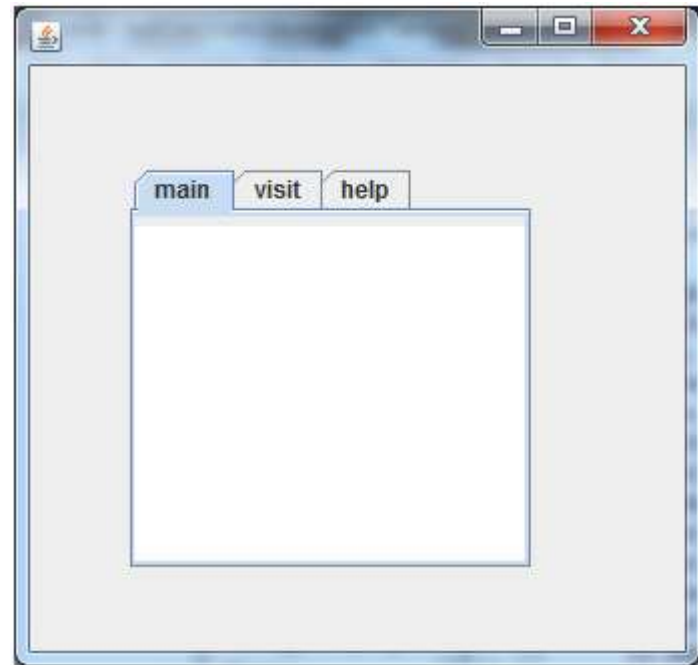
JTabbedPane

```
import javax.swing.*;

public class TabbedPaneExample {
    JFrame f;

    TabbedPaneExample(){
        f=new JFrame();
        JTextArea ta=new JTextArea(200,200);
        JPanel p1=new JPanel();
        p1.add(ta);
        JPanel p2=new JPanel();
        JPanel p3=new JPanel();
        JTabbedPane tp=new JTabbedPane();
        tp.setBounds(50,50,200,200);
        tp.add("main",p1);
        tp.add("visit",p2);
        tp.add("help",p3);
        f.add(tp);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }

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



JScrollPane

- A scroll pane is a container that represents a small area to view other component. If the component is larger than the visible area, scroll pane provides horizontal and/or vertical scroll bars automatically for scrolling the components through the pane.
- A scroll pane is an object of the JScrollPane class which extends JComponent.

Constructor

JScrollPane()

JScrollPane(Component)

JScrollPane(int, int)

JScrollPane(Component, int, int)

Purpose

It creates a scroll pane. The 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).

JScrollPane

Useful Methods

Modifier	Method	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)	It sets or gets the specified corner. The int parameter specifies which corner and must be one of the following constants defined in JScrollPaneConstants: UPPER_LEFT_CORNER, UPPER_RIGHT_CORNER, LOWER_LEFT_CORNER, LOWER_RIGHT_CORNER, LOWER_LEADING_CORNER, LOWER_TRAILING_CORNER, UPPER_LEADING_CORNER, UPPER_TRAILING_CORNER.
------	------------------------------	--

Component	getCorner(String)	
-----------	-------------------	--

void	setViewportView(Component)	Set the scroll pane's client.
------	----------------------------	-------------------------------

JScrollPane