Java Swing

UNIT-III

GANADIPATHY TULSIS JAIN
ENGINEERING COLLEGE

- **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.

- 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

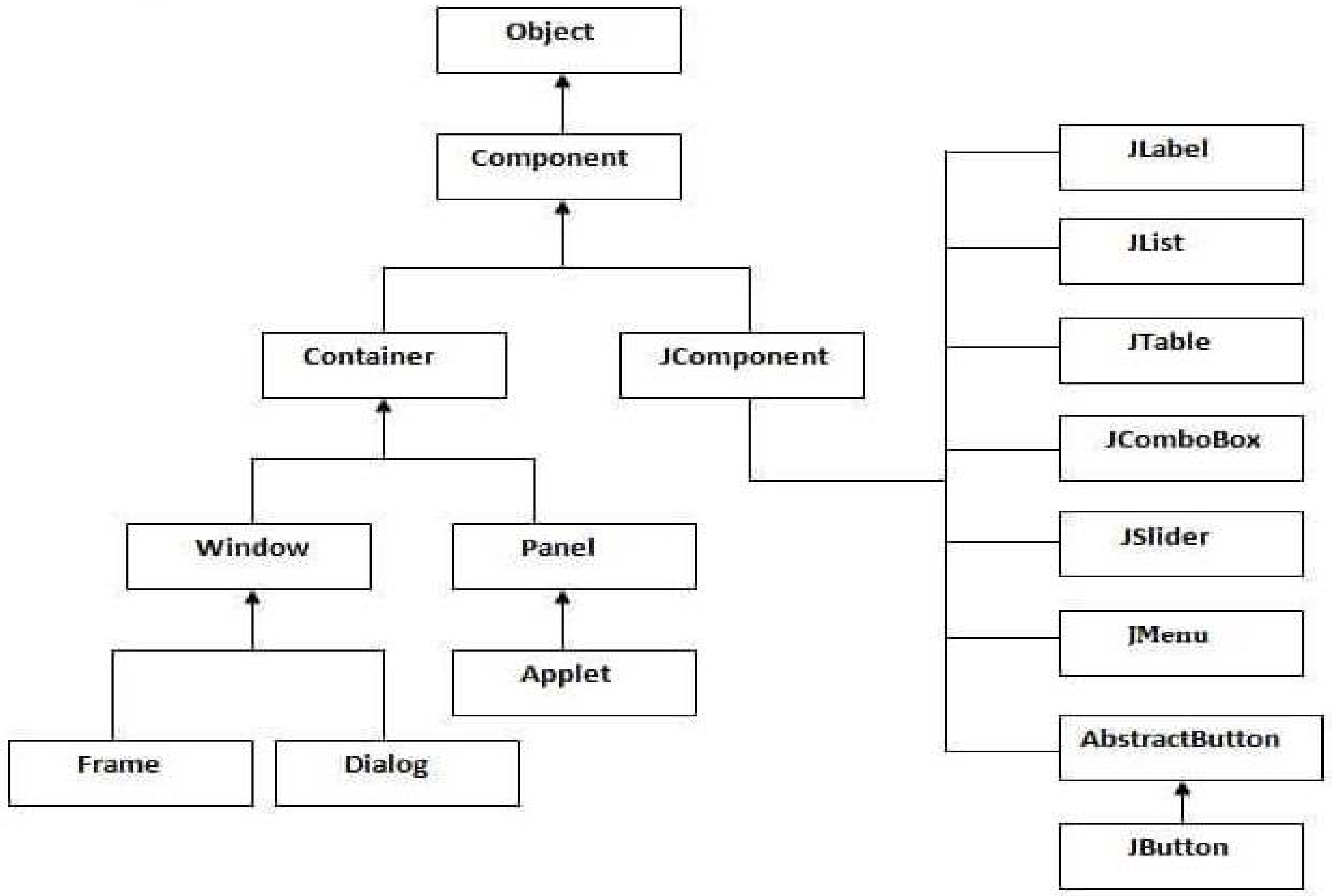
There are many differences between java awt and swing that are given below.

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.



Commonly used Methods of Component class

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

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.

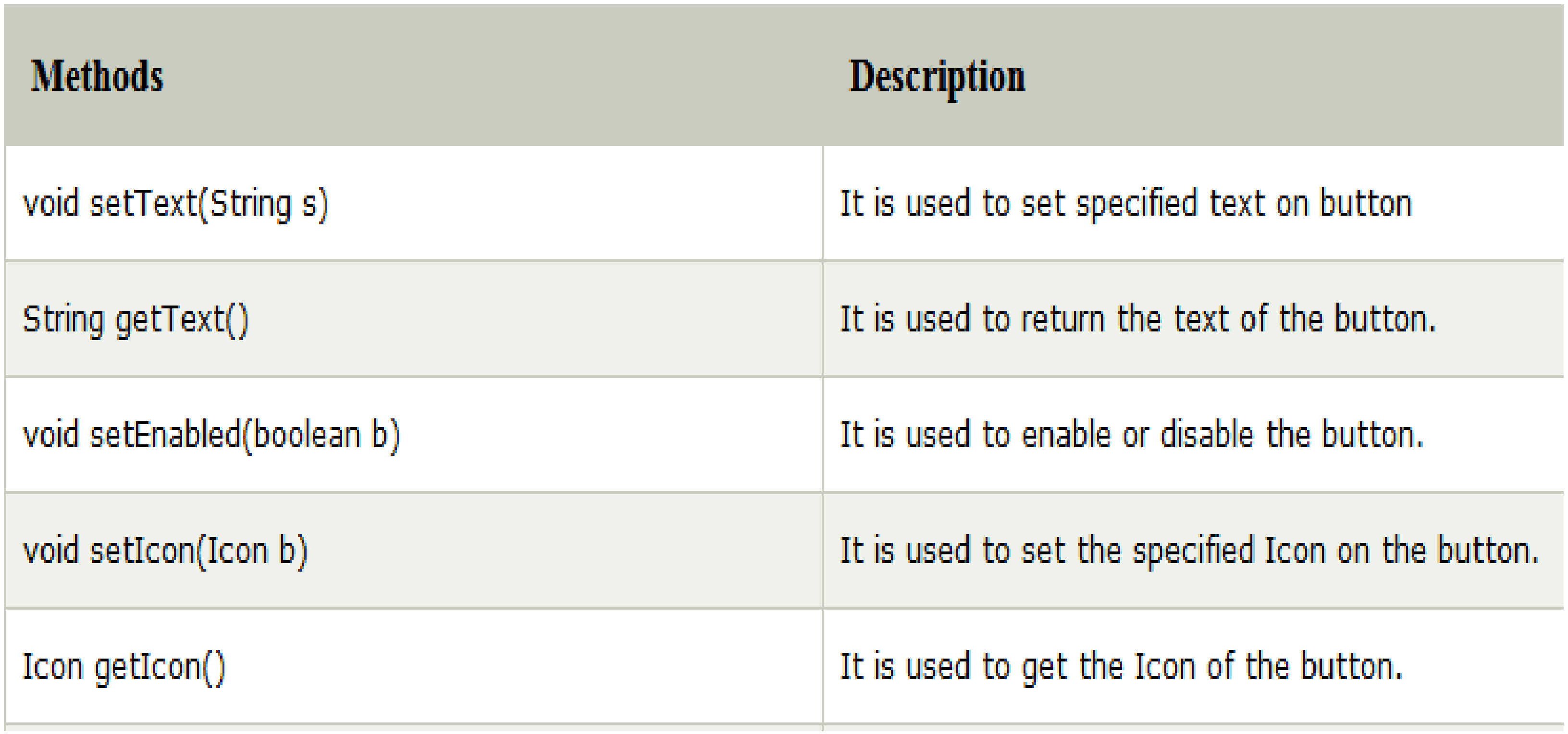
1. JButton class
2. JRadioButton class
3. JTextArea class
4. JComboBox class
5. JTable class
6. JColorChooser class
7. JProgressBar class
8. JSlider class
9. Digital Watch
10. Graphics in swing
11. Displaying image
12. Edit menu code for Notepad
13. OpenDialog Box
14. Notepad
15. Puzzle Game
16. Pic Puzzle Game
17. Tic Tac Toe Game
18. BorderLayout
19. GridLayout
20. FlowLayout
21. CardLayout

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

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

- **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.

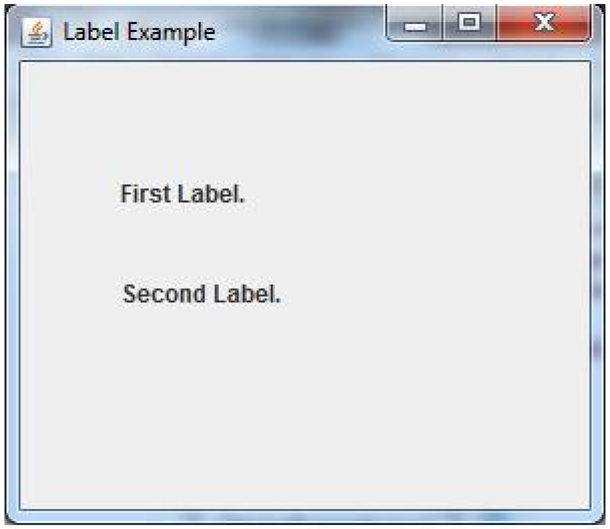


Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()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

```
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);
}
}
```

- **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.

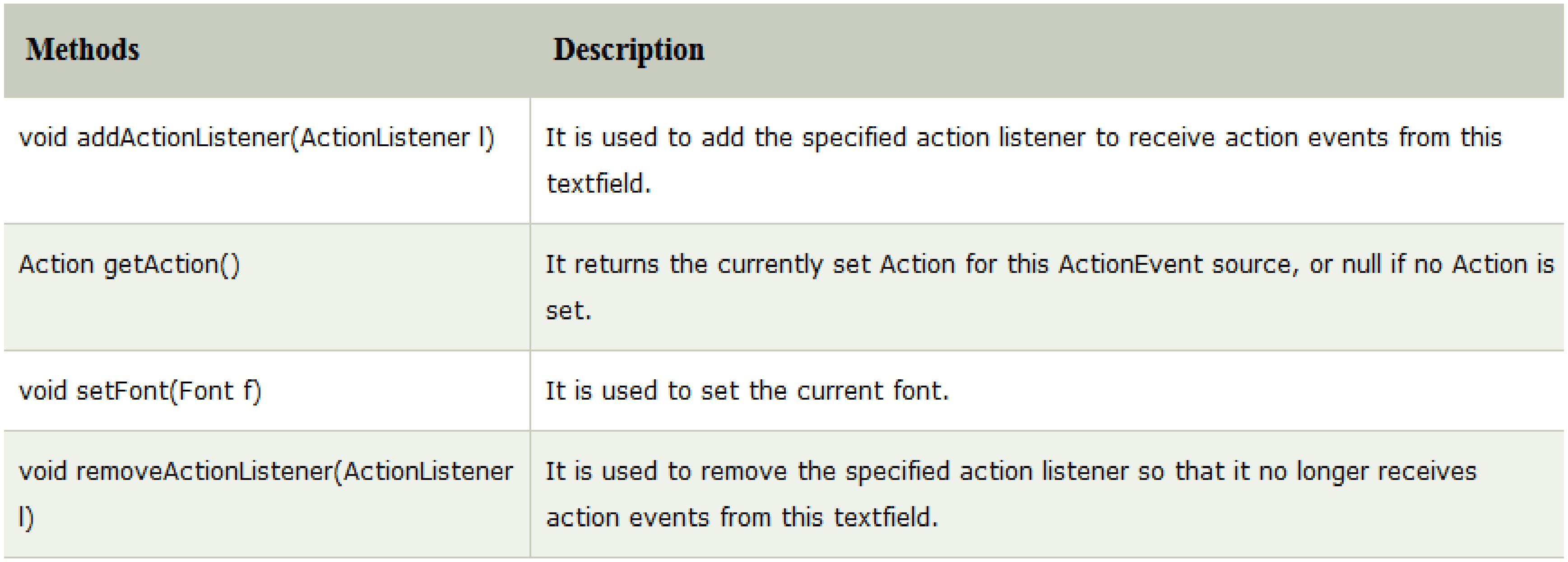
```
import javax.swing.*;
class LabelExample
{
public static void main(String args[])
    {
    JFrame f= new JFrame("Label Example");
    JLabel l1,l2;
    l1=new JLabel("First Label.");
    l1.setBounds(50,50, 100,30);
    l2=new JLabel("Second Label.");
    l2.setBounds(50,100, 100,30);
    f.add(l1);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



First Label.

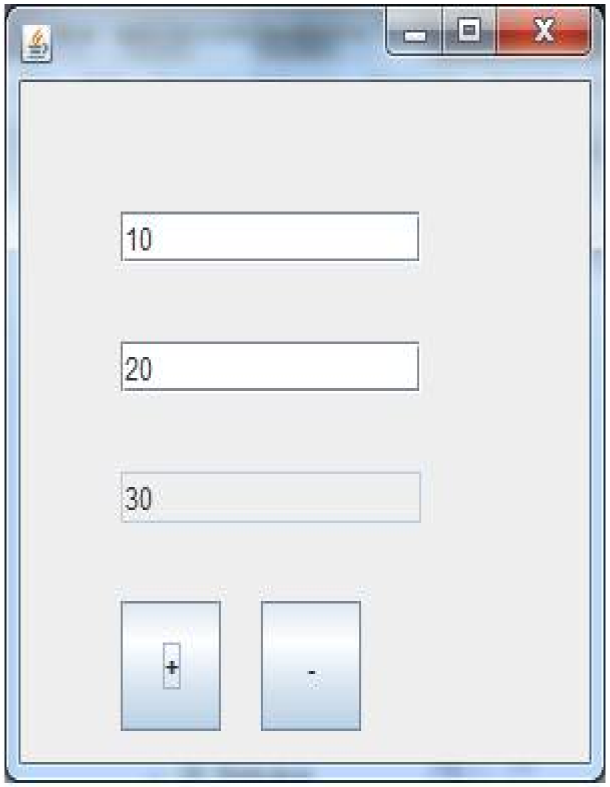
Second Label.

- **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.

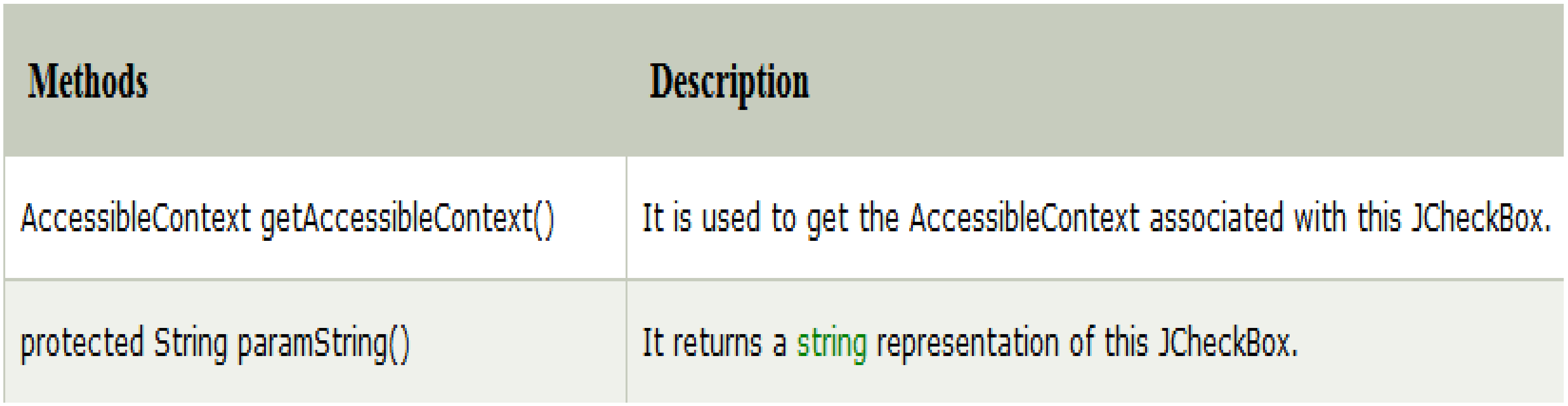


Methods	Description
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	It is used to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

```
import javax.swing.*;
import java.awt.event.*;
public class TextFieldExample implements ActionListener{
      JTextField tf1,tf2,tf3;
      JButton b1;
        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);
        b1.addActionListener(this);
f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);
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;
}
String result=String.valueOf(c);
tf3.setText(result);
}
public static void main(String[] args) {
new TextFieldExample();
} }
```

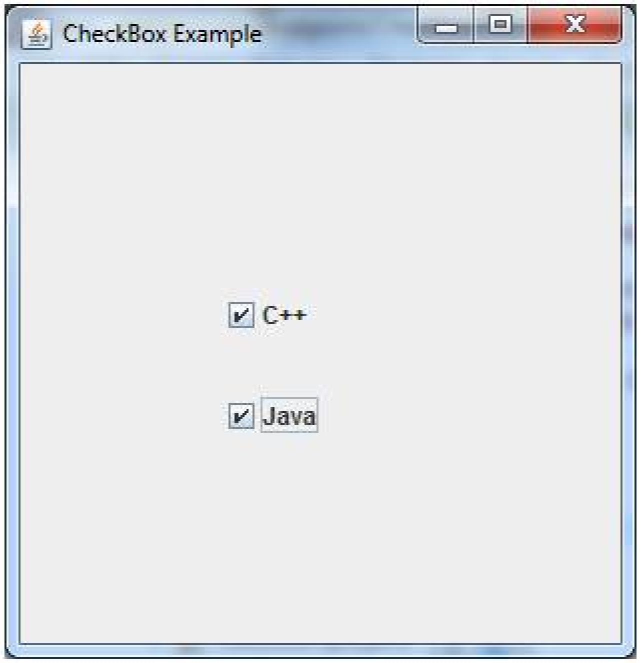


- **Java JCheckBox**
- The JCheckBox class is used to create a checkbox.
- It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".
- It inherits JToggleButton class.



Methods	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.
protected String paramString()	It returns a string representation of this 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();
   }}
```



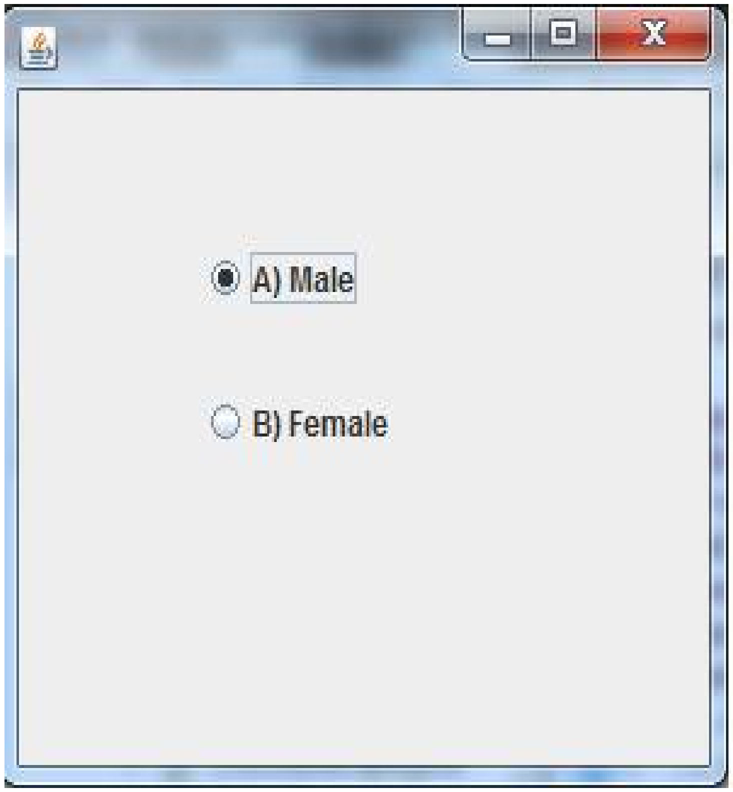
- **Java JRadioButton**
- The JRadioButton class is used to create a radio button.

- It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

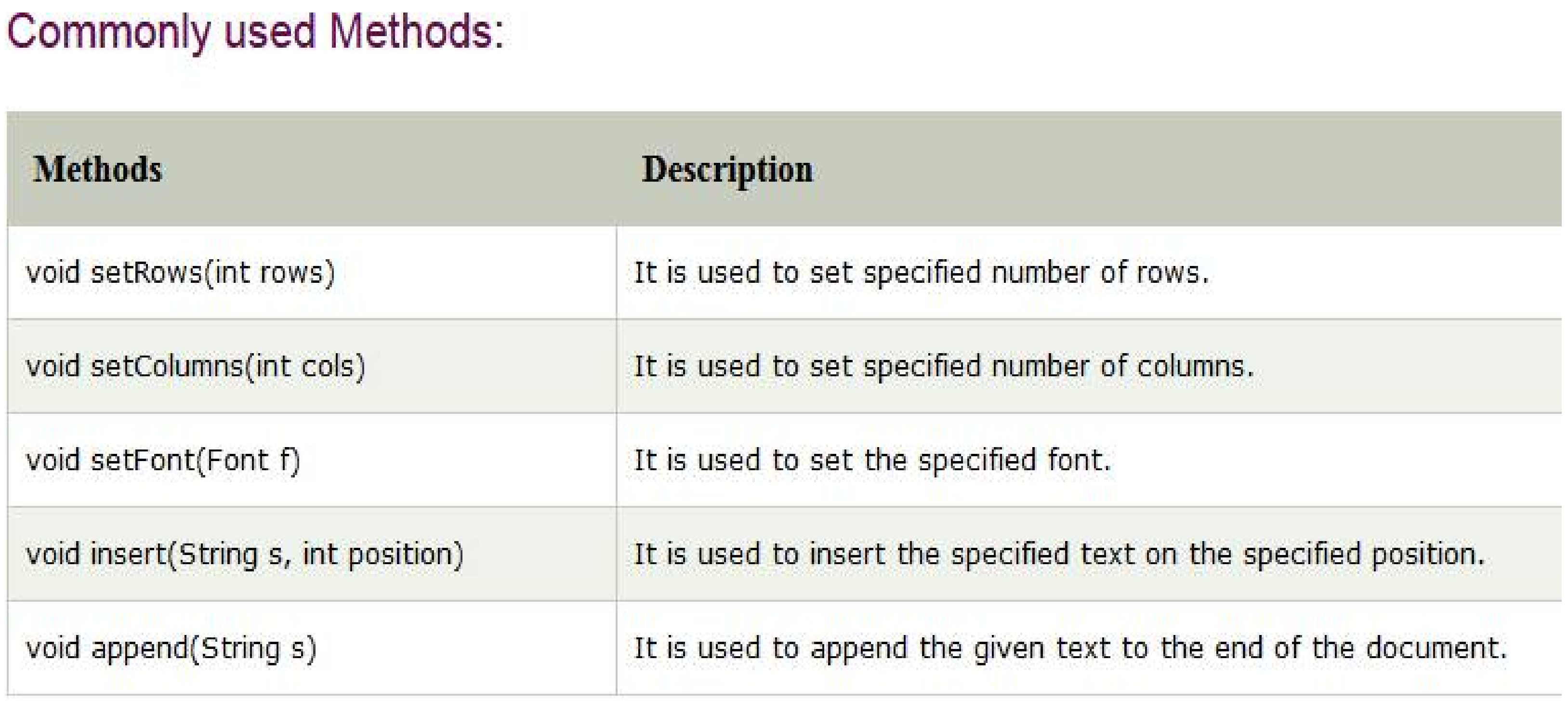
- It should be added in ButtonGroup to select one radio button only.

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

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



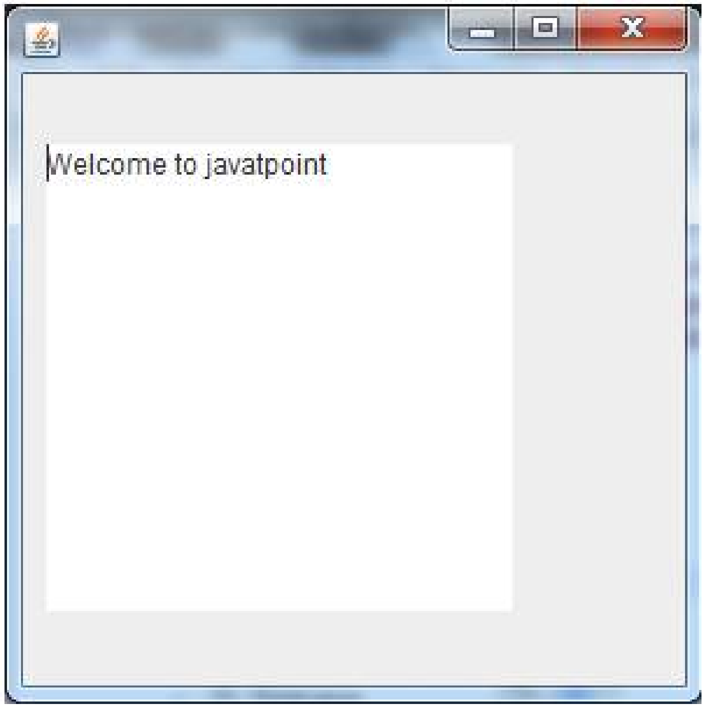
- **Java JTextArea**
- The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text.
- It inherits JTextComponent class



Commonly used Methods:

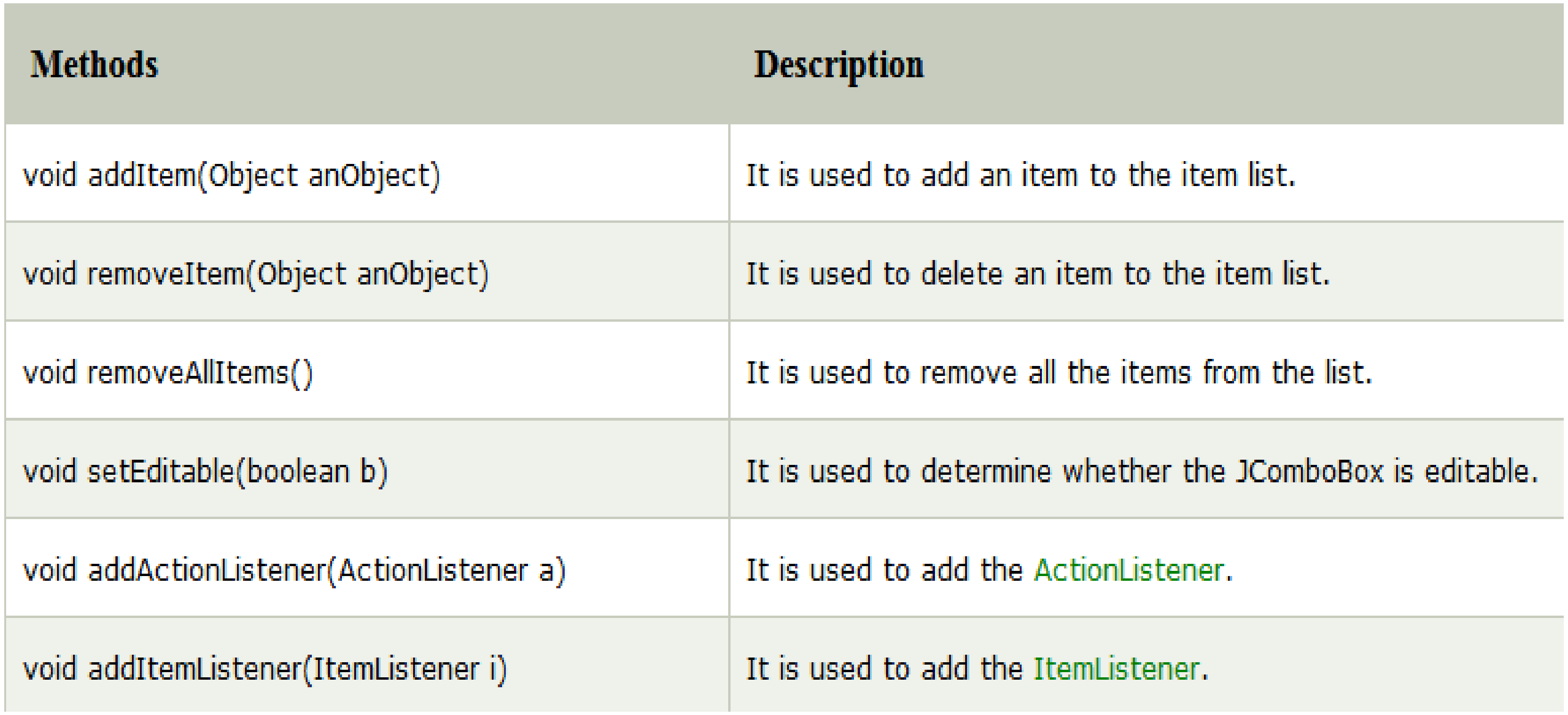
Methods	Description
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the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

```
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();
    }}
```



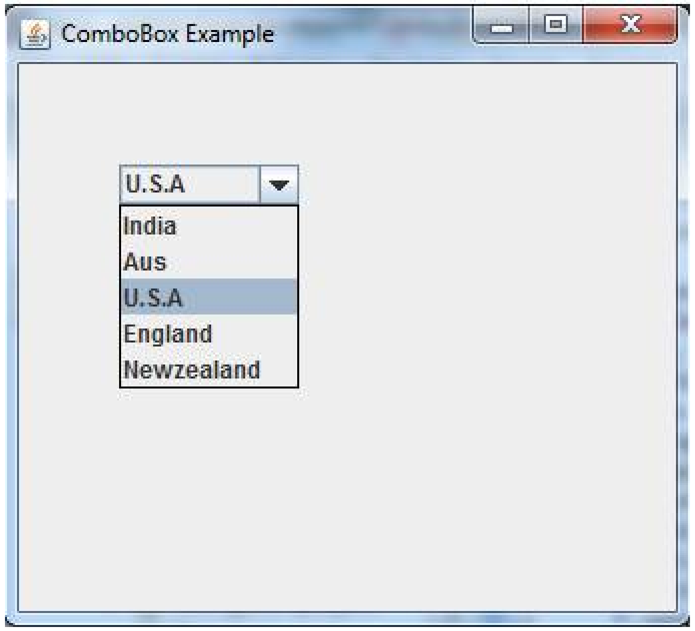
Welcome to javatpoint

- **Java 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.



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

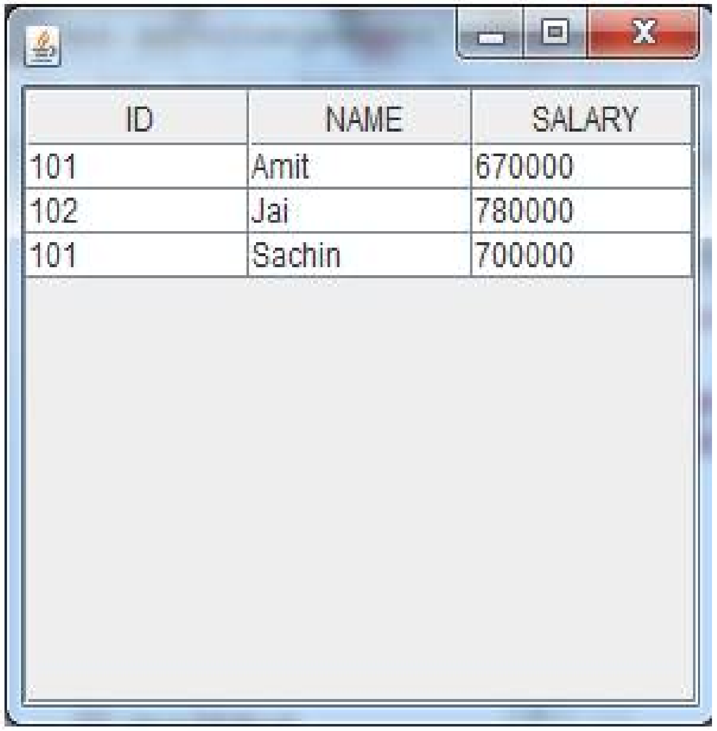
```
import javax.swing.*;
public class ComboBoxExample {
        JFrame f;
        ComboBoxExample(){
        f=new JFrame("ComboBox Example");
        String country[]={"India","Aus","U.S.A","England","Newzealand"};
        JComboBox cb=new JComboBox(country);
        cb.setBounds(50, 50,90,20);
        f.add(cb);
        f.setLayout(null);
        f.setSize(400,500);
        f.setVisible(true);
}
public static void main(String[] args) {
new ComboBoxExample();
}
}
```



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

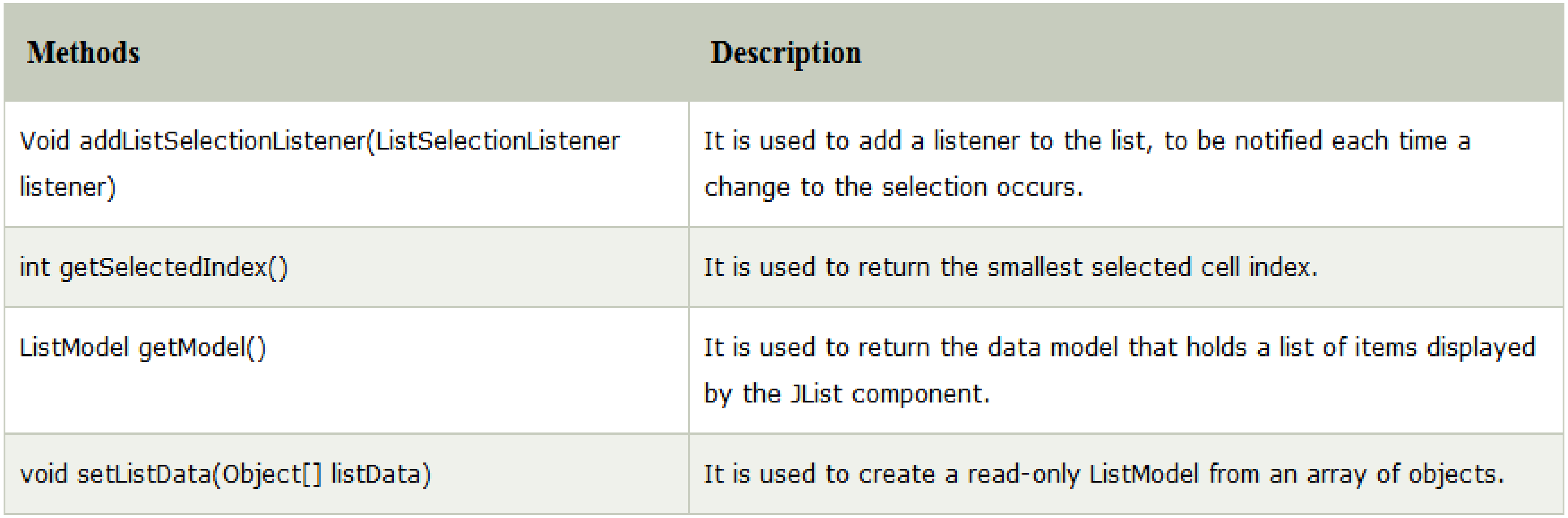
- **JTable class declaration**
- Let's see the declaration for javax.swing.JTable class.

```
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();
}
}
```



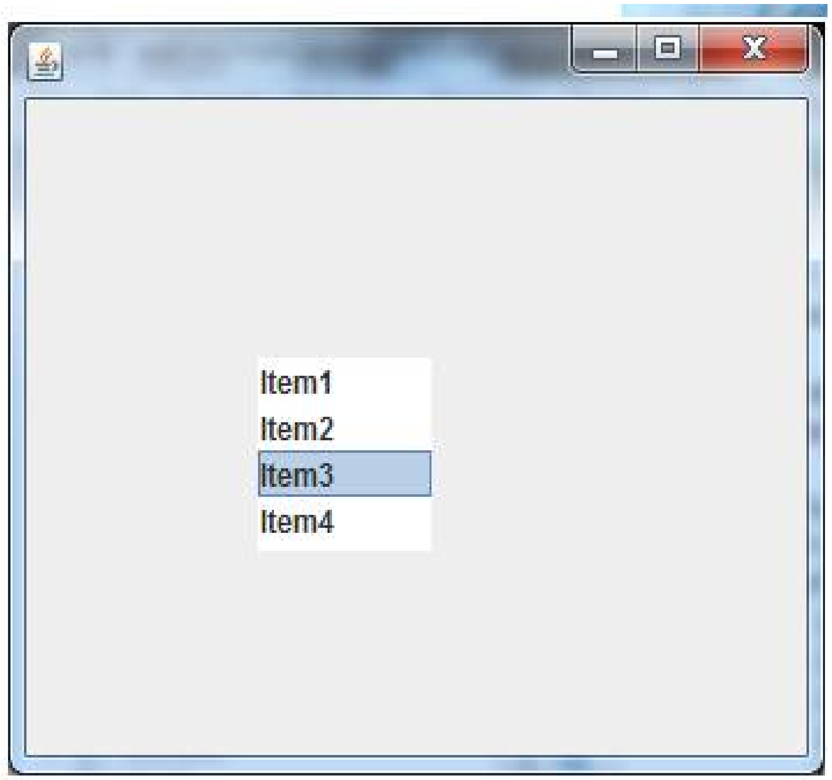
ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

- **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.



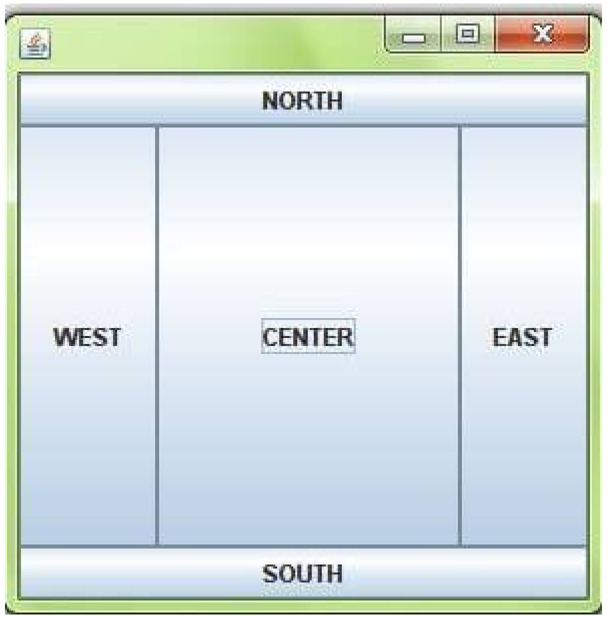
Methods	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.
int getSelectedIndex()	It is used to return the smallest selected cell index.
ListModel getModel()	It is used to return 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.

```
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();
   }}
```

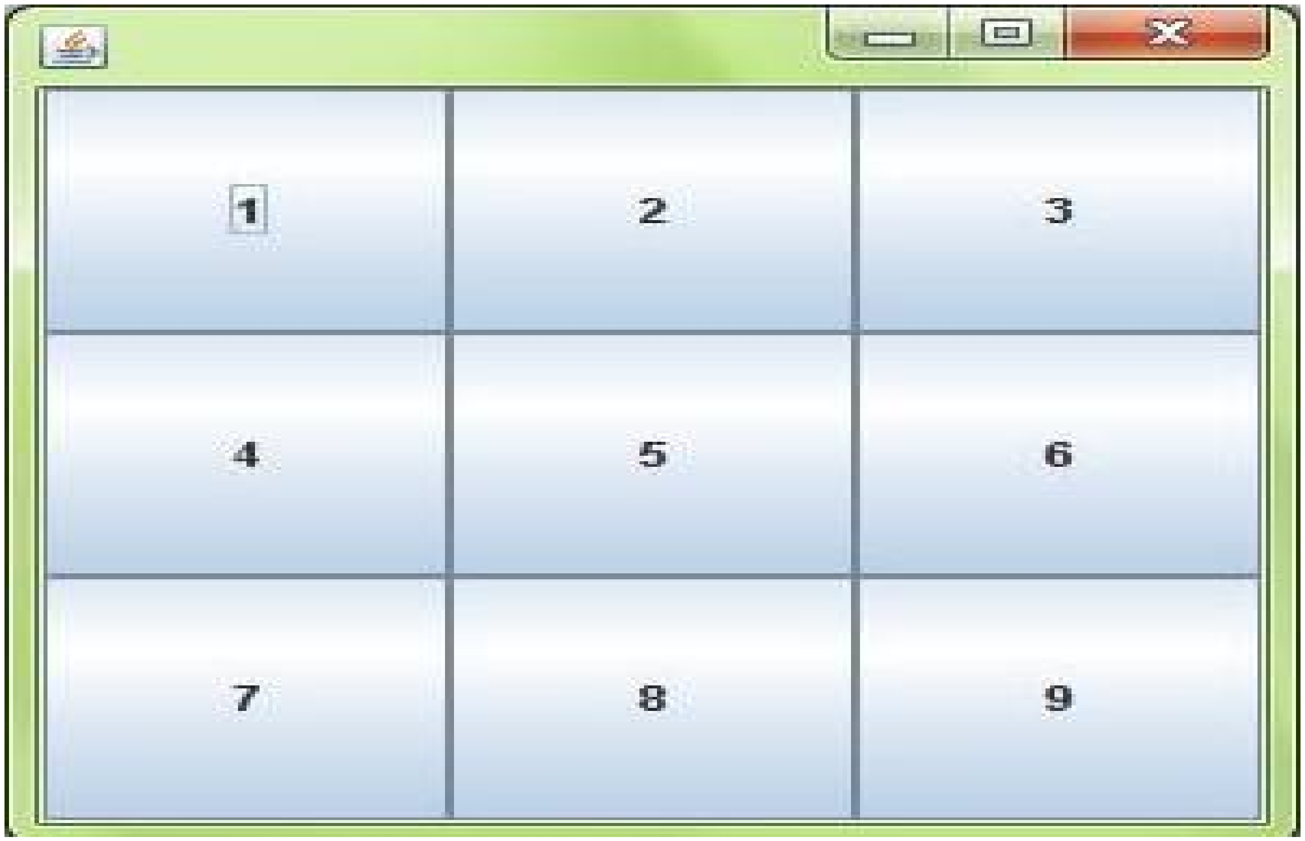


- **Java LayoutManagers**
- java.awt.BorderLayout
- java.awt.FlowLayout
- java.awt.GridLayout
- java.awt.CardLayout
- java.awt.GridBagLayout
- javax.swing.BoxLayout
- javax.swing.GroupLayout
- javax.swing.ScrollPaneLayout
- javax.swing.SpringLayout etc.

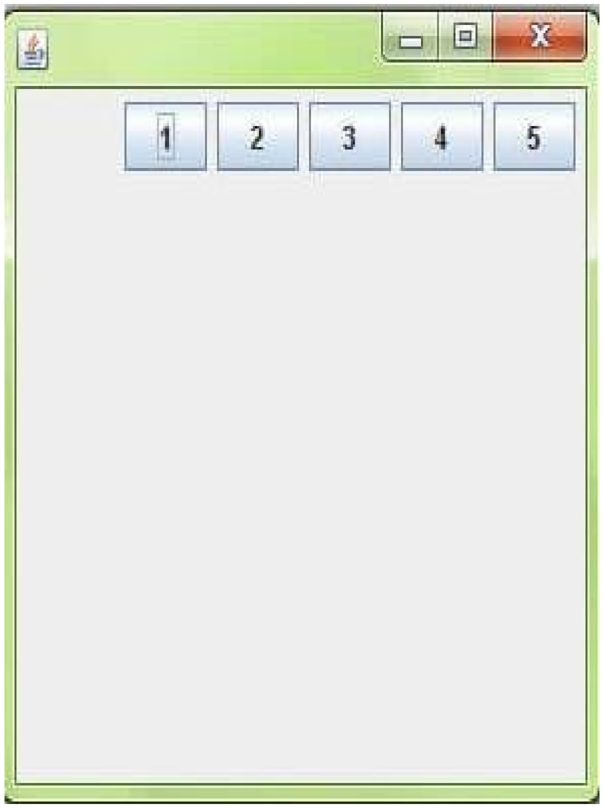
- Java BorderLayout
- The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:
- **public static final int NORTH**
- **public static final int SOUTH**
- **public static final int EAST**
- **public static final int WEST**
- **public static final int CENTER**



- **Java GridLayout**
- The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.



- **Java FlowLayout**
- The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.
- **Fields of FlowLayout class**
- **public static final int LEFT**
- **public static final int RIGHT**
- **public static final int CENTER**
- **public static final int LEADING**
- **public static final int TRAILING**



- **Java CardLayout**
- The CardLayout class manages the components in such a manner that only one component is visible at a time.
- It treats each component as a card that is why it is known as CardLayout.
- **Commonly used methods of CardLayout class**
- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

RMI-Remote Method Invocation

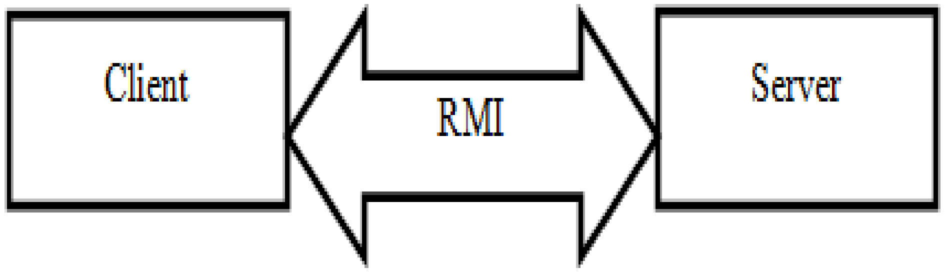
- RMI stands for “Remote Method Invocation (RMI)” means communicating the object across the network.

- RMI is a one type of structure or system that allows an object running in one Java Virtual Machine (client) to invoke methods on an object running in another Java Virtual Machine (server).

- This object is called a **Remote object** and such a system is also called **RMI Distributed application**.

- RMI provides for remote communication between programs written in the JAVA.

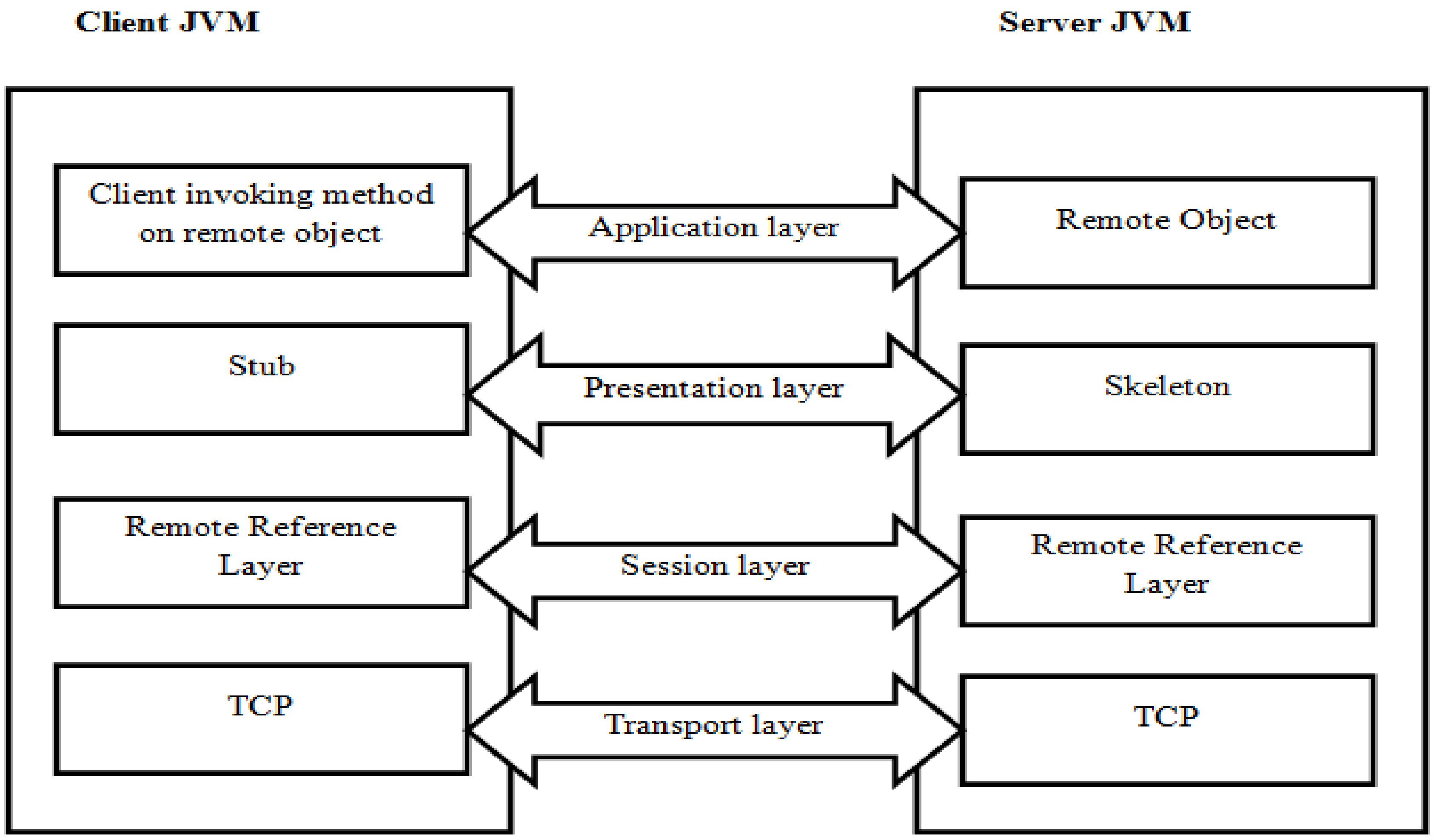
- **General Functioning Diagram**
- Client
- Server
- RMI



- **The complete RMI system has a four layer,**
- Application layer
- Proxy layer
- Remote Reference layer
- Transport layer

- Mainly the RMI application contains the three components,

1. RMI server 2.RMI client 3.RMI registry


Client JVM
Server JVM
Client invoking method on remote object
Stub
Remote Reference Layer
TCP
Application layer
Presentation layer
Session layer
Transport layer
Remote Object
Skeleton
Remote Reference Layer
TCP