# FAST

National University of Computer and Emerging Sciences Peshawar

Lecture # 18

# Software Construction and Development (Java Programming)

**Instructor:** Muhammad Abdullah Orakzai

DEPARTMENT OF COMPUTER SCIENCE


الذی علم بالقلم۔ علم الانسان ما لم یعلم۔



# Graphical User Interface in Java (GUI)

# Contents

# Graphical User Interfaces

- 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.

# Support for GUI in Java

- The classes that are used to create GUI components are part of the "java.awt" or "javax.swing" package.

- Both these packages provide rich set of user interface components.

# GUI classes vs. Non-GUI Support Classes

The classes present in the awt and swing packages can be classified into two broad categories:

1) GUI Classes

2) Non-GUI Support classes

# 1) GUI Classes

The GUI classes as the name indicates are visible and user can interact with them.

**Examples** of these are JButton, JFrame & JRadioButton etc.

# 2) Non-GUI Support classes

- 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 (discussed latter) & Event handling classes etc.

# java.awt package

- AWT stands for “Abstract Windowing Toolkit“ contains original 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.

# java.awt package…

- AWT components are often called Heavy Weight Components (HWC) as they rely on the local platform's windowing system to determine their functionality and their look and feel.

- Every time you create an AWT component it creates a corresponding process on the operating system.

- As compared to this SWING components are managed through threads and are known as Light Weight Components.

# java.awt package…

- This package also provides the classes for robust event handling and layout managers.

# javax.swing package

- 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.

- Swing components are often referred to as Light Weight Components as they are completely written in java. Several swing components are still HWC e.g. JFrame etc.

# Java AWT Tutorial

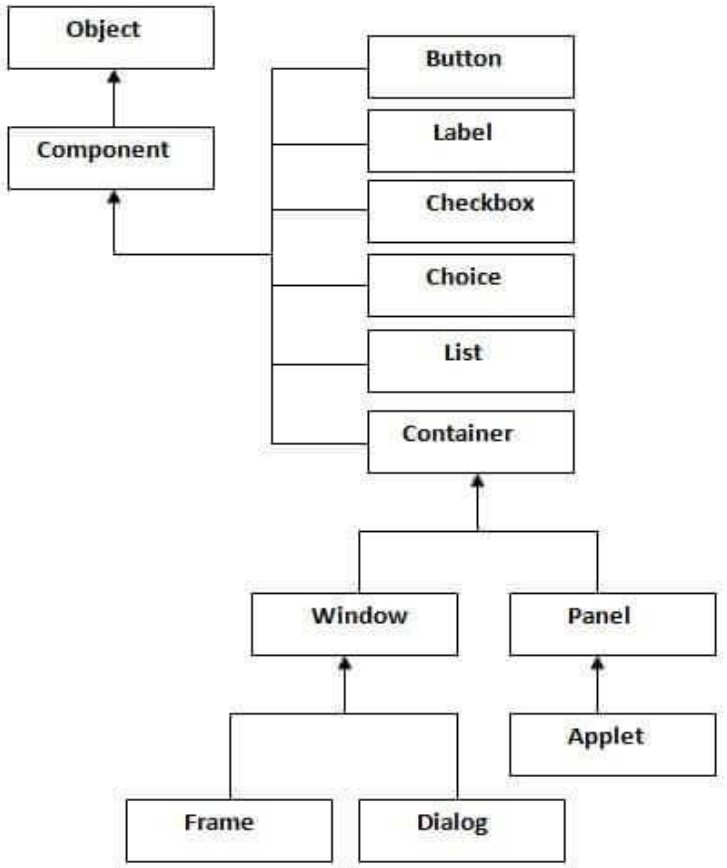
- **Java AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

- The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

# Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



# Java AWT Tutorial

**Container:** The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

**Window:** The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

# Java AWT Tutorial...

**Panel:** The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

**Frame:** The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

# Useful Methods of Component class

| Method | Description |
| --- | --- |
| public void add(Component c) | inserts a component on this component. |
| public void setSize(int width,int height) | sets the size (width and height) of the component. |
| public void setLayout(LayoutManager m) | defines the layout manager for the component. |
| public void setVisible(boolean status) | changes the visibility of the component, by default false. |

# Java Swing Tutorial

- **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

| 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**. |

# Pluggable look and feel

- Pluggable look and feel is a mechanism used in the Java Swing widget toolkit allowing to change the look and feel of the graphical user interface at runtime.

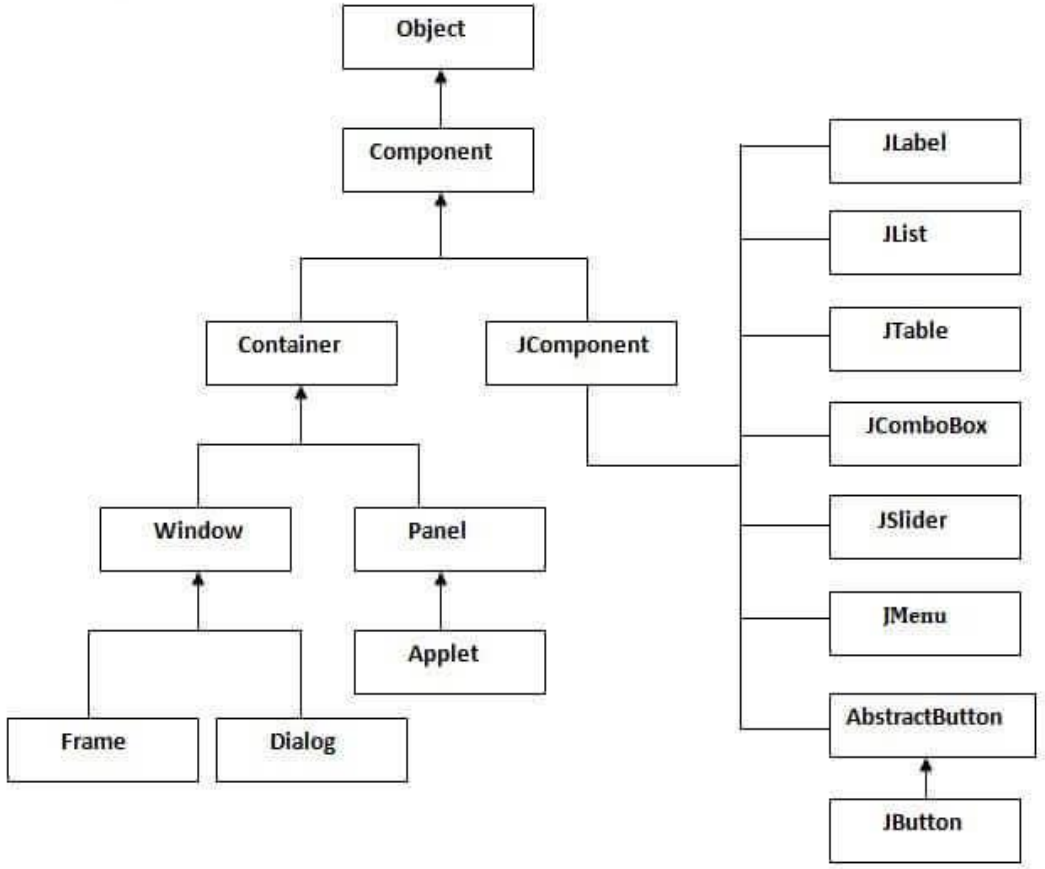
- The **look and feel** can be changed at runtime.

# What is JFC ?

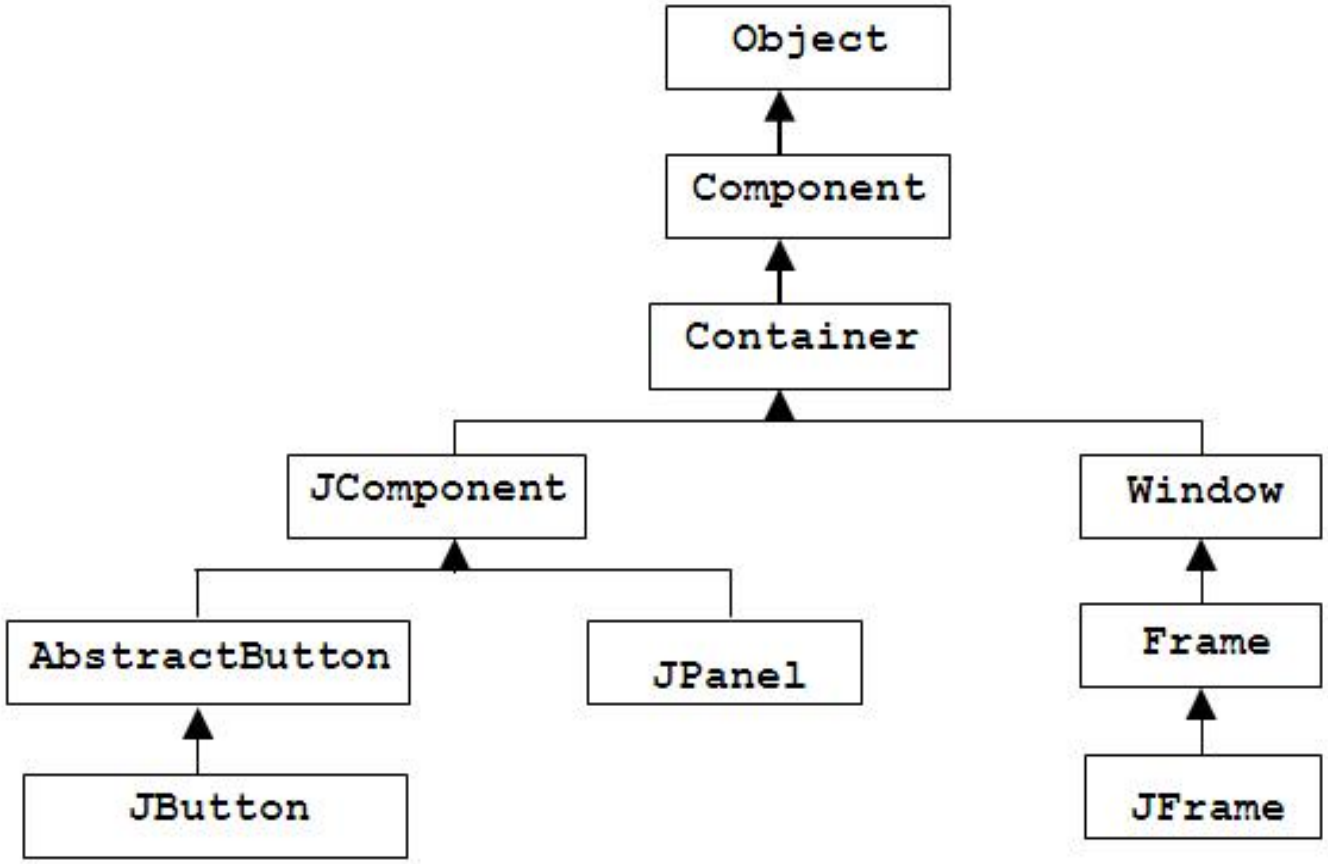
The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

# Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



# 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 the size (width and height) of the component. |
| public void setLayout(LayoutManager m) | set the layout manager for the component. |
| public void setVisible(boolean status) | changes the visibility of the component. It is by default false. |

# GUI Creation Steps

1. Import required packages.
2. Setup top level container.
3. Get the component area of the top level container.
4. Apply layout to component area.
5. Create and add components.
6. Set size of frame and make it visible.

# Java Swing Examples

There are two ways to create a frame:

1. By creating the object of Frame class (association)
2. By extending Frame class (inheritance)

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

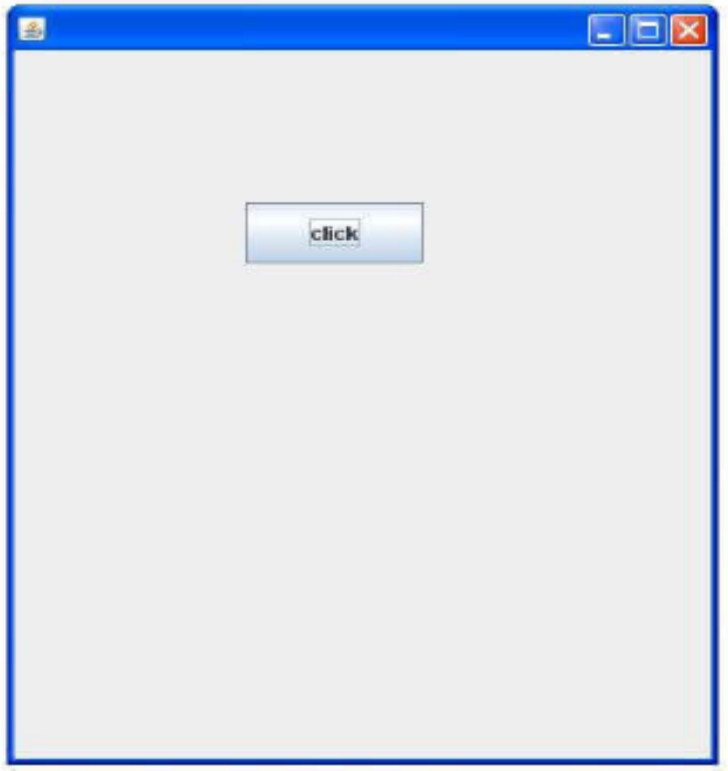
# Simple Java Swing Example

Let's see a simple swing example where 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
```

# Simple Java Swing Example…

```
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
}
}
```



# Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

```
import javax.swing.*;

public class Simple {

JFrame f;

Simple() {     // Constructor Body Start

f=new JFrame();        //creating instance of JFrame
```

# Example of Swing by Association inside constructor…

```
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
}  //  constructor body end
```

# Example of Swing by Association inside constructor…

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

The setBounds(int xaxis, int yaxis, int width, int height)is used in the above example that sets the position of the button.

# 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

JFrame f;

Simple2(){

JButton b=new JButton("click");          //create button

b.setBounds(130,100,100, 40);
```

# Simple example of Swing by inheritance

```
add(b);         //adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}    //constructor body end
public static void main(String[] args) {
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.

## JButton class declaration

Let's see the declaration for javax.swing.JButton class.

***public class*** *JButton* ***extends*** *AbstractButton* ***implements*** *Accessible*

# Commonly used Constructors in JButton

| 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. |

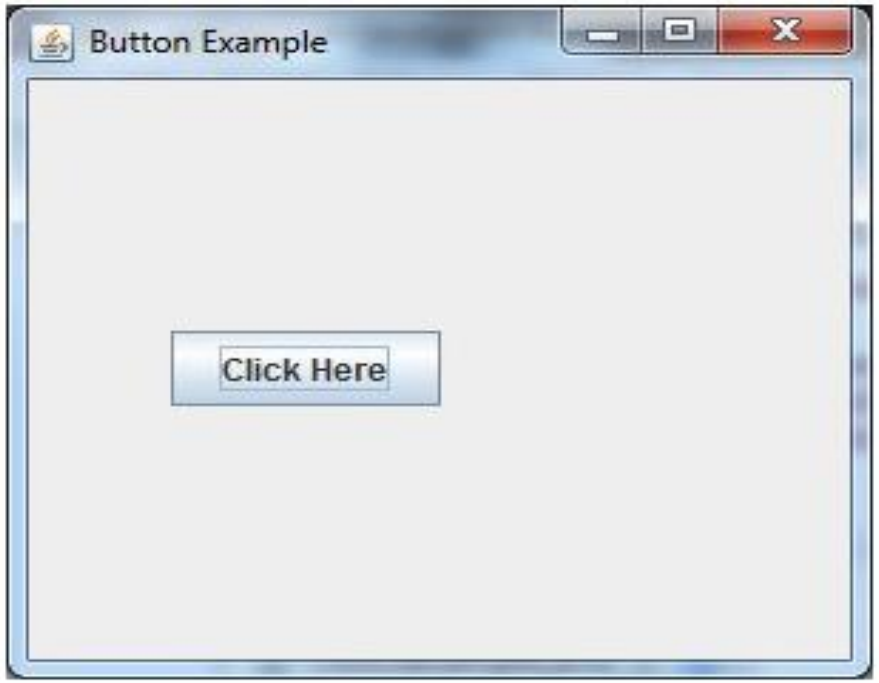
# Commonly used Methods of AbstractButton class

| 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 action listener to this object. |

# Java JButton Example

```
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);
}
}
```

Output:



# 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.

## JLabel class declaration

Let's see the declaration for javax.swing.JLabel class.

***public class*** *JLabel* ***extends*** *JComponent* ***implements*** *SwingConstants, Accessible*

# Commonly used Constructors in JLabel

| 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 in JLabel

| 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);
```

# Java JLabel Example...

```
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);
    }
    }
```

# 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

Let's see the declaration for javax.swing.JTextField class.

***public class*** *JTextField* ***extends*** *JTextComponent* ***implements*** *SwingConstants*

# Commonly used constructors in JTextField

| 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 in JTextField…

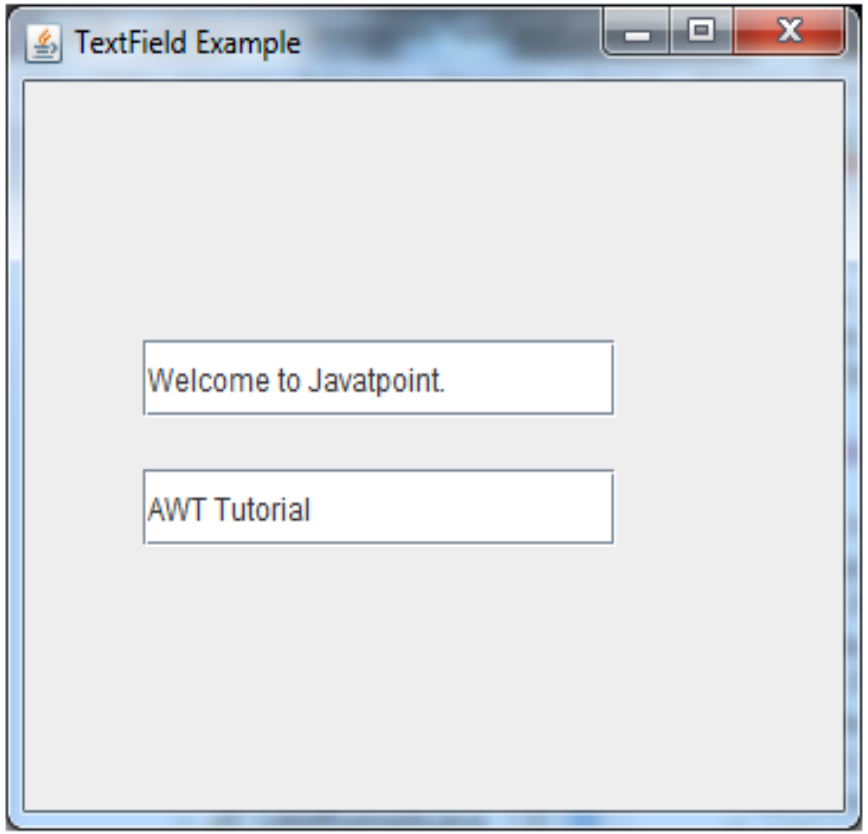
| Methods | Description |
| --- | --- |
| void addActionListener(ActionListener l) | It is used to add the specified action listener to receive action events from this textfield. |
| Action getAction() | It returns the currently set Action for this ActionEvent source, or null if no Action is set. |
| 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 FAST NUCES Peshawar");
    t1.setBounds(50,100, 200,30);
```

# Java JTextField Example...

```
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);
}
}
```



# GUI Creation Steps

# 1) import required packages

import java.awt.*  and/or javax.swing.* package.

# 2) Setup the top level containers

- A container is a collection of related components, which allows other components to be nested inside it. In application with JFrame, we attach components to the content pane - a container.

- Two important methods the container class has **add** and **setLayout.**

- The add method is used for adding components to the content pane while setLayout method is used to specify the layout manager.

# 2) Setup the top level containers…

- Container are classified into two broad categories that are Top Level Containers and General Purpose Containers.

- Top level containers can contain (add) other containers as well as basic Components (buttons, labels etc) while general purpose containers are typically used to collect basic components and are added to top level containers.

- General purpose containers cannot exist alone they must be added to top level containers.

# 2) Setup the top level containers…

- Examples of top level container are JFrame, Dialog and Applet etc. Our application uses one of these.
- Examples of general purpose container are JPanel, Toolbar and ScrollPane etc.
- So, take a top level container and create its instance. Consider the following code of line if JFrame is selected as a top level container

```
JFrame frame = new JFrame();
```

# 3) Get the component area of the top level container

- Review the hierarchy given above, and observe that JFrame *is a* frame *is a* window. So, it can be interpreted as JFrame *is a* window.

- Every window has two areas. System Area & Component Area.

- The programmer cannot add/remove components to the System Area.

## 3) Get the component area of the top level container…

- The Component Area often known as Client area is a workable place for the
- Programmer Components can be added/removed in this area.
- So, to add components, as you guessed right component area of the JFrame is required. It can be accomplished by the following code of line.

## 3) Get the component area of the top level container…

Container *con* = *frame*.getContentPane();

- *frame* is an instance of JFrame and by calling *getContentPane()* method on it, it returns the component area.

- This component area is of type container and that is why it is stored in a variable of a Container class.

- As already discussed, container allows other components to be added / removed.

# 4) Apply layout to component area

- The layout (size & position etc. How they appear) of components in a container is usually governed by Layout Managers.

- The layout manager is responsible for deciding the layout policy and size of its components added to the container.

- Layout managers are represented in java as classes. (Layout Managers are going to be discussed in detail later).

# 4) Apply layout to component area…

- To set the layout, as already discussed use setLayout method and pass object of layout manager as an argument.

```
con.setLayout( new FlowLayout( ) );
```

- We passed an object of FlowLayout to the setLayout method here.

- We can also use the following lines of code instead of above.

```
FlowLayout layout = new FlowLayout();
con.setLayout(layout);
```

# 5) Create and Add components

- Create required components by calling their constructor.

```
JButton button = new JButton ( );
```

- After creating all components you are interested in is to add these components into the component area of your JFrame (i.e ContentPane, the reference to which is in variable con of type Container).

# 5) Create and Add components…

- Use *add* method of the Container to accomplish this and pass it the component to be added.

```
con.add(button);
```

# 6) Set size of frame and make it visible

- A frame must be made visible via a call to setVisible(true) and its size defined via a call setSize(rows in pixel, columns in pixel) to be displayed on the screen.

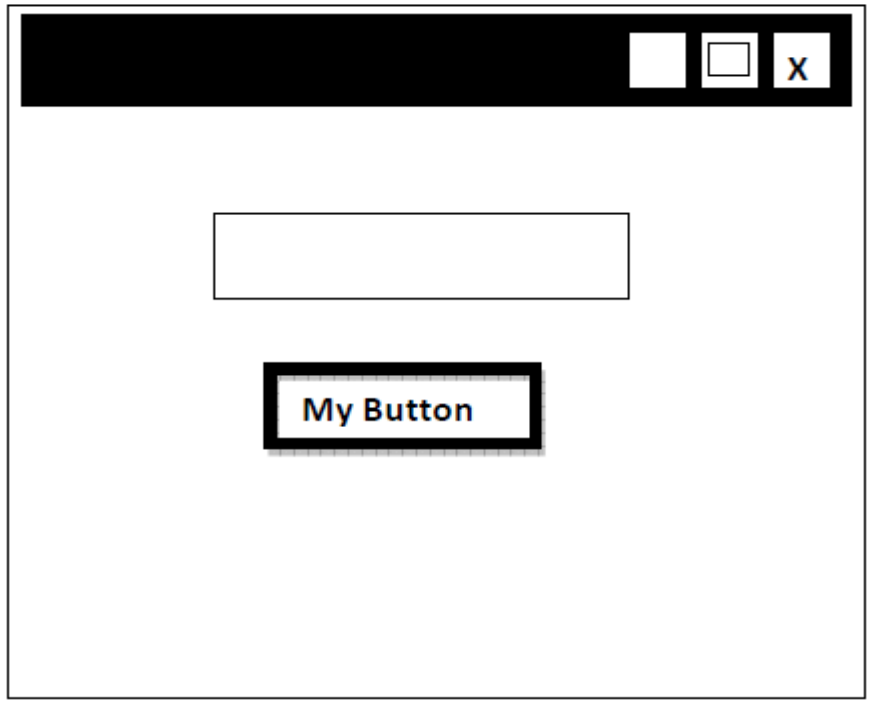
```
frame.setSize(200,300) ;
frame.setVisible(true) ;
```

**Note:** By default, all JFrame's are invisible. To make visible frame visible we have passed *true* to the setVisible method.

```
frame.setVisible(false) ;
```

# Making a Simple GUI using the previous steps

The below figured GUI contains one text field and a button. Let's code it by following the six GUI creation steps we discussed.



# Code for Simple GUI

```
// File GUITest.java
//Step 1: import packages
import java.awt.*;
import javax.swing.*;
public class GUITest {
JFrame myFrame ;
public void initGUI ( ) {          //initGUI() method used for setting layout of GUI
//Step 2: setup the top level container
myFrame = new JFrame();
```

# Code for Simple GUI…

```
//Step 3: Get the component area of top-level container
Container c = myFrame.getContentPane();
//Step 4: Apply layouts
c.setLayout( new FlowLayout( ) );
//Step 5: create & add components
JTextField tf = new JTextField(10);
JButton b1 = new JButton("My Button");
c.add(tf);
c.add(b1);
```

# Code for Simple GUI…

```
//Step 6: set size of frame and make it visible
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
myFrame.setSize(200,150);
myFrame.setVisible(true);
}     //end initGUI method
```

# Code for Simple GUI…

```
public GUITest () {      // default constructor
initGUI ();
}
public static void main (String args[ ]) {
GUITest gui = new GUITest();
}
} // end of class
```

# Important Points to Consider

- *main* method (from where program execution starts) is written in the same class. The main method can be in a separate class instead of writing in the same class its your choice.

- Inside main, an object of GUI test class is created that results in calling of constructor of the class and from the constructor, *initGUI* method is called that is responsible for setting up the GUI.

# Important Points to Consider…

- The following line of code is used to exit the program when you close the window

*myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);*

# Important Points to Consider…

- If you delete this line and run your program, the desired GUI would be displayed.
- However if you close the window by using (X) button on top left corner of your window, you'll notice that the control doesn't return back to command prompt. The reason for this is that the java process is still running. However if you put this line in your code, when you exit your prompt will return.

# 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.

Let's see the declaration for javax.swing.JTextArea class.

**public class** JTextArea **extends** JTextComponent

# Java JTextArea...

**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. |

# Java JTextArea...

## Commonly used Methods:

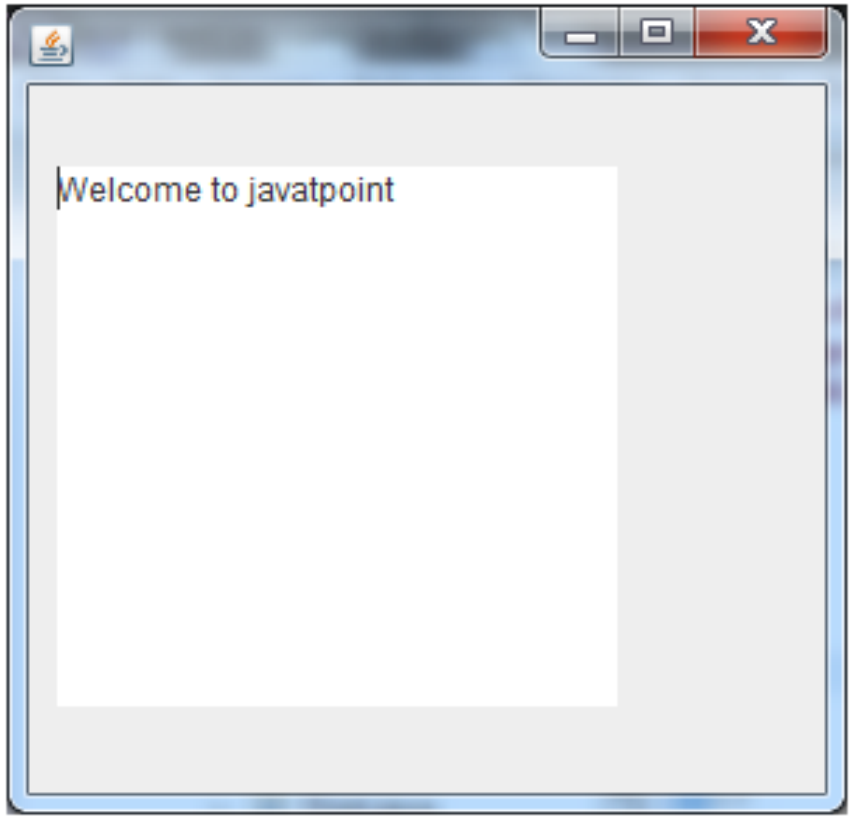
| Methods | Description |
|---|---|
| void setRows(int rows) | It is used to set specified number of rows. |
| void setColumns(int cols) | It is used to set specified number of columns. |
| void setFont(Font f) | It is used to set the specified font. |
| void insert(String s, int position) | It is used 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. |

# Java 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);
    }
```

# Java JTextArea Example...

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



# 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.

## JCheckBox class declaration

Let's see the declaration for javax.swing.JCheckBox class.

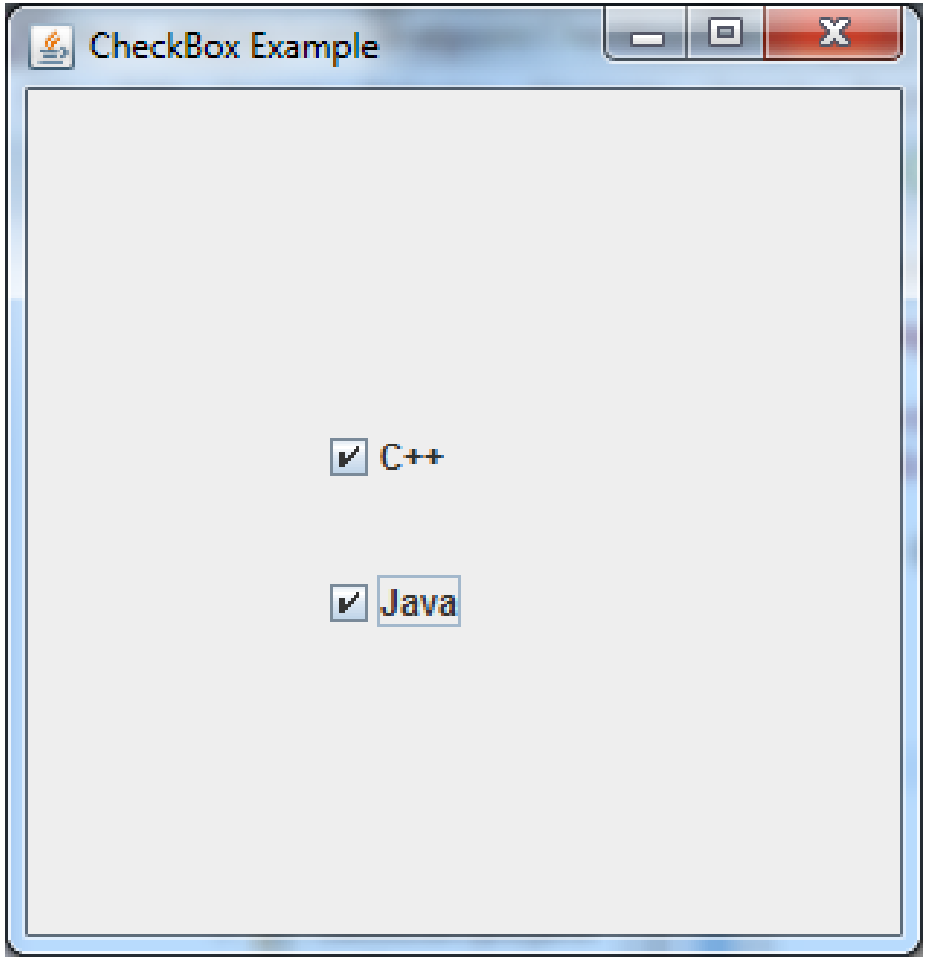
**public class** JCheckBox **extends** JToggleButton **implements** Accessible

# Java JCheckBox...

```
package java_gui_work;


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, 70,70);
```

# Java JCheckBox...

```
 f.add(checkBox1);
  f.add(checkBox2);
  f.setSize(400,400);
  f.setLayout(null);
  f.setVisible(true);
}
```



# Java JCheckBox...

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

# Java JRadioButtons

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**

Let's see the declaration for javax.swing.JRadioButton class.

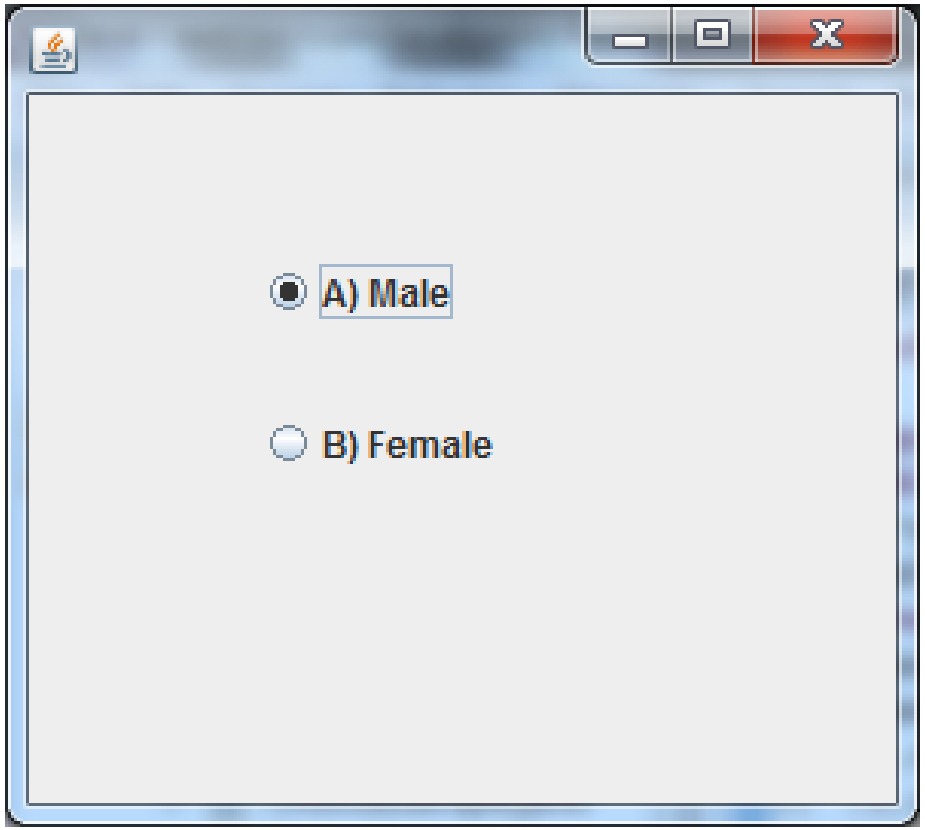
**public class** JRadioButton **extends** JToggleButton **implements** Accessible

# Java JRadioButtons...

```
package java_gui_work;
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();
```

# Java JRadioButtons...

```
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 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**

Let's see the declaration for javax.swing.JComboBox class.

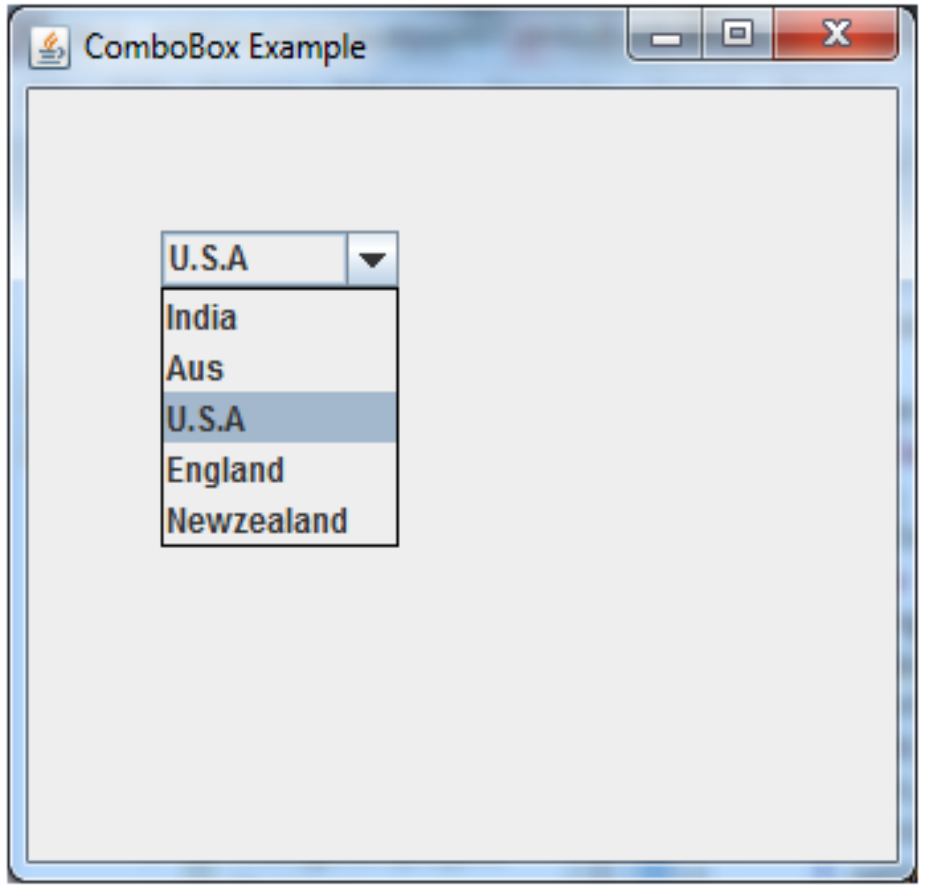
**public class** JComboBox **extends** JComponent **implements** ItemSelectable, ListD ataListener, ActionListener, Accessible

# Java JComboBox…

```
package java_gui_work;
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);
```

# Java JComboBox...

```
f.add(cb);
    f.setLayout(null);
    f.setSize(400,500);
    f.setVisible(true);
}
public static void main(String[] args) {
    new ComboBoxExample();
}
}
```



# Home Work

1) Java Jtable

(https://www.javatpoint.com/java-jtable)

2) Java Jlist

(javatpoint.com/java-jlist)

3) Java JOptionPane

(https://www.javatpoint.com/java-jlist)

4) Java JScrollBar

(https://www.javatpoint.com/java-jscrollbar)

5) Java JMenuBar, JMenu and JMenuItem

(https://www.javatpoint.com/java-jmenuitem-and-jmenu)

6) Java JPopupMenu

(https://www.javatpoint.com/java-jpopupmenu)

# Home Work

7) Java JCheckBoxMenuItem

(https://www.javatpoint.com/java-jcheckboxmenuitem)

# References

https://www.javatpoint.com/java-swing

# THANK YOU