AWT

1. Class hierarchy
2. User interface components

   Creation of Frame

   a) Labels

   b) Buttons

   c) Scrollbars

   d) Text components

   - Text Field
   - Text Area

   e) Checkbox

   f) Checkbox groups

   g) Choices

   h) Lists

   i) Panels

   j) Scroll pane

   k) Dialogs

   l) Menu bar

AWT(Abstract Window Toolkit):-

Java AWT (Abstract Window Toolkit) is *an API to develop Graphical User Interface (GUI) or windows-based applications* in Java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

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

Why AWT is platform dependent?

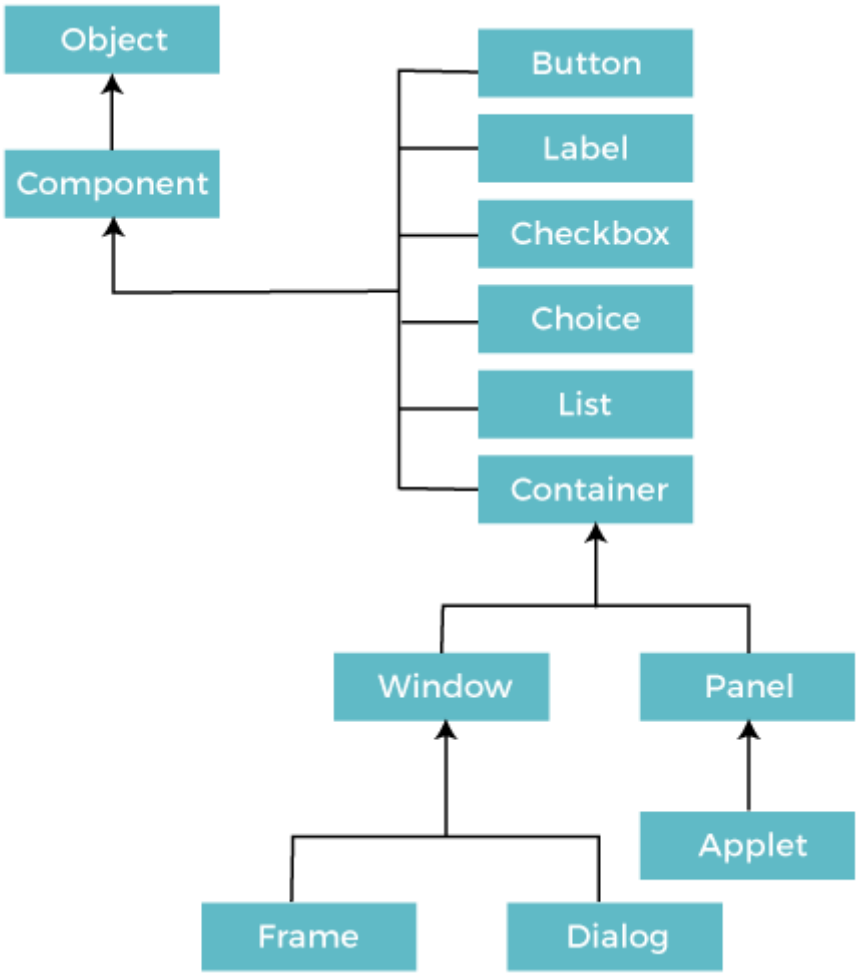
Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, CheckBox, button, etc.

For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.

In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

1.Class hierarchy

The hierarchy of Java AWT classes are given below.



Components

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

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**.

It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

Note: A container itself is a component (see the above diagram), therefore we can add a container inside container.

Types of containers:

There are four types of containers in Java AWT:

1. Window
2. Panel
3. Frame
4. Dialog

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. We need to create an instance of Window class to create this container.

Panel

The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

Frame

The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

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 AWT Example Frame

To create simple AWT example, you need a frame. There are two ways to create a GUI using Frame in AWT.

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

AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

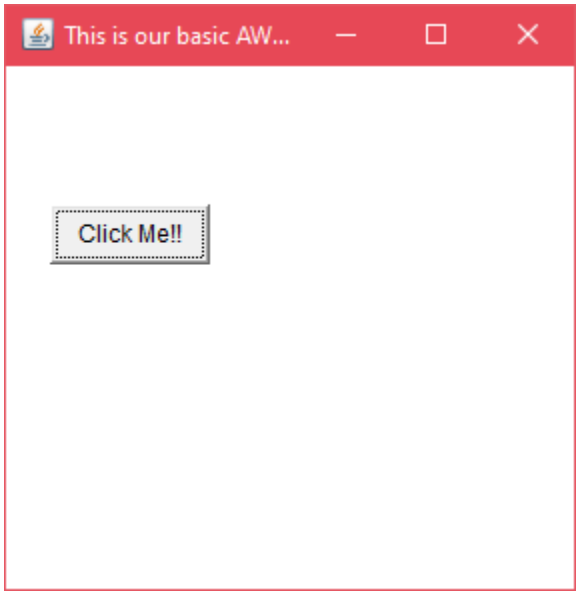
AWTExample1.java

```
1. // importing Java AWT class
2. import java.awt.*;
3. 
4. // extending Frame class to our class AWTExample1
5. public class AWTExample1 extends Frame {
6. 
7.   // initializing using constructor
8.   AWTExample1() {
9. 
10.     // creating a button
11.     Button b = new Button("Click Me!!");
12. 
13.     // setting button position on screen
14.     b.setBounds(30,100,80,30);
15. 
16.     // adding button into frame
17.     add(b);
18. 
19.     // frame size 300 width and 300 height
20.     setSize(300,300);
21. 
22.     // setting the title of Frame
23.     setTitle("This is our basic AWT example");
```

```
24.
25.     // no layout manager
26.     setLayout(null);
27.
28.     // now frame will be visible, by default it is not visible
29.     setVisible(true);
30. }
31.
32. // main method
33. public static void main(String args[]) {
34.
35. // creating instance of Frame class
36. AWTExample1 f = new AWTExample1();
37.
38. }
39.
40. }
```

The setBounds(int x-axis, int y-axis, int width, int height) method is used in the above example that sets the position of the awt button.

Output:



AWT Example by Association

Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are creating a TextField, Label and Button component on the Frame.

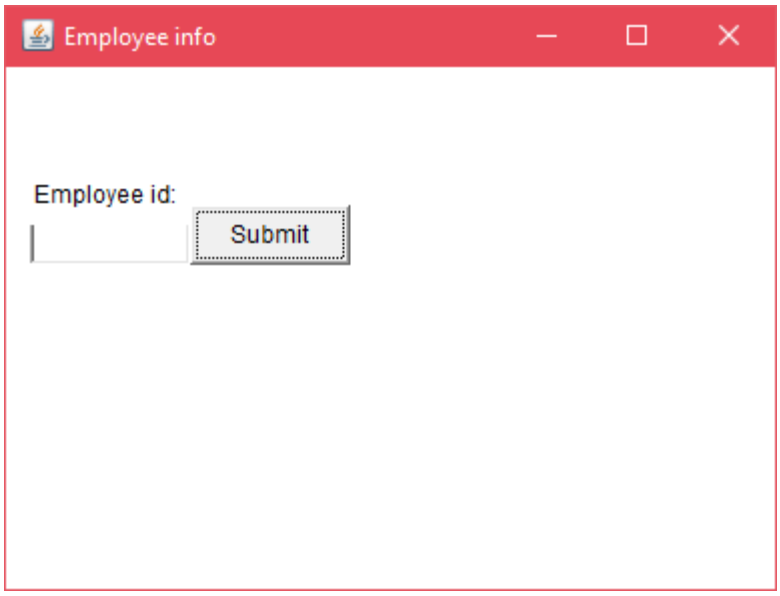
AWTExample2.java

```
1. // importing Java AWT class
2. import java.awt.*;
```

```
3.
4.  // class AWTExample2 directly creates instance of Frame class
5.  class AWTExample2 {
6.
7.     // initializing using constructor
8.     AWTExample2() {
9.
10.       // creating a Frame
11.       Frame f = new Frame();
12.
13.       // creating a Label
14.       Label l = new Label("Employee id:");
15.
16.       // creating a Button
17.       Button b = new Button("Submit");
18.
19.       // creating a TextField
20.       TextField t = new TextField();
21.
22.       // setting position of above components in the frame
23.       l.setBounds(20, 80, 80, 30);
24.       t.setBounds(20, 100, 80, 30);
25.       b.setBounds(100, 100, 80, 30);
26.
27.       // adding components into frame
28.       f.add(b);
29.       f.add(l);
30.       f.add(t);
31.
32.       // frame size 300 width and 300 height
33.       f.setSize(400,300);
34.
35.       // setting the title of frame
36.       f.setTitle("Employee info");
37.
38.       // no layout
39.       f.setLayout(null);
40.
41.       // setting visibility of frame
42.       f.setVisible(true);
43. }
```

```
44.
45. // main method
46. public static void main(String args[]) {
47.
48. // creating instance of Frame class
49. AWTExample2 awt_obj = new AWTExample2();
50.
51. }
52.
53. }
```

Output:



3. User interface components

a)Labels

The object of the Label 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 a programmer but a user cannot edit it directly.

It is called a passive control as it does not create any event when it is accessed. To create a label, we need to create the object of **Label** class.

AWT Label Class Declaration

1. **public class** Label **extends** Component **implements** Accessible

AWT Label Fields

The java.awt.Component class has following fields:

1. **static int LEFT:** It specifies that the label should be left justified.
2. **static int RIGHT:** It specifies that the label should be right justified.
3. **static int CENTER:** It specifies that the label should be placed in center.

Label class Constructors

Sr. no.	Constructor	Description
1.	Label()	It constructs an empty label.
2.	Label(String text)	It constructs a label with the given string (left justified by default).
3.	Label(String text, int alignement)	It constructs a label with the specified string and the specified alignment.

Label Class Methods

Specified

Sr. no.	Method name	Description
1.	void setText(String text)	It sets the texts for label with the specified text.
2.	Void setAlignment(int alignment)	It sets the alignment for label with the specified alignment.
3.	String getText()	It gets the text of the label
4.	int getAlignment()	It gets the current alignment of the label.

Method inherited

The above methods are inherited by the following classes:

- java.awt.Component
- java.lang.Object

Java AWT Label Example

In the following example, we are creating two labels l1 and l2 using the Label(String text) constructor and adding them into the frame.

LabelExample.java

```
1. import java.awt.*;
2.
3. public class LabelExample {
4. public static void main(String args[]){
5.
6.     // creating the object of Frame class and Label class
7.     Frame f = new Frame ("Label example");
8.     Label l1, l2;
9.
10.    // initializing the labels
11.    l1 = new Label ("First Label.");
12.    l2 = new Label ("Second Label.");
13.
14.    // set the location of label
15.    l1.setBounds(50, 100, 100, 30);
16.    l2.setBounds(50, 150, 100, 30);
17.
18.    // adding labels to the frame
19.    f.add(l1);
20.    f.add(l2);
21.
22.    // setting size, layout and visibility of frame
23.    f.setSize(400,400);
24.    f.setLayout(null);
25.    f.setVisible(true);
26. }
27. }
```

Output:

b)Buttons

Java AWT Button

A button is basically a control component with a label that generates an event when pushed. The **Button** class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

When we press a button and release it, AWT sends an instance of **ActionEvent** to that button by calling **processEvent** on the button. The **processEvent** method of the button receives the all the events, then it passes an action event by calling its own method **processActionEvent**. This method passes the action event on to action listeners that are interested in the action events generated by the button.

To perform an action on a button being pressed and released, the **ActionListener** interface needs to be implemented. The registered new listener can receive events from the button by calling **addActionListener** method of the button.

AWT Button Class Declaration

1. **public class** Button **extends** Component **implements** Accessible

Button Class Constructors

Following table shows the types of Button class constructors

Sr. no.	Constructor	Description
1.	Button( )	It constructs a new button with an empty string i.e. it has no label.
2.	Button (String text)	It constructs a new button with given string as its label.

Button Class Methods

Sr. no.	Method	Description
1.	void setText (String text)	It sets the string message on the button
2.	String getText()	It fetches the String message on the button.
3.	void setLabel (String label)	It sets the label of button with the specified string.

4.	String getLabel()	It fetches the label of the button.
7.	void addActionListener(ActionListener l)	It adds the specified action listener to get the action events from the button.
8.	String getActionCommand()	It returns the command name of the action event fired by the button.
9.	void removeActionListener (ActionListener l)	It removes the specified action listener so that it no longer receives action events from the button.
10.	void setActionCommand(String command)	It sets the command name for the action event given by the button.

Note: The Button class inherits methods from java.awt.Component and java.lang.Object classes.

Java AWT Button Example

Example 1:

ButtonExample.java

```
1. import java.awt.*;
2. public class ButtonExample {
3. public static void main (String[] args) {
4.
5.     // create instance of frame with the label
6.     Frame f = new Frame("Button Example");
7.
8.     // create instance of button with label
9.     Button b = new Button("Click Here");
10.
11.    // set the position for the button in frame
12.    b.setBounds(50,100,80,30);
13.
14.    // add button to the frame
15.    f.add(b);
16.    // set size, layout and visibility of frame
17.    f.setSize(400,400);
18.    f.setLayout(null);
19.    f.setVisible(true);
20. }
21. }
```

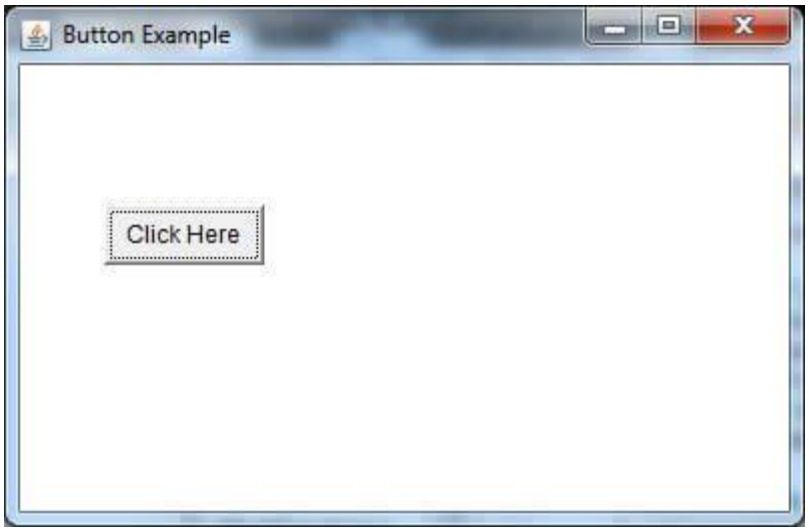
To compile the program using command prompt type the following commands

1. C:\Users\Anurati\Desktop\abcDemo>javac ButtonExample.java

If there's no error, we can execute the code using:

1. C:\Users\Anurati\Desktop\abcDemo>java ButtonExample

Output:



c)Scrollbars

The object of Scrollbar class is used to add horizontal and vertical scrollbar. Scrollbar is a GUI component allows us to see invisible number of rows and columns.

It can be added to top-level container like Frame or a component like Panel. The Scrollbar class extends the **Component** class.

AWT Scrollbar Class Declaration

1. **public class** Scrollbar **extends** Component **implements** Adjustable, Accessible

Scrollbar Class Fields

The fields of java.awt.Image class are as follows:

- **static int HORIZONTAL** - It is a constant to indicate a horizontal scroll bar.
- **static int VERTICAL** - It is a constant to indicate a vertical scroll bar.

Scrollbar Class Constructors

Sr. no.	Constructor	Description
1	Scrollbar()	Constructs a new vertical scroll bar.
2	Scrollbar(int orientation)	Constructs a new scroll bar with the specified orientation.
3	Scrollbar(int orientation, int value, int visible, int minimum, int maximum)	Constructs a new scroll bar with the specified orientation, initial value, visible amount, and minimum and maximum values.

Where the parameters,

- **orientation**: specifiey whether the scrollbar will be horizontal or vertical.
- **Value**: specify the starting position of the knob of Scrollbar on its track.
- **Minimum**: specify the minimum width of track on which scrollbar is moving.
- **Maximum**: specify the maximum width of track on which scrollbar is moving.

Method Inherited by Scrollbar

The methods of Scrollbar class are inherited from the following classes:

- java.awt.Component
- java.lang.Object

Scrollbar Class Methods

Sr. no.	Method name	Description
1.	int getMaximum()	It gets the maximum value of the scroll bar.
2.	int getMinimum()	It gets the minimum value of the scroll bar.
3.	int getOrientation()	It returns the orientation of scroll bar.
4.	int getUnitIncrement()	It fetches the unit increment of the scroll bar.
5.	int getValue()	It fetches the current value of scroll bar.
6.	int getVisibleAmount()	It fetches the visible amount of scroll bar.

7.	void setBlockIncrement(int v)	It sets the block increment from scroll bar.
8.	void setMaximum (int newMaximum)	It sets the maximum value of the scroll bar.
9.	void setMinimum (int newMinimum)	It sets the minimum value of the scroll bar.
10.	void setOrientation (int orientation)	It sets the orientation for the scroll bar.
11.	void setUnitIncrement(int v)	It sets the unit increment for the scroll bar.
12.	void setValue (int newValue)	It sets the value of scroll bar with the given argument value.
13.	void setValues (int value, int visible, int minimum, int maximum)	It sets the values of four properties for scroll bar: value, visible amount, minimum and maximum.
14.	void setVisibleAmount (int newAmount)	It sets the visible amount of the scroll bar.

Java AWT Scrollbar Example

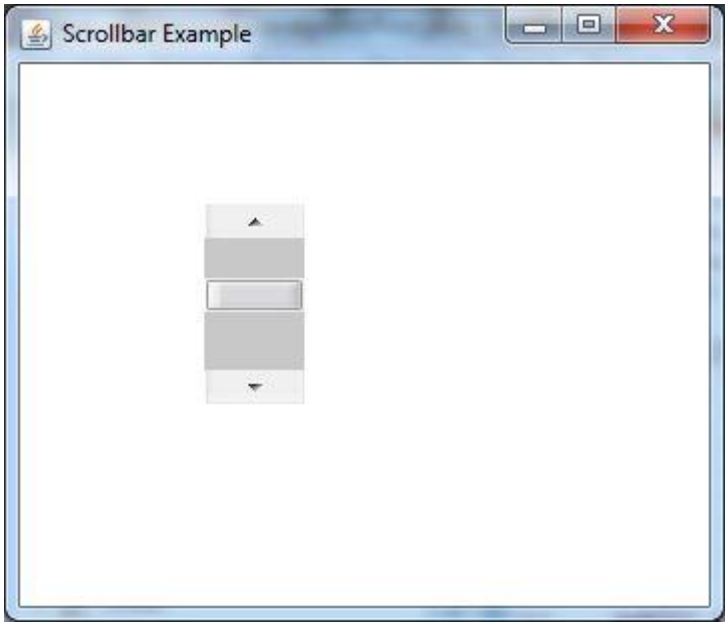
In the following example, we are creating a scrollbar using the Scrollbar() and adding it into the Frame.

ScrollbarExample1.java

```
1. // importing awt package
2. import java.awt.*;
3.
4. public class ScrollbarExample1 {
5.
6. // class constructor
7. ScrollbarExample1() {
8.
9.       // creating a frame
10.          Frame f = new Frame("Scrollbar Example");
11.      // creating a scroll bar
12.          Scrollbar s = new Scrollbar();
13.
14.      // setting the position of scroll bar
15.          s.setBounds (100, 100, 50, 100);
16.
17.      // adding scroll bar to the frame
18.          f.add(s);
19.
20.      // setting size, layout and visibility of frame
```

```
21.             f.setSize(400, 400);
22.             f.setLayout(null);
23.             f.setVisible(true);
24. }
25.
26. // main method
27. public static void main(String args[]) {
28.         new ScrollbarExample1();
29. }
30. }
```

Output:



d)Text components

Text Components class is extends by

1.TextField

2.TextArea

1.TextField

The object of a **TextField** class is a text component that allows a user to enter a single line text and edit it. It inherits **TextComponent** class, which further inherits **Component** class.

When we enter a key in the text field (like key pressed, key released or key typed), the event is sent to **TextField**. Then the **KeyEvent** is passed to the registered **KeyListener**. It can also be done using **ActionEvent**; if the ActionEvent is enabled on the text field, then the ActionEvent may be fired by pressing return key. The event is handled by the **ActionListener** interface.

AWT TextField Class Declaration

1. **public class** TextField **extends** TextComponent

TextField Class constructors

Sr. no.	Constructor	Description
1.	TextField()	It constructs a new text field component.
2.	TextField(String text)	It constructs a new text field initialized with the given string text to be displayed.
3.	TextField(int columns)	It constructs a new textfield (empty) with given number of columns.
4.	TextField(String text, int columns)	It constructs a new text field with the given text and given number of columns (width).

TextField Class Methods

Sr. no.	Method name	Description
1.	boolean echoCharIsSet()	It tells whether text field has character set for echoing or not.
2.	int getColumns()	It fetches the number of columns in text field.
3.	char getEchoChar()	It fetches the character that is used for echoing.
4.	Dimension getMinimumSize()	It fetches the minimum dimensions for the text field.
5.	Dimension getMinimumSize(int columns)	It fetches the minimum dimensions for the text field with specified number of columns.
6.	Dimension getPreferredSize()	It fetches the preferred size of the text field.
7.	Dimension getPreferredSize(int columns)	It fetches the preferred size of the text field with specified number of columns.
8.	void setColumns(int columns)	It sets the number of columns in text field.
9.	void setEchoChar(char c)	It sets the echo character for text field.

10.	void setText(String t)	It sets the text presented by this text component to the specified text.

Method Inherited

The AWT TextField class inherits the methods from below classes:

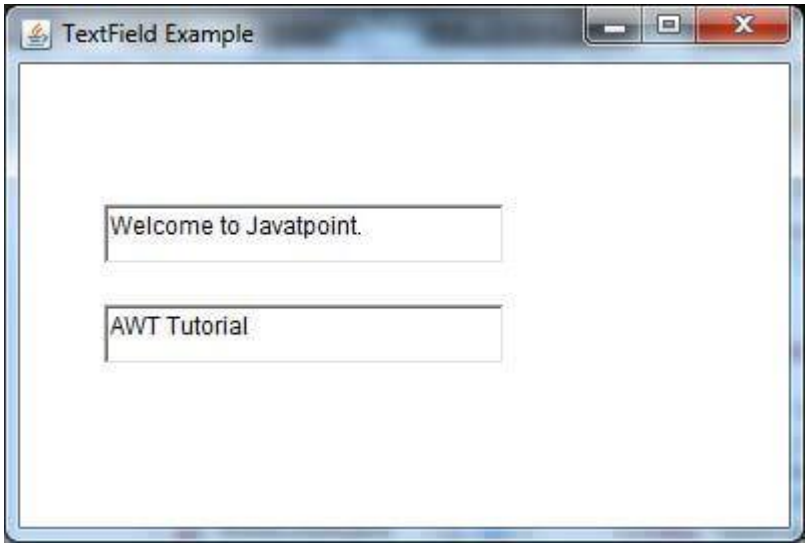
1. java.awt.TextComponent
2. java.awt.Component
3. java.lang.Object

Java AWT TextField Example

TextFieldExample1.java

```
1. // importing AWT class
2. import java.awt.*;
3. public class TextFieldExample1 {
4.     // main method
5.     public static void main(String args[]) {
6.     // creating a frame
7.     Frame f = new Frame("TextField Example");
8.
9.     // creating objects of textfield
10.    TextField t1, t2;
11.    // instantiating the textfield objects
12.    // setting the location of those objects in the frame
13.    t1 = new TextField("Welcome to Javatpoint.");
14.    t1.setBounds(50, 100, 200, 30);
15.    t2 = new TextField("AWT Tutorial");
16.    t2.setBounds(50, 150, 200, 30);
17.    // adding the components to frame
18.    f.add(t1);
19.    f.add(t2);
20.    // setting size, layout and visibility of frame
21.    f.setSize(400,400);
22.    f.setLayout(null);
23.    f.setVisible(true);
24. }
25. }
```

Output:



2.TextArea

The object of a TextArea class is a multiline region that displays text. It allows the editing of multiple line text. It inherits TextComponent class.

The text area allows us to type as much text as we want. When the text in the text area becomes larger than the viewable area, the scroll bar appears automatically which helps us to scroll the text up and down, or right and left.

AWT TextArea Class Declaration

1. **public class** TextArea **extends** TextComponent

Fields of TextArea Class

The fields of java.awt.TextArea class are as follows:

- **static int SCROLLBARS_BOTH** - It creates and displays both horizontal and vertical scrollbars.
- **static int SCROLLBARS_HORIZONTAL_ONLY** - It creates and displays only the horizontal scrollbar.
- **static int SCROLLBARS_VERTICAL_ONLY** - It creates and displays only the vertical scrollbar.
- **static int SCROLLBARS_NONE** - It doesn't create or display any scrollbar in the text area.

Class constructors:

Sr. no.	Constructor	Description
1.	TextArea()	It constructs a new and empty text area with no text in it.

2.	TextArea (int row, int column)	It constructs a new text area with specified number of rows and columns and empty string as text.
3.	TextArea (String text)	It constructs a new text area and displays the specified text in it.
4.	TextArea (String text, int row, int column)	It constructs a new text area with the specified text in the text area and specified number of rows and columns.
5.	TextArea (String text, int row, int column, int scrollbars)	It construcst a new text area with specified text in text area and specified number of rows and columns and visibility.

Methods Inherited

The methods of TextArea class are inherited from following classes:

- java.awt.TextComponent
- java.awt.Component
- java.lang.Object

TetArea Class Methods

Sr. no.	Method name	Description
1.	int getColumns()	It returns the number of columns of text area.
2.	Dimension getMinimumSize()	It determines the minimum size of a text area.
3.	Dimension getMinimumSize(int rows, int columns)	It determines the minimum size of a text area with the given number of rows and columns.
4.	Dimension getPreferredSize()	It determines the preferred size of a text area.
5.	Dimension preferredSize(int rows, int columns)	It determines the preferred size of a text area with given number of rows and columns.
6.	int getRows()	It returns the number of rows of text area.
7.	int getScrollbarVisibility()	It returns an enumerated value that indicates which scroll bars the text area uses.

8.	void insert(String str, int pos)	It inserts the specified text at the specified position in this text area.
9.	void replaceRange(String str, int start, int end)	It replaces text between the indicated start and end positions with the specified replacement text.
10.	void setColumns(int columns)	It sets the number of columns for this text area.
11.	void setRows(int rows)	It sets the number of rows for this text area.

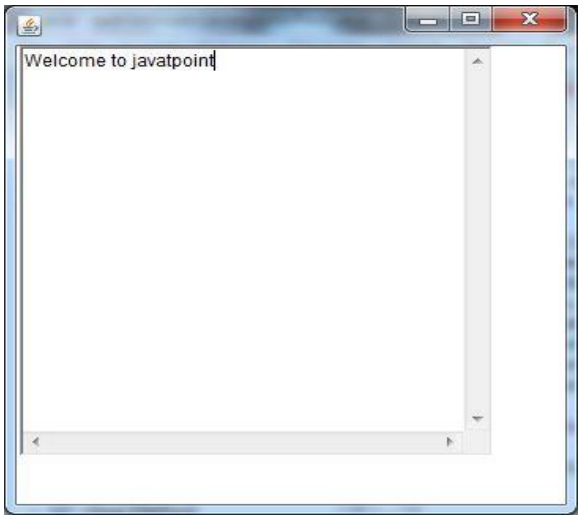
Java AWT TextArea Example

The below example illustrates the simple implementation of TextArea where we are creating a text area using the constructor TextArea(String text) and adding it to the frame.

TextAreaExample .java

```
1. //importing AWT class
2. import java.awt.*;
3. public class TextAreaExample
4. {
5. // constructor to initialize
6.     TextAreaExample() {
7. // creating a frame
8.         Frame f = new Frame();
9. // creating a text area
10.             TextArea area = new TextArea("Welcome to javatpoint");
11. // setting location of text area in frame
12.         area.setBounds(10, 30, 300, 300);
13. // adding text area to frame
14.         f.add(area);
15. // setting size, layout and visibility of frame
16.         f.setSize(400, 400);
17.         f.setLayout(null);
18.         f.setVisible(true);
19.     }
20. // main method
21. public static void main(String args[])
22. {
23.    new TextAreaExample();
24. }
25. }
```

Output:



e)Checkbox

The Checkbox 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".

AWT Checkbox Class Declaration

1. **public class** Checkbox **extends** Component **implements** ItemSelectable, Accessible

Checkbox Class Constructors

Sr. no.	Constructor	Description
1.	Checkbox()	It constructs a checkbox with no string as the label.
2.	Checkbox(String label)	It constructs a checkbox with the given label.
3.	Checkbox(String label, boolean state)	It constructs a checkbox with the given label and sets the given state.
4.	Checkbox(String label, boolean state, CheckboxGroup group)	It constructs a checkbox with the given label, set the given state in the specified checkbox group.
5.	Checkbox(String label, CheckboxGroup group, boolean state)	It constructs a checkbox with the given label, in the given checkbox group and set to the specified state.

Method inherited by Checkbox

The methods of Checkbox class are inherited by following classes:

- java.awt.Component
- java.lang.Object

Checkbox Class Methods

Sr. no.	Method name	Description
1.	void addItemListener(ItemListener IL)	It adds the given item listener to get the item events from the checkbox.
2.	CheckboxGroup getCheckboxGroup()	It determines the group of checkbox.
3.	String getLabel()	It fetched the label of checkbox.
4.	boolean getState()	It returns true if the checkbox is on, else returns off.
5.	void removeItemListener(ItemListener l)	It removes the specified item listener so that the item listener doesn't receive item events from the checkbox anymore.
6.	void setCheckboxGroup(CheckboxGroup g)	It sets the checkbox's group to the given checkbox.
7.	void setLabel(String label)	It sets the checkbox's label to the string argument.
8.	void setState(boolean state)	It sets the state of checkbox to the specified state.

Java AWT Checkbox Example

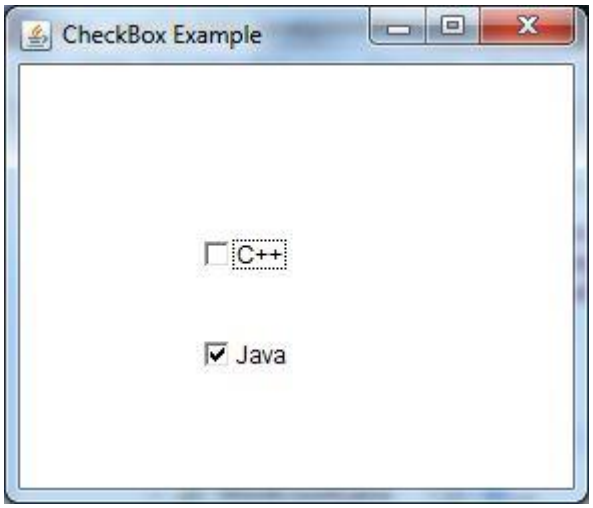
In the following example we are creating two checkboxes using the Checkbox(String label) constructo and adding them into the Frame using add() method.

CheckboxExample1.java

```
1. // importing AWT class
2. import java.awt.*;
3. public class CheckboxExample1
4. {
5. // constructor to initialize
6.     CheckboxExample1() {
7. // creating the frame with the title
8.       Frame f = new Frame("Checkbox Example");
9. // creating the checkboxes
```

```
10.         Checkbox checkbox1 = new Checkbox("C++");
11.         checkbox1.setBounds(100, 100,  50, 50);
12.         Checkbox checkbox2 = new Checkbox("Java", true);
13. // setting location of checkbox in frame
14. checkbox2.setBounds(100, 150,  50, 50);
15. // adding checkboxes to frame
16.         f.add(checkbox1);
17.         f.add(checkbox2);
18.
19. // setting size, layout and visibility of frame
20.         f.setSize(400,400);
21.         f.setLayout(null);
22.         f.setVisible(true);
23.     }
24. // main method
25. public static void main (String args[])
26. {
27.     new CheckboxExample1();
28. }
29. }
```

Output:



f)Checkbox groups

The object of CheckboxGroup class is used to group together a set of Checkbox. At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state. It inherits the object class.

Note: CheckboxGroup enables you to create radio buttons in AWT. There is no special control for creating radio buttons in AWT.

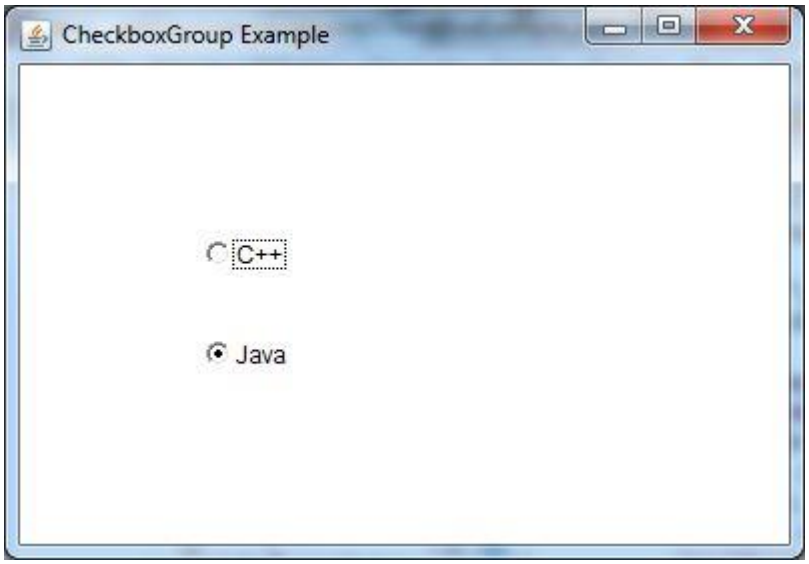
AWT CheckboxGroup Class Declaration

```
1. public class CheckboxGroup extends Object implements Serializable
```

Java AWT CheckboxGroup Example

```
1. import java.awt.*;
2. public class CheckboxGroupExample
3. {
4.         CheckboxGroupExample(){
5.         Frame f= new Frame("CheckboxGroup Example");
6.          CheckboxGroup cbg = new CheckboxGroup();
7.          Checkbox checkBox1 = new Checkbox("C++", cbg, false);
8.          checkBox1.setBounds(100,100, 50,50);
9.          Checkbox checkBox2 = new Checkbox("Java", cbg, true);
10.         checkBox2.setBounds(100,150, 50,50);
11.         f.add(checkBox1);
12.         f.add(checkBox2);
13.         f.setSize(400,400);
14.         f.setLayout(null);
15.         f.setVisible(true);
16.     }
17. public static void main(String args[])
18. {
19.     new CheckboxGroupExample();
20. }
21. }
```

Output:



g)Choice

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 Component class.

AWT Choice Class Declaration

1. **public class** Choice **extends** Component **implements** ItemSelectable, Accessible

Choice Class constructor

Sr. no.	Constructor	Description
1.	Choice()	It constructs a new choice menu.

Methods inherited by class

The methods of Choice class are inherited by following classes:

- java.awt.Component
- java.lang.Object

Choice Class Methods

Sr. no.	Method name	Description
1.	void add(String item)	It adds an item to the choice menu.
2.	void addItemListener(ItemListener l)	It adds the item listener that receives item events from the choice menu.
3.	String getItem(int index)	It gets the item (string) at the given index position in the choice menu.
4.	int getItemCount()	It returns the number of items of the choice menu.
5.	int getSelectedIndex()	Returns the index of the currently selected item.
6.	String getSelectedItem()	Gets a representation of the current choice as a string.

7.	void insert(String item, int index)	Inserts the item into this choice at the specified position.
8.	void remove(int position)	It removes an item from the choice menu at the given index position.
9.	void remove(String item)	It removes the first occurrence of the item from choice menu.
10.	void removeAll()	It removes all the items from the choice menu.
11.	void removeItemListener (ItemListener l)	It removes the mentioned item listener. Thus is doesn't receive item events from the choice menu anymore.
12.	void select(int pos)	It changes / sets the selected item in the choice menu to the item at given index position.
13.	void select(String str)	It changes / sets the selected item in the choice menu to the item whose string value is equal to string specified in the argument.

Java AWT Choice Example

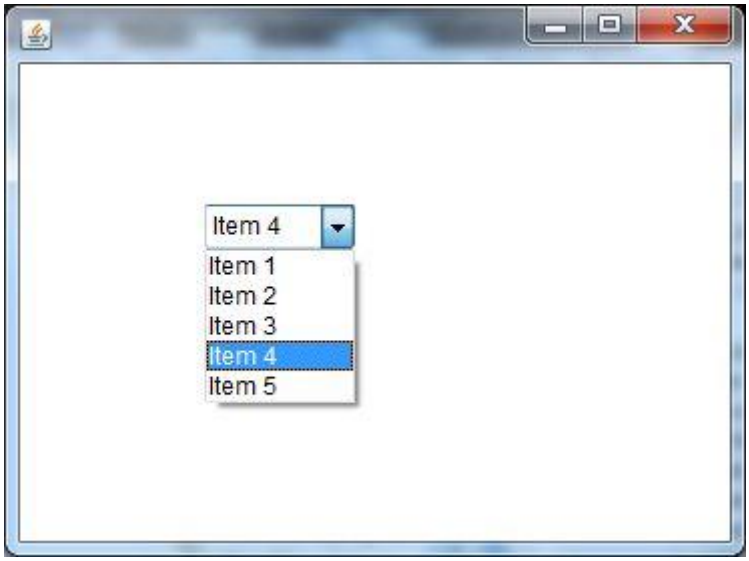
In the following example, we are creating a choice menu using Choice() constructor. Then we add 5 items to the menu using add() method and Then add the choice menu into the Frame.

ChoiceExample1.java

```
1. // importing awt class
2. import java.awt.*;
3. public class ChoiceExample1 {
4.
5.     // class constructor
6.     ChoiceExample1() {
7.
8.     // creating a frame
9.     Frame f = new Frame();
10.
11.    // creating a choice component
12.    Choice c = new Choice();
13.
14.    // setting the bounds of choice menu
15.    c.setBounds(100, 100, 75, 75);
16.
17.    // adding items to the choice menu
```

```
18.         c.add("Item 1");
19.         c.add("Item 2");
20.         c.add("Item 3");
21.         c.add("Item 4");
22.         c.add("Item 5");
23.
24.         // adding choice menu to frame
25.         f.add(c);
26.
27.         // setting size, layout and visibility of frame
28.         f.setSize(400, 400);
29.         f.setLayout(null);
30.         f.setVisible(true);
31.     }
32.
33. // main method
34. public static void main(String args[])
35. {
36.    new ChoiceExample1();
37. }
38. }
```

Output:



h)List

The object of List class represents a list of text items. With the help of the List class, user can choose either one item or multiple items. It inherits the Component class.

AWT List class Declaration

1. **public class** List **extends** Component **implements** ItemSelectable, Accessible

AWT List Class Constructors

Sr. no.	Constructor	Description
1.	List()	It constructs a new scrolling list.
2.	List(int row_num)	It constructs a new scrolling list initialized with the given number of rows visible.
3.	List(int row_num, Boolean multipleMode)	It constructs a new scrolling list initialized which displays the given number of rows.

Methods Inherited by the List Class

The List class methods are inherited by following classes:

- java.awt.Component
- java.lang.Object

List Class Methods

Sr. no.	Method name	Description
1.	void add(String item)	It adds the specified item into the end of scrolling list.
2.	void add(String item, int index)	It adds the specified item into list at the given index position.
3.	void addActionListener(ActionListener l)	It adds the specified action listener to receive action events from list.
4.	void addItemListener(ItemListener l)	It adds specified item listener to receive item events from list.
6.	void deselect(int index)	It deselects the item at given index position.
9.	String getItem(int index)	It fetches the item related to given index position.
10.	int getItemCount()	It gets the count/number of items in the list.
12.	String[] getItems()	It fetched the items from the list.

13.	Dimension getMinimumSize()	It gets the minimum size of a scrolling list.
14.	Dimension getMinimumSize(int rows)	It gets the minimum size of a list with given number of rows.
15.	Dimension getPreferredSize()	It gets the preferred size of list.
16.	Dimension getPreferredSize(int rows)	It gets the preferred size of list with given number of rows.
17.	int getRows()	It fetches the count of visible rows in the list.
18.	int getSelectedIndex()	It fetches the index of selected item of list.
19.	int[] getSelectedIndexes()	It gets the selected indices of the list.
20.	String getSelectedItem()	It gets the selected item on the list.
21.	String[] getSelectedItems()	It gets the selected items on the list.
22.	Object[] getSelectedObjects()	It gets the selected items on scrolling list in array of objects.
23.	int getVisibleIndex()	It gets the index of an item which was made visible by method makeVisible()
24.	void makeVisible(int index)	It makes the item at given index visible.
25.	boolean isIndexSelected(int index)	It returns true if given item in the list is selected.
26.	boolean isMultipleMode()	It returns the true if list allows multiple selections.
31.	void removeActionListener(ActionListener l)	It removes specified action listener. Thus it doesn't receive further action events from the list.
32.	void removeItemListener(ItemListener l)	It removes specified item listener. Thus it doesn't receive further action events from the list.
33.	void remove(int position)	It removes the item at given index position from the list.
34.	void remove(String item)	It removes the first occurrence of an item from list.
35.	void removeAll()	It removes all the items from the list.
36.	void replaceItem(String newVal, int index)	It replaces the item at the given index in list with the new string specified.

37.	void select(int index)	It selects the item at given index in the list.
38.	void setMultipleMode(boolean b)	It sets the flag which determines whether the list will allow multiple selection or not.

Java AWT List Example

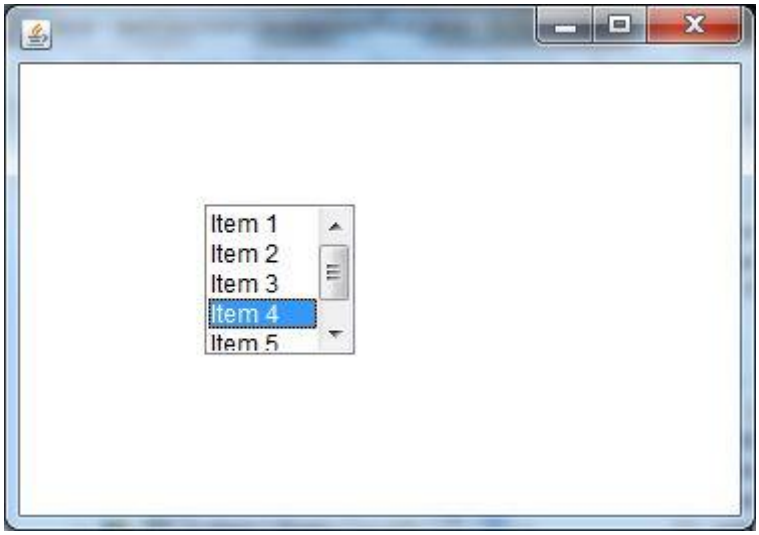
In the following example, we are creating a List component with 5 rows and adding it into the Frame.

ListExample1.java


```
1. // importing awt class
2. import java.awt.*;
3.
4. public class ListExample1
5. {
6.     // class constructor
7.      ListExample1() {
8.      // creating the frame
9.         Frame f = new Frame();
10.       // creating the list of 5 rows
11.        List l1 = new List(5);
12.
13.        // setting the position of list component
14.        l1.setBounds(100, 100, 75, 75);
15.
16.        // adding list items into the list
17.        l1.add("Item 1");
18.        l1.add("Item 2");
19.        l1.add("Item 3");
20.        l1.add("Item 4");
21.        l1.add("Item 5");
22.
23.        // adding the list to frame
24.        f.add(l1);
25.
26.        // setting size, layout and visibility of frame
27.        f.setSize(400, 400);
28.        f.setLayout(null);
29.        f.setVisible(true);
30.     }
31.
```

```
32. // main method
33. public static void main(String args[])
34. {
35.    new ListExample1();
36. }
37. }
```

Output:



i)Panel

The Panel is a simplest container class. It provides space in which an application can attach any other component. It inherits the Container class.

It doesn't have title bar.

AWT Panel class declaration

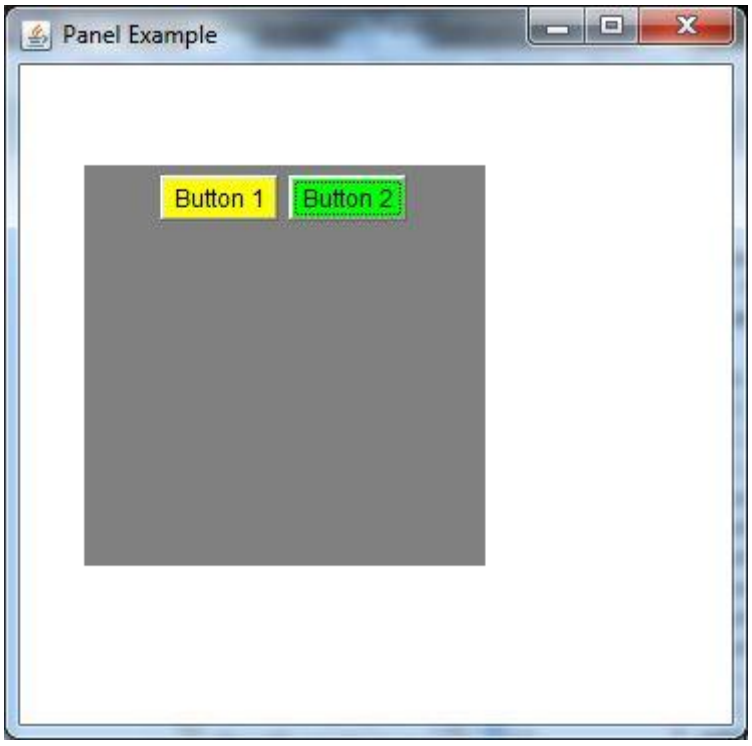
```
1. public class Panel extends Container implements Accessible
```

Java AWT Panel Example

```
1. import java.awt.*;
2. public class PanelExample {
3.     PanelExample()
4.       {
5.       Frame f= new Frame("Panel Example");
6.       Panel panel=new Panel();
7.       panel.setBounds(40,80,200,200);
8.       panel.setBackground(Color.gray);
9.       Button b1=new Button("Button 1");
10.      b1.setBounds(50,100,80,30);
```

```
11.         b1.setBackground(Color.yellow);
12.         Button b2=new Button("Button 2");
13.         b2.setBounds(100,100,80,30);
14.         b2.setBackground(Color.green);
15.         panel.add(b1); panel.add(b2);
16.         f.add(panel);
17.         f.setSize(400,400);
18.         f.setLayout(null);
19.         f.setVisible(true);
20.         }
21.         public static void main(String args[])
22.         {
23.         new PanelExample();
24.         }
25. }
```

Output:



j)Scroll pane

When screen real estate is limited, use a scroll pane to display a component that is large or one whose size can change dynamically.

Fields	
Modifier and Type	**Field and Description**

static int	**SCROLLBARS_ALWAYS** Specifies that horizontal/vertical scrollbars should always be shown regardless of the respective sizes of the scrollpane and child.
static int	**SCROLLBARS_AS_NEEDED** Specifies that horizontal/vertical scrollbar should be shown only when the size of the child exceeds the size of the scrollpane in the horizontal/vertical dimension.
static int	**SCROLLBARS_NEVER** Specifies that horizontal/vertical scrollbars should never be shown regardless of the respective sizes of the scrollpane and child.

Constructors

Constructor and Description
ScrollPane() Create a new scrollpane container with a scrollbar display policy of "as needed".
ScrollPane(int scrollbarDisplayPolicy) Create a new scrollpane container.

Example program:-

```
import java.awt.*;

public class ScrollPaneEx extends Frame{
   ScrollPaneEx()
   {
      setSize(400,400);

      setTitle("ScrollPaneExample");
      setVisible(true);
```

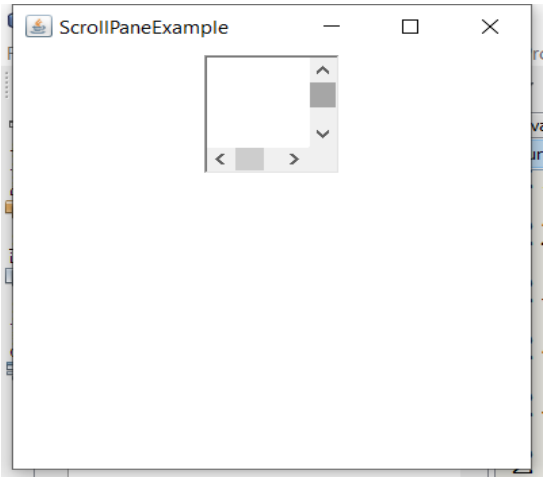
```
        this.setLayout(new FlowLayout());

        ScrollPane p=new ScrollPane(ScrollPane.SCROLLBARS_ALWAYS);

        add(p);

    }
    public static void main(String args[])
    {
        ScrollPaneEx e=new ScrollPaneEx();
    }
}
```

Output:-



k)Dialog

The Dialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Window class.

Unlike Frame, it doesn't have maximize and minimize buttons.

Frame vs Dialog

Frame and Dialog both inherits Window class. Frame has maximize and minimize buttons but Dialog doesn't have.

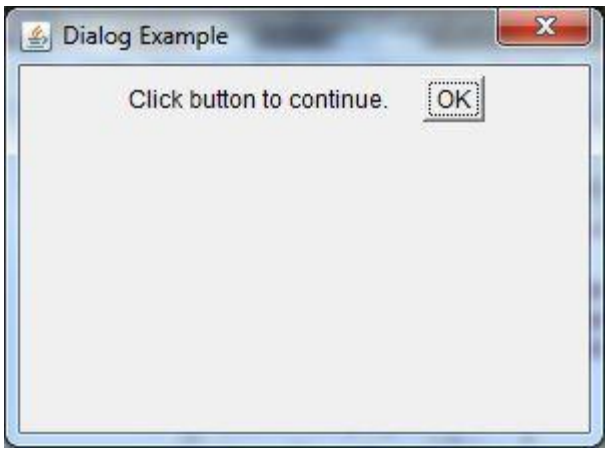
AWT Dialog class declaration

1. **public class** Dialog **extends** Window

Java AWT Dialog Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class DialogExample {
4.     private static Dialog d;
5.     DialogExample() {
6.         Frame f= new Frame();
7.         d = new Dialog(f , "Dialog Example", true);
8.         d.setLayout( new FlowLayout() );
9.         Button b = new Button ("OK");
10.        b.addActionListener ( new ActionListener()
11.        {
12.            public void actionPerformed( ActionEvent e )
13.            {
14.                DialogExample.d.setVisible(false);
15.            }
16.        });
17.        d.add( new Label ("Click button to continue."));
18.        d.add(b);
19.        d.setSize(300,300);
20.        d.setVisible(true);
21.    }
22.    public static void main(String args[])
23.    {
24.        new DialogExample();
25.    }
26. }
```

Output:



L)Menu bar

- A menu bar can be created using **MenuBar** class.
- A menu bar may contain one or multiple menus, and these menus are created using **Menu** class.
- A menu may contain one of multiple menu items and these menu items are created using **MenuItem** class.

Signature: public class MenuBar extends MenuComponent implements MenuContainer, Accessible.

Simple constructors of MenuBar, Menu and MenuItem

Constructor	Description
public MenuBar()	Creates a menu bar to which one or many menus are added.
public Menu(String title)	Creates a menu with a title.
public MenuItem(String title)	Creates a menu item with a title.

Menu bar example program;-

```
1.  import java.awt.*;
2.  class MenuExample
3.  {
4.      MenuExample(){
5.          Frame f= new Frame("Menu and MenuItem Example");
6.          MenuBar mb=new MenuBar();
7.          Menu menu=new Menu("Menu");
8.          Menu submenu=new Menu("Sub Menu");
9.          MenuItem i1=new MenuItem("Item 1");
10.         MenuItem i2=new MenuItem("Item 2");
11.         MenuItem i3=new MenuItem("Item 3");
12.         MenuItem i4=new MenuItem("Item 4");
13.         MenuItem i5=new MenuItem("Item 5");
14.         menu.add(i1);
15.         menu.add(i2);
16.         menu.add(i3);
17.         submenu.add(i4);
18.         submenu.add(i5);
19.         menu.add(submenu);
20.         mb.add(menu);
21.         f.setMenuBar(mb);
22.         f.setSize(400,400);
```

```
23.         f.setLayout(null);
24.         f.setVisible(true);
25. }
26. public static void main(String args[])
27. {
28. new MenuExample();
29. }
30. }
```

Output: