UNIT-3

Applet Basic

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

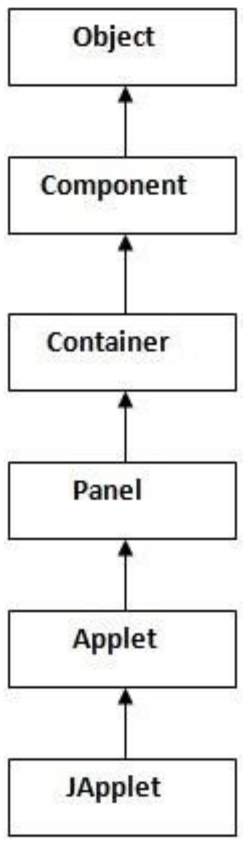
Advantages of Applet

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many platforms, including Linux, Windows, and Mac Os etc.

Disadvantages of Applet

- Plug in is required at client browser to execute applet.

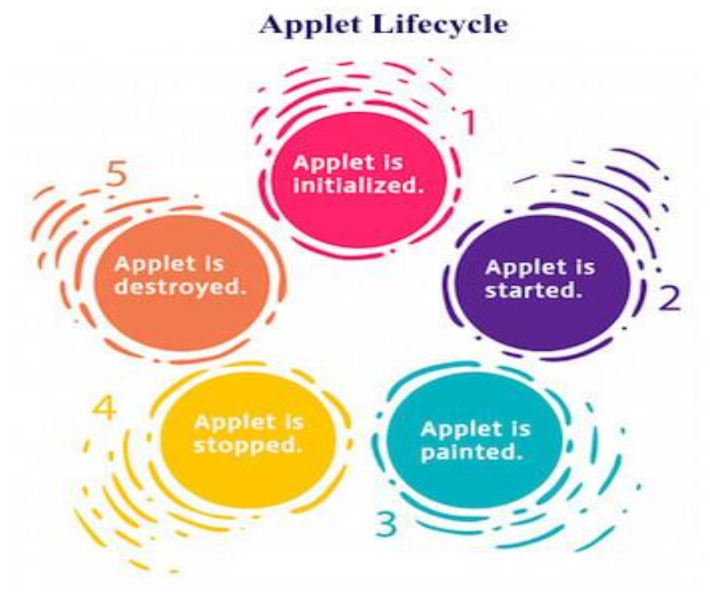
Hierarchy of Applet



As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.



Lifecycle methods for Applet:

java.applet.Applet class

1. public void init(): is used to initialized the Applet. It is invoked only once.
2. public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. public void stop(): is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. public void destroy(): is used to destroy the Applet. It is invoked only once.

java.awt.Component class

1. public void paint(Graphics g): is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Run an Applet

There are two ways to run an applet

1. By html file.
2. By appletviewer tool (for testing purpose).

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
public void paint(Graphics g)
{
g.drawString("welcome",150,150);
}
}
```

Note: class must be public because its object is created by Java Plugin software that resides on the browser.

myapplet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

Displaying image

For displaying image, we can use the method drawImage() of Graphics class.

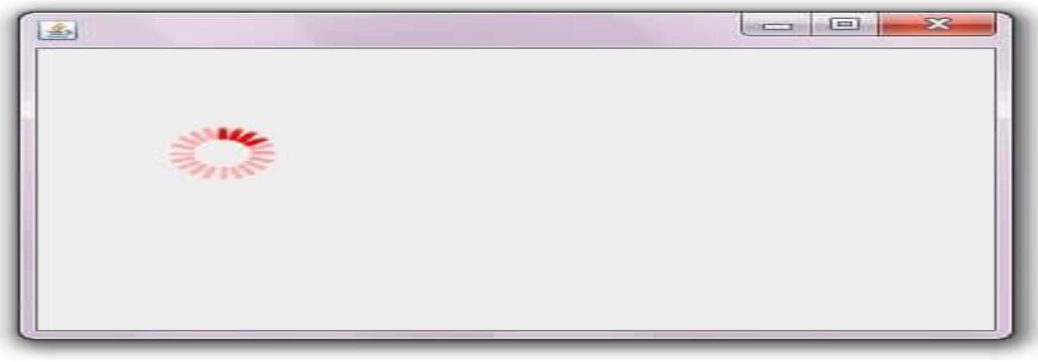
Syntax of drawImage() method:

1. public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.

Example of displaying image in swing:

```
import java.awt.*;
import javax.swing.JFrame;
public class MyCanvas extends Canvas
{
public void paint(Graphics g)
{
```

```
Toolkit t=Toolkit.getDefaultToolkit();
Image i=t.getImage("p3.gif");
g.drawImage(i, 120,100,this);
}
public static void main(String[] args)
{
MyCanvas m=new MyCanvas();
JFrame f=new JFrame();
f.add(m);
f.setSize(400,400);
f.setVisible(true);
}
}
```



Animation

Applet is mostly used in games and animation. For this purpose image is required to be moved.

Example of animation in applet:

```
import java.awt.*;
import java.applet.*;
public class AnimationExample extends Applet
{
Image picture;
public void init()
{
picture =getImage(getDocumentBase(),"bike_1.gif");
}
public void paint(Graphics g)
{
for(int i=0;i<500;i++)
{
g.drawImage(picture, i,30, this);

try{Thread.sleep(100);}catch(Exception e){}
}
}
}
```

In the above example, drawImage() method of Graphics class is used to display the image.

The fourth argument of drawImage() method of is ImageObserver object. The Component class implements Image Observer interface.

So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

myapplet.html

```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

Painting

We can perform painting operation in applet by the mouse Dragged() method of Mouse Motion Listener.

Example of Painting in Applet:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class MouseDrag extends Applet implements MouseMotionListener
{
public void init()
{
addMouseMotionListener(this);
setBackground(Color.red);
}
public void mouseDragged(MouseEvent me)
{
Graphics g=getGraphics();
g.setColor(Color.white);
g.fillOval(me.getX(),me.getY(),5,5);
}
public void mouseMoved(MouseEvent me){}
}
```

In the above example, getX() and getY() method of MouseEvent is used to get the current x-axis and y-axis. The getGraphics() method of Component class returns the object of Graphics.

myapplet.html

```
<html>
<body>
<applet code="MouseDrag.class" width="300" height="300">
</applet>
</body>
</html>
```

<u>Displaying Clock</u>

Digital clock can be created by using the Calendar and SimpleDateFormat class. Let's see the simple example:

Example of Digital clock in Applet:

```
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.text.*;
public class DigitalClock extends Applet implements Runnable
{
Thread t = null;
int hours=0, minutes=0, seconds=0;
String timeString = "";
public void init()
{
setBackground( Color.green);
}
public void start()
{
t = new Thread( this );
t.start();
}

public void run()
{
try
{
while (true)
{
```

```
Calendar cal = Calendar.getInstance();
hours = cal.get( Calendar.HOUR_OF_DAY );
if ( hours > 12 ) hours -= 12;
minutes = cal.get( Calendar.MINUTE );
seconds = cal.get( Calendar.SECOND );
SimpleDateFormat formatter = new SimpleDateFormat("hh:mm:ss");
Date date = cal.getTime();
timeString = formatter.format( date );
repaint();
t.sleep( 1000 );
}
}
catch (Exception e) { }
}
public void paint( Graphics g )
{
g.setColor( Color.blue );
g.drawString( timeString, 50, 50 );
}
}
```

In the above example, getX() and getY() method of MouseEvent is used to get the current x-axis and y-axis. The getGraphics() method of Component class returns the object of Graphics.

myapplet.html

```
<html>
<body>
<applet code="DigitalClock.class" width="300" height="300">
</applet>
</body>
</html>
```

Event Handling

As we perform event handling in AWT or Swing, we can perform it in applet also. Let's see the simple example of event handling in applet that prints a message by click on the button.

Example of EventHandling in applet:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
```

```
public class EventApplet extends Applet implements ActionListener
{
Button b;
TextField tf;
public void init()
{
tf=new TextField();
tf.setBounds(30,40,150,20);
b=new Button("Click");
b.setBounds(80,150,60,50);
add(b);add(tf);
b.addActionListener(this);
setLayout(null);
}
 public void actionPerformed(ActionEvent e)
{
  tf.setText("Welcome");
 }
 }
```

myapplet.html

```
<html>
<body>
<applet code="EventApplet.class" width="300" height="300">
</applet>
</body>
</html>
```

JApplet class

As we prefer Swing to AWT. Now we can use JApplet that can have all the controls of swing. The JApplet class extends the Applet class.

Example of EventHandling in JApplet:

```
import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
public class EventJApplet extends JApplet implements ActionListener
{
JButton b;
JTextField tf;
public void init()
{
```

```
tf=new JTextField();
tf.setBounds(30,40,150,20);
b=new JButton("Click");
b.setBounds(80,150,70,40);
add(b);add(tf);
b.addActionListener(this);
setLayout(null);
}
public void actionPerformed(ActionEvent e)
{
tf.setText("Welcome");
}
}
```

In the above example, we have created all the controls in init() method because it is invoked only once.

myapplet.html

```
<html>
<body>
<applet code="EventJApplet.class" width="300" height="300">
</applet>
</body>
</html>
```

AWT Basic

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.

The AWT tutorial will help the user to understand Java GUI programming in simple and easy steps.

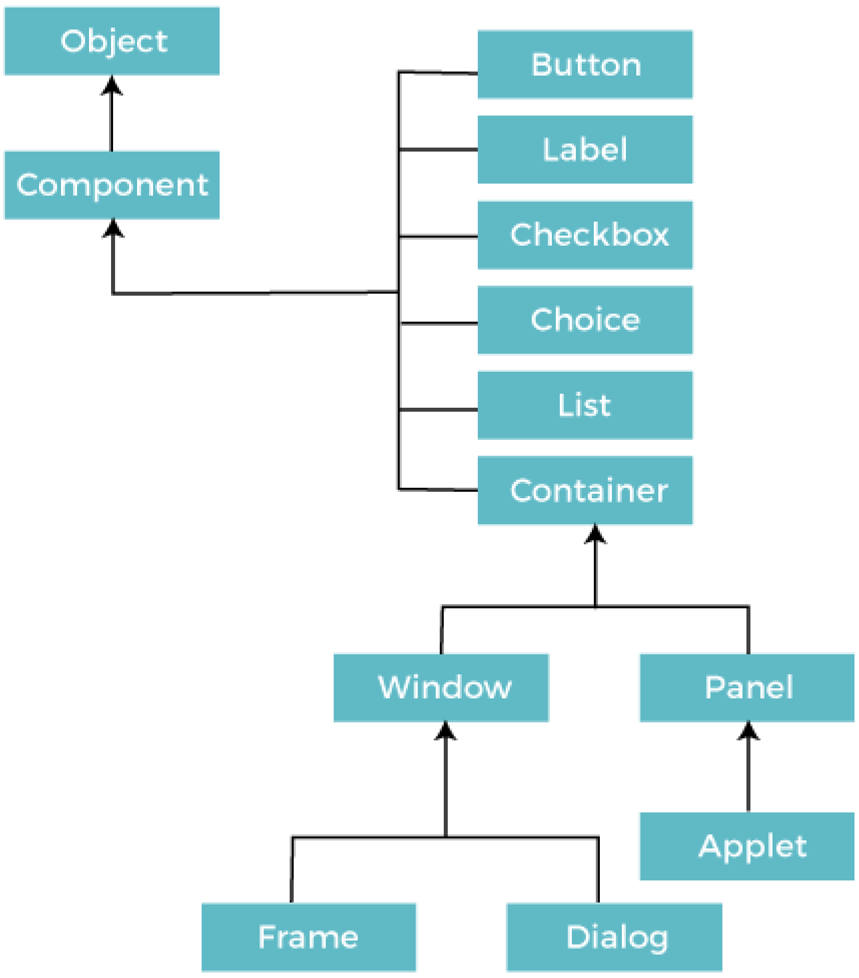
Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, ChechBox, button, etc.

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.

Java AWT Hierarchy

The hierarchy of Java AWT classes is 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 other components like buttons, textfields, labels etc. The classes that extend 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 three types of containers in Java AWT:

1. Window
2. Panel
3. Frame

Window

The window is the container that has 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

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

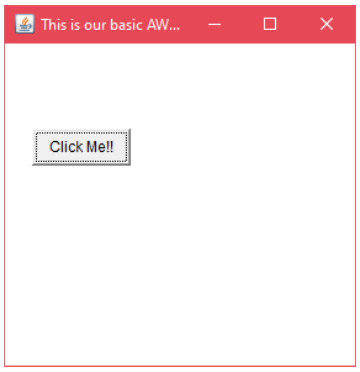
Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

AWTExample1.java

```
import java.awt.*;
public class AWTExample1 extends Frame
{
AWTExample1()
{
Button b = new Button("Click Me!!");
b.setBounds(30,100,80,30);
add(b);
setSize(300,300);
setTitle("This is our basic AWT example");
setLayout(null);
setVisible(true);
}
public static void main(String args[])
{
AWTExample1 f = new AWTExample1();
}
}
```

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:



Event Handling

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- Button
  - public void addActionListener(ActionListener a){}
- MenuItem
  - public void addActionListener(ActionListener a){}
- TextField
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- TextArea
  - public void addTextListener(TextListener a){}
- Checkbox
  - public void addItemListener(ItemListener a){}
- Choice
  - public void addItemListener(ItemListener a){}
- List
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

Java Event Handling Code

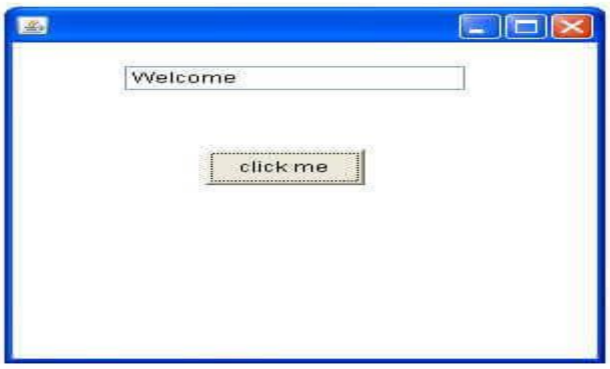
We can put the event handling code into one of the following places:

1. Within class
2. Other class
3. Anonymous class

Java event handling by implementing ActionListener

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener
{
TextField tf;
AEvent()
{
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);
b.addActionListener(this);
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e)
{
tf.setText("Welcome");
}
public static void main(String args[])
{
new AEvent();
}
}
```

public void setBounds(int xaxis, int yaxis, int width, int height); have been used in the above example that sets the position of the component it may be button, textfield etc.



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. The Java application can use the button's action command as a messaging protocol.

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.
5.	void addNotify()	It creates the peer of the button.
6.	AccessibleContext getAccessibleContext()	It fetched the accessible context associated with 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.	ActionListener[ ] getActionListeners()	It returns an array of all the action listeners registered on the button.
10.	T[ ] getListeners(Class listenerType)	It returns an array of all the objects currently registered as FooListeners upon this Button.

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

Java AWT Button Example

Example 1:

ButtonExample.java

```
import java.awt.*;
public class ButtonExample
{
public static void main (String[] args)
{

Frame f = new Frame("Button Example");
Button b = new Button("Click Here");
b.setBounds(50,100,80,30);
f.add(b);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
}
```

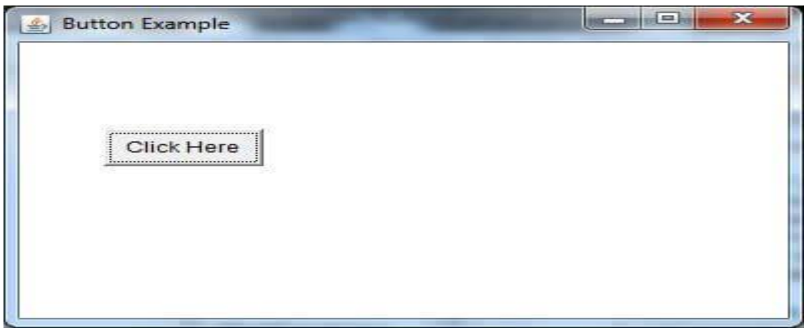
To compile the program using command prompt type the following commands

```
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:



Label

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.
5.	void addNotify()	It creates the peer for the label.

Method inherited

The above methods are inherited by the following classes:

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

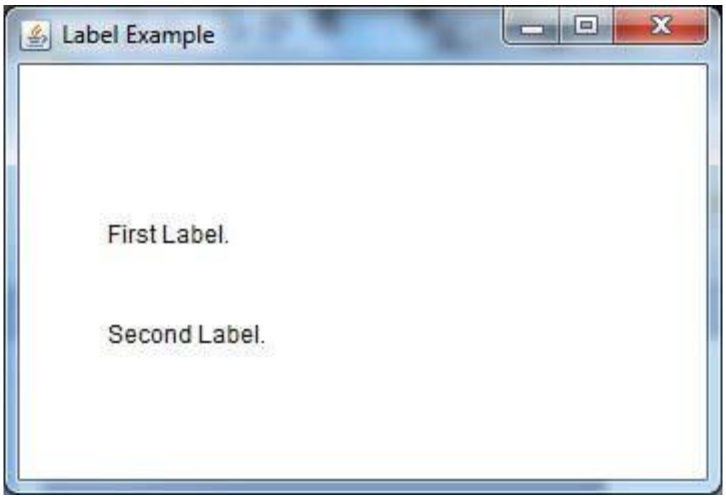
Java AWT Label Example

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

```
import java.awt.*;
 public class LabelExample
{
public static void main(String args[])
{
Frame f = new Frame ("Label example");
Label l1, l2;
l1 = new Label ("First Label.");
l2 = new Label ("Second Label.");
l1.setBounds(50, 100, 100, 30);
l2.setBounds(50, 150, 100, 30);
f.add(l1);
f.add(l2);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
}
```

Output:



```
First Label.

Second Label.
```

<u>Text Field</u>

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.	void addNotify()	It creates the peer of text field.
2.	boolean echoCharIsSet()	It tells whether text field has character set for echoing or not.
3.	void addActionListener(ActionListener l)	It adds the specified action listener to receive action events from the text field.
4.	ActionListener[] getActionListeners()	It returns array of all action listeners registered on text field.
5.	AccessibleContext getAccessibleContext()	It fetches the accessible context related to the text field.
6.	int getColumns()	It fetches the number of columns in text field.
7.	char getEchoChar()	It fetches the character that is used for echoing.
8.	Dimension getMinimumSize()	It fetches the minimum dimensions for the text field.
9.	Dimension getMinimumSize(int columns)	It fetches the minimum dimensions for the text field with specified number of columns.
10.	Dimension getPreferredSize()	It fetches the preferred size of the text field.

Method Inherited

The AWT TextField class inherits the methods from below classes:

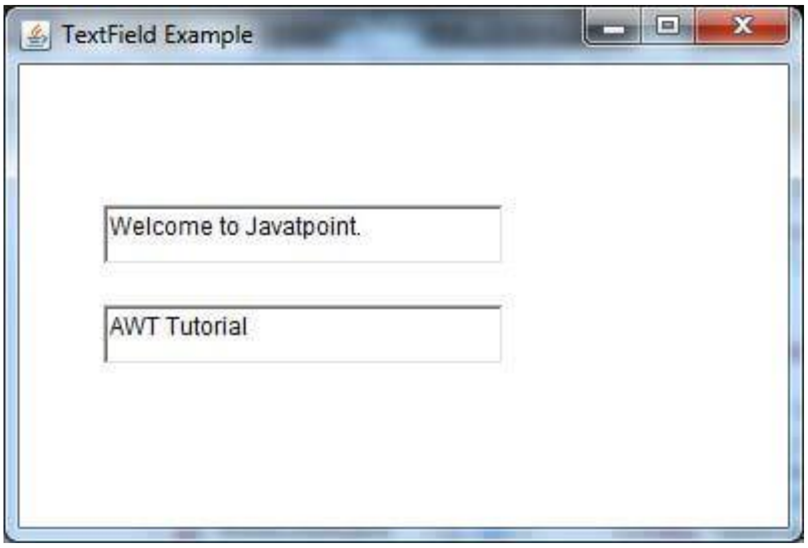
1. java.awt.TextComponent
2. java.awt.Component
3. java.lang.Object

Java AWT TextField Example

TextFieldExample1.java

```
import java.awt.*;
public class TextFieldExample1
{
public static void main(String args[])
{
Frame f = new Frame("TextField Example");
TextField t1, t2;
t1 = new TextField("Welcome to Javatpoint.");
t1.setBounds(50, 100, 200, 30);
t2 = new TextField("AWT Tutorial");
t2.setBounds(50, 150, 200, 30);
f.add(t1);
f.add(t2);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
}
```

Output:



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, Acc
   essible
```

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.	AccessibleContext getAccessibleContext()	It fetches the accessible context of checkbox.
3.	void addNotify()	It creates the peer of checkbox.
4.	CheckboxGroup getCheckboxGroup()	It determines the group of checkbox.
5.	ItemListener[] getItemListeners()	It returns an array of the item listeners registered on checkbox.
6.	String getLabel()	It fetched the label of checkbox.
7.	T[] getListeners(Class listenerType)	It returns an array of all the objects registered as FooListeners.
8.	Object[] getSelectedObjects()	It returns an array (size 1) containing checkbox label and returns null if checkbox is not selected.
9.	boolean getState()	It returns true if the checkbox is on, else returns off.
10.	protected String paramString()	It returns a string representing the state of checkbox.

Java AWT Checkbox Example

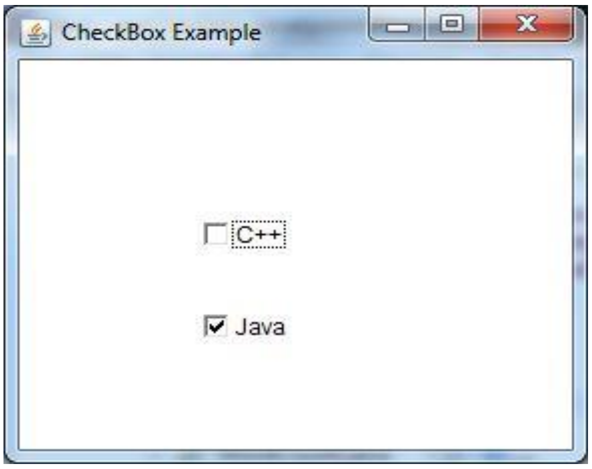
In the following example we are creating two checkboxes using the Checkbox(String label) constructo and adding them into the Frame using add() method.

CheckboxExample1.java

```
import java.awt.*;
public class CheckboxExample1
{
CheckboxExample1()
{
Frame f = new Frame("Checkbox Example");
Checkbox checkbox1 = new Checkbox("C++");
checkbox1.setBounds(100, 100,  50, 50);
Checkbox checkbox2 = new Checkbox("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 CheckboxExample1();
}
}
```

Output:



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

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.	void addNotify()	It creates the peer of choice.
4.	AccessibleContext getAccessibleContext()	It gets the accessbile context related to the choice.
5.	String getItem(int index)	It gets the item (string) at the given index position in the choice menu.
6.	int getItemCount()	It returns the number of items of the choice menu.
7.	ItemListener[] getItemListeners()	It returns an array of all item listeners registered on choice.
8.	T[] getListeners(Class listenerType)	Returns an array of all the objects currently registered as FooListeners upon this Choice.
9.	int getSelectedIndex()	Returns the index of the currently selected item.
10.	String getSelectedItem()	Gets a representation of the current choice as a string.

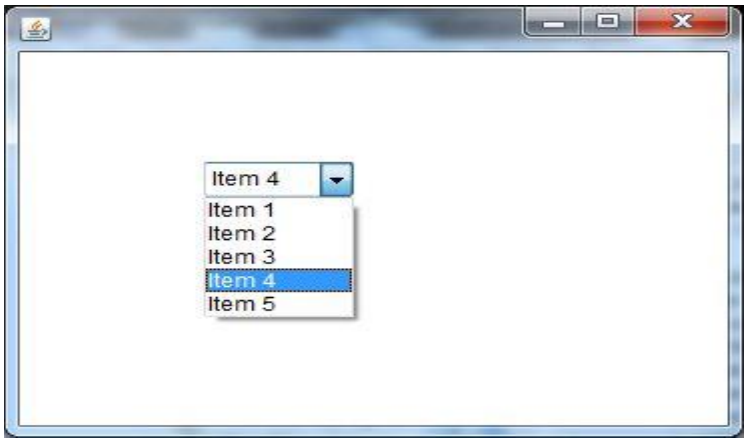
Java AWT Choice Example

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

```
import java.awt.*;
public class ChoiceExample1
{
ChoiceExample1()
{
Frame f = new Frame();
Choice c = new Choice();
c.setBounds(100, 100, 75, 75);
c.add("Item 1");
c.add("Item 2");
c.add("Item 3");
c.add("Item 4");
c.add("Item 5");
f.add(c);
f.setSize(400, 400);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[])
{
new ChoiceExample1();
}
}
```

Output:



Mouse Listener

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

Methods of MouseListener interface

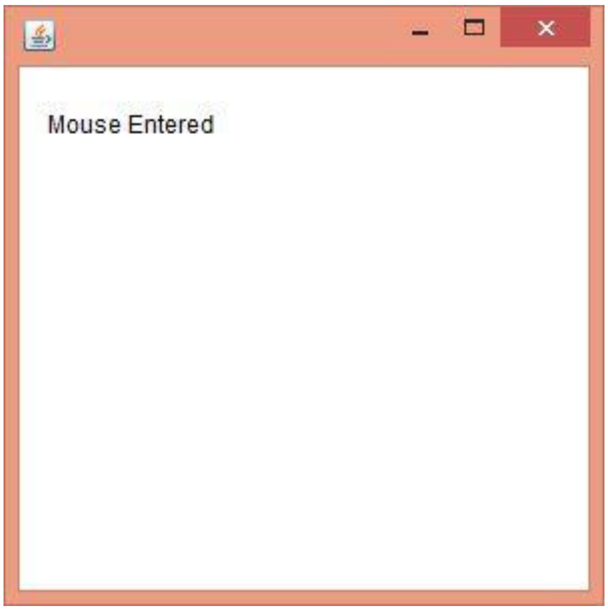
The signature of 5 methods found in MouseListener interface is given below:

1. public abstract void mouseClicked(MouseEvent e);
2. public abstract void mouseEntered(MouseEvent e);
3. public abstract void mouseExited(MouseEvent e);
4. public abstract void mousePressed(MouseEvent e);
5. public abstract void mouseReleased(MouseEvent e);

Java MouseListener Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener
{
Label l;
MouseListenerExample()
{
addMouseListener(this);
l=new Label();
l.setBounds(20,50,100,20);
add(l);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void mouseClicked(MouseEvent e)
{
l.setText("Mouse Clicked");
}
public void mouseEntered(MouseEvent e)
{
l.setText("Mouse Entered");
}
public void mouseExited(MouseEvent e)
{
l.setText("Mouse Exited");
}
public void mousePressed(MouseEvent e)
{
l.setText("Mouse Pressed");
}
public void mouseReleased(MouseEvent e)
{
l.setText("Mouse Released");
}
public static void main(String[] args)
{
new MouseListenerExample();
}
}
```

Output:



```
Mouse Entered
```

Key Listener

The Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package, and it has three methods.

Interface declaration

Following is the declaration for java.awt.event.KeyListener interface:

1. public interface KeyListener extends EventListener

Methods of KeyListener interface

The signature of 3 methods found in KeyListener interface is given below:

Sr. no.	Method name	Description
1.	public abstract void keyPressed (KeyEvent e);	It is invoked when a key has been pressed.
2.	public abstract void keyReleased (KeyEvent e);	It is invoked when a key has been released.
3.	public abstract void keyTyped (KeyEvent e);	It is invoked when a key has been typed.

Methods inherited

This interface inherits methods from the following interface:

- java.awt.EventListener

Java KeyListener Example

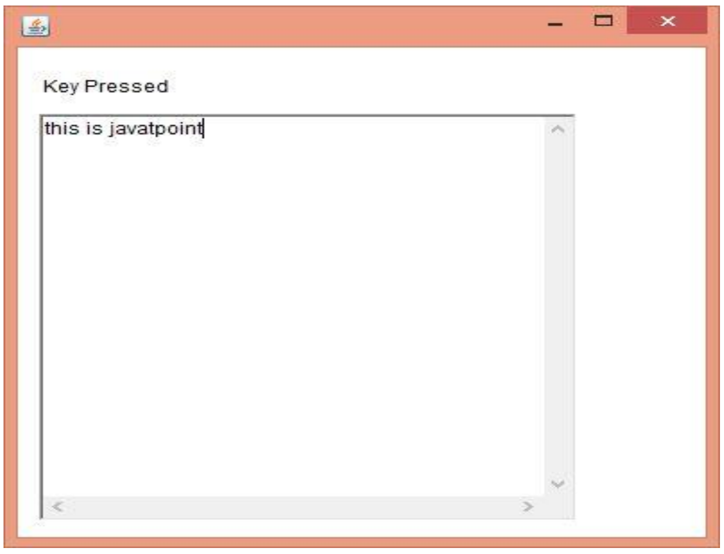
In the following example, we are implementing the methods of the KeyListener interface.

KeyListenerExample.java

```
import java.awt.*;
import java.awt.event.*;
public class KeyListenerExample extends Frame implements KeyListener
{
Label l;
TextArea area;
KeyListenerExample()
{
l = new Label();
l.setBounds (20, 50, 100, 20);
area = new TextArea();
area.setBounds (20, 80, 300, 300);
area.addKeyListener(this);
add(l);
add(area);
setSize (400, 400);
setLayout (null);
setVisible (true);
}
public void keyPressed (KeyEvent e)
{
l.setText ("Key Pressed");
}
public void keyReleased (KeyEvent e)
{
l.setText ("Key Released");
}
public void keyTyped (KeyEvent e)
{
l.setText ("Key Typed");
}
public static void main(String[] args)
{

new KeyListenerExample();
}
}
```

Output:



```
Key Pressed
this is javatpoint
```

Swing

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

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

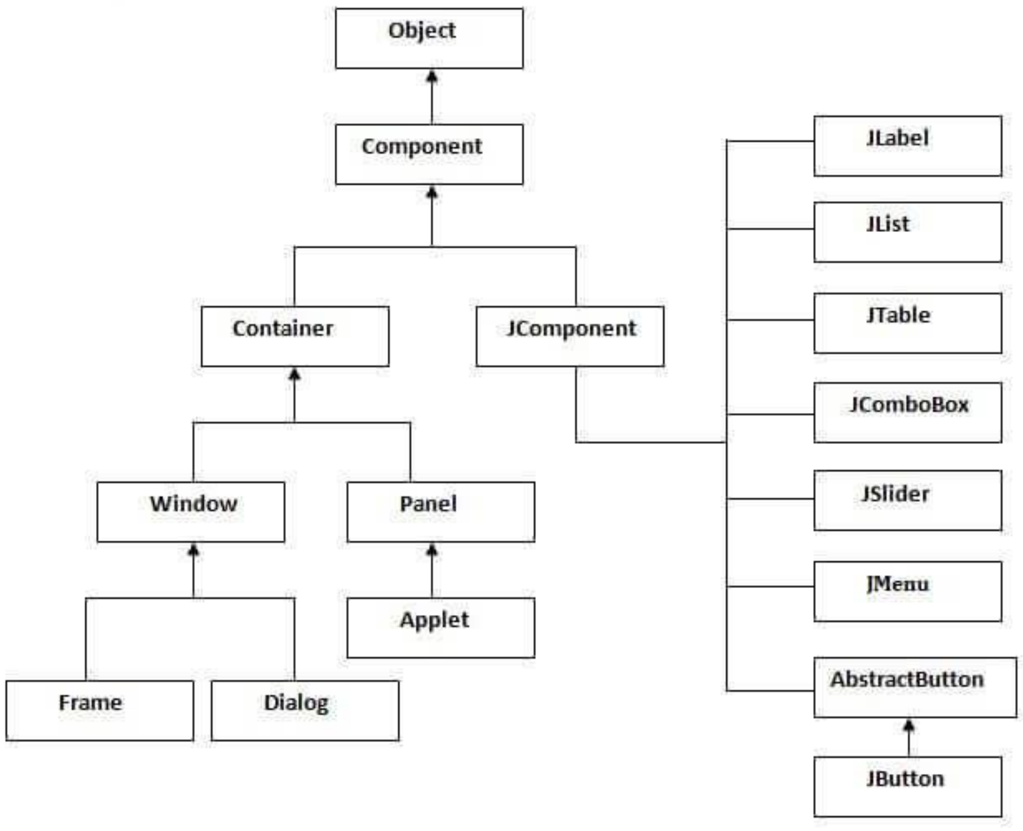
No.	Java AWT	Java Swing
1)	AWT components are platform-dependent.	Java swing components are platform-independent.
2)	AWT components are heavyweight.	Swing components are lightweight.
3)	AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
4)	AWT provides less components 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.

Java Swing Examples

There are two ways to create a frame:

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

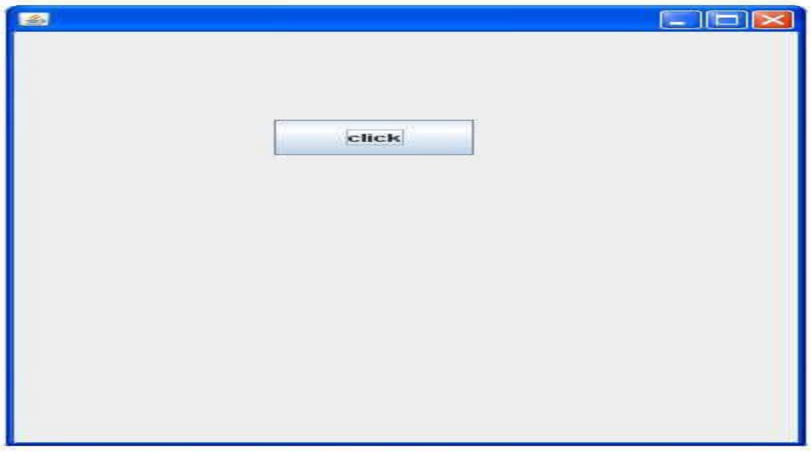
We can write the code of swing inside the main(), constructor or any other method.

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.

File: FirstSwingExample.java

```
import javax.swing.*;
public class FirstSwingExample
{
public static void main(String[] args)
 {
JFrame f=new JFrame();//creating instance of JFrame
JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);//x axis, y axis, width, height
f.add(b);//adding button in JFrame
f.setSize(400,500);
f.setLayout(null);
f.setVisible(true);
}
}
```



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.

1. public class JButton extends AbstractButton implements Accessible

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

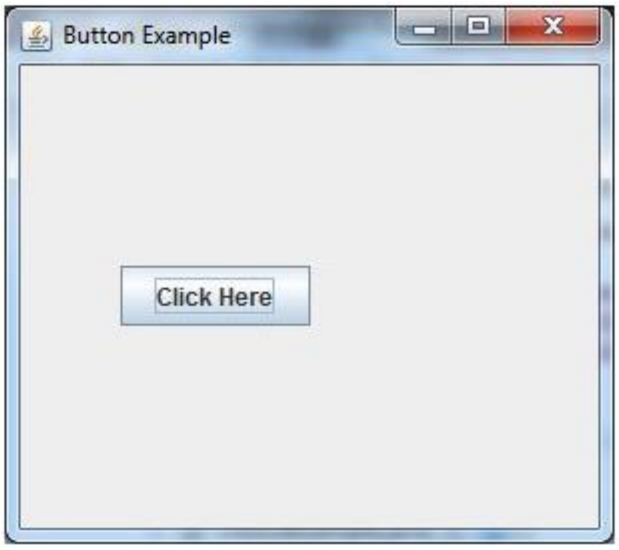
Commonly used Methods of AbstractButton class:

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

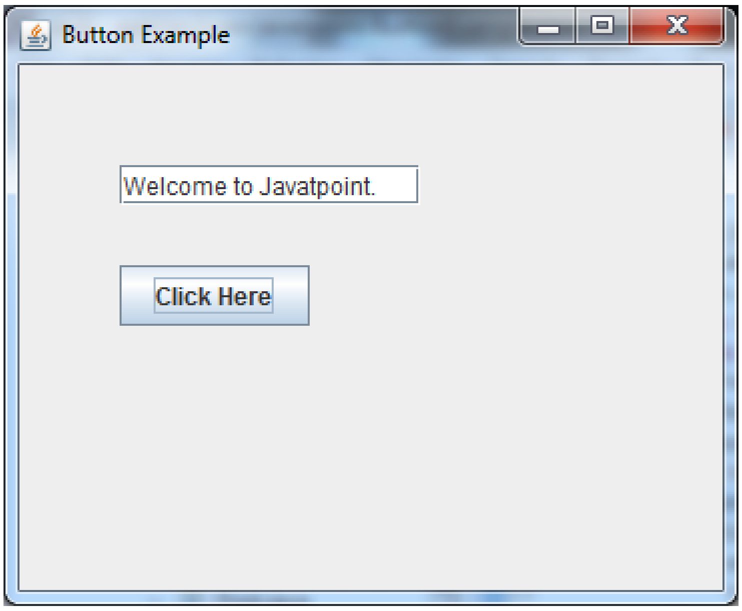
Output:



Java JButton Example with ActionListener

```
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample
{
public static void main(String[] args)
{
JFrame f=new JFrame("Button Example");
final JTextField tf=new JTextField();
tf.setBounds(50,50, 150,20);
JButton b=new JButton("Click Here");
b.setBounds(50,100,95,30);
b.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{
tf.setText("Welcome to Javatpoint.");
}
}
f.add(b);f.add(tf);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
}
```

Output:



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.

1. public class JLabel extends JComponent implements SwingConstants, Accessible

Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.

JLabel(Icon i)	Creates a JLabel instance with the specified image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

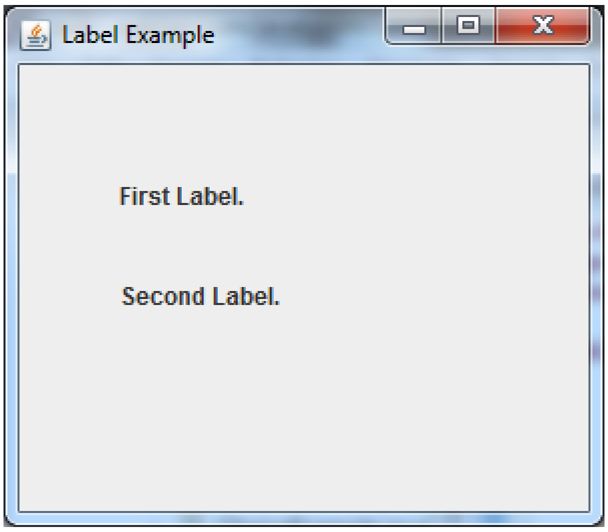
Commonly used Methods:

Methods	Description
String getText()	t returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

Java JLabel Example

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

Output:



```
First Label.

Second Label.
```

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.

1. public class JTextField extends JTextComponent implements SwingConsta nts

Commonly used Constructors:

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.
JTextField(String text, int columns)	Creates a new TextField initialized with the specified text and columns.
JTextField(int columns)	Creates a new empty TextField with the specified number of columns.

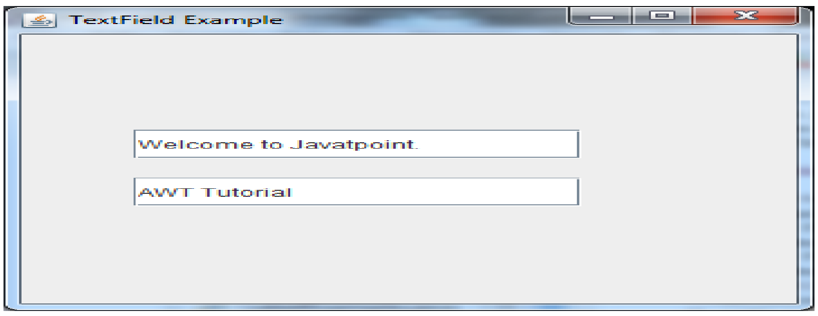
Commonly used Methods:

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

Java JTextField Example

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

Output:



JPasswordField

The object of a JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text. It inherits JTextField class.

JPasswordField class declaration

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

1. public class JPasswordField extends JTextField

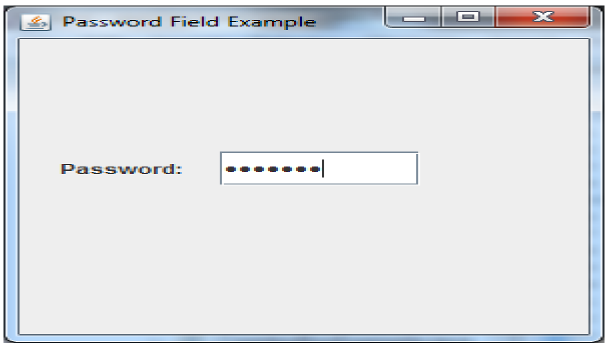
Commonly used Constructors:

Constructor	Description
JPasswordField()	Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
JPasswordField(int columns)	Constructs a new empty JPasswordField with the specified number of columns.
JPasswordField(String text)	Constructs a new JPasswordField initialized with the specified text.
JPasswordField(String text, int columns)	Construct a new JPasswordField initialized with the specified text and columns.

Java JPasswordField Example

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

Output:



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.

1. public class JCheckBox extends JToggleButton implements Accessible

Commonly used Constructors:

Constructor	Description
JJCheckBox()	Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied.

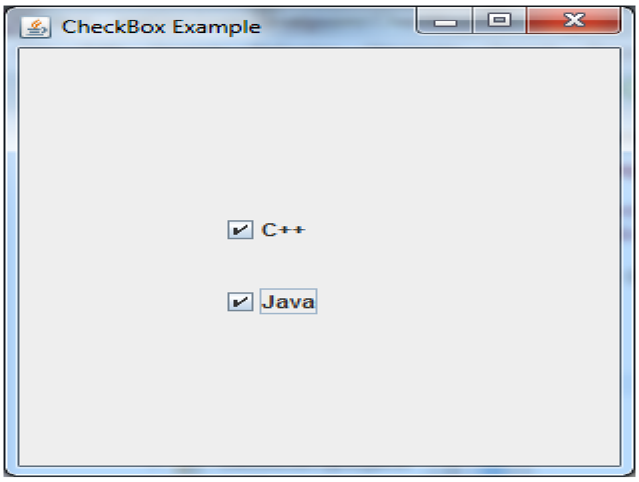
Commonly used Methods:

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.

Java JCheckBox Example

```
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();
}}
```

Output:



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.

1. public class JComboBox extends JComponent implements ItemSelectable, ListDataListener, ActionListener, Accessible

Commonly used Constructors:

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array.
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified Vector.

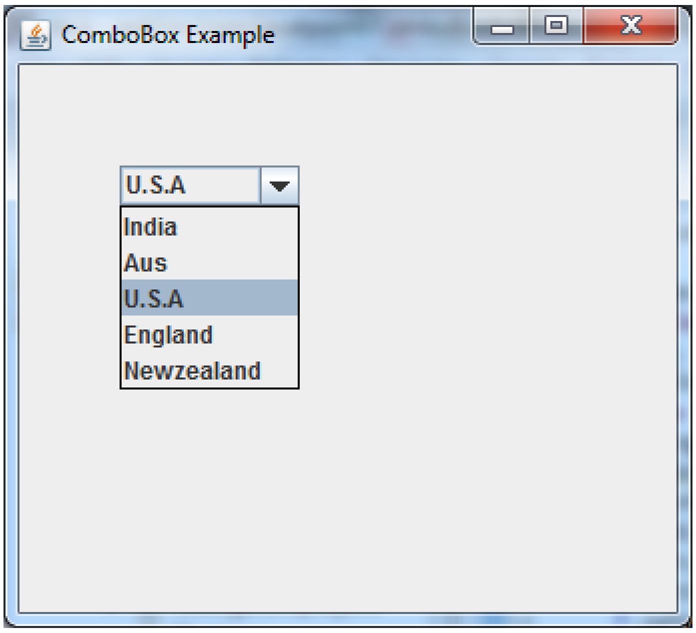
Commonly used Methods:

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

Java JComboBox Example

```
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();
}
}
```

Output:



JProgressBar

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

JProgressBar class declaration

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

1. public class JProgressBar extends JComponent implements SwingConstants , Accessible

Commonly used Constructors:

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

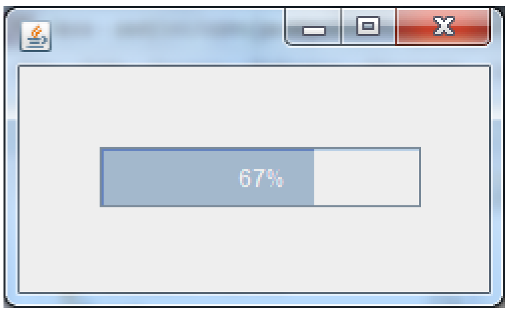
Commonly used Methods:

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

Java JProgressBar Example

```
import javax.swing.*;
public class ProgressBarExample extends JFrame
{
JProgressBar jb;
int i=0,num=0;
ProgressBarExample()
{
jb=new JProgressBar(0,2000);
jb.setBounds(40,40,160,30);
jb.setValue(0);
jb.setStringPainted(true);
add(jb);
setSize(250,150);
setLayout(null);
}
public void iterate()
{
while(i<=2000)
{
jb.setValue(i);
i=i+20;
try{Thread.sleep(150);}catch(Exception e){}
}
}
public static void main(String[] args) {
ProgressBarExample m=new ProgressBarExample();
m.setVisible(true);
m.iterate();
}
}
```

Output:



JSlider

The Java JSlider class is used to create the slider. By using JSlider, a user can select a value from a specific range.

Commonly used Constructors of JSlider class

Constructor	Description
JSlider()	creates a slider with the initial value of 50 and range of 0 to 100.
JSlider(int orientation)	creates a slider with the specified orientation set by either JSlider.HORIZONTAL or JSlider.VERTICAL with the range 0 to 100 and initial value 50.
JSlider(int min, int max)	creates a horizontal slider using the given min and max.
JSlider(int min, int max, int value)	creates a horizontal slider using the given min, max and value.
JSlider(int orientation, int min, int max, int value)	creates a slider using the given orientation, min, max and value.

Commonly used Methods of JSlider class

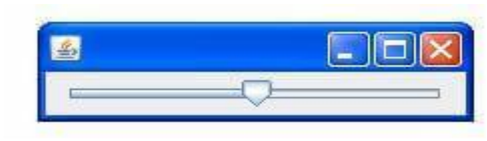
Method	Description
public void setMinorTickSpacing(int n)	is used to set the minor tick spacing to the slider.
public void setMajorTickSpacing(int n)	is used to set the major tick spacing to the slider.
public void setPaintTicks(boolean b)	is used to determine whether tick marks are painted.
public void setPaintLabels(boolean b)	is used to determine whether labels are painted.
public void setPaintTracks(boolean b)	is used to determine whether track is painted.

Java JSlider Example

```
import javax.swing.*;
public class SliderExample1 extends JFrame
{

public SliderExample1() {
JSlider slider = new JSlider(JSlider.HORIZONTAL, 0, 50, 25);
JPanel panel=new JPanel();
panel.add(slider);
add(panel);
}
public static void main(String s[])
{
SliderExample1 frame=new SliderExample1();
frame.pack();
frame.setVisible(true);
}
}
```

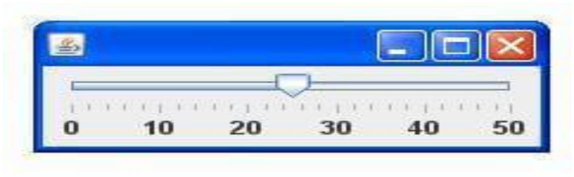
Output:



Java JSlider Example: painting ticks

```
import javax.swing.*;
public class SliderExample extends JFrame
{
public SliderExample()
{
JSlider slider = new JSlider(JSlider.HORIZONTAL, 0, 50, 25);
slider.setMinorTickSpacing(2);
slider.setMajorTickSpacing(10);
slider.setPaintTicks(true);
slider.setPaintLabels(true);
JPanel panel=new JPanel();
panel.add(slider);
add(panel);
}
public static void main(String s[])
{
SliderExample frame=new SliderExample();
frame.pack();
frame.setVisible(true);
}
}
```

Output:


0
10
20
30
40
50