

# Java AWT

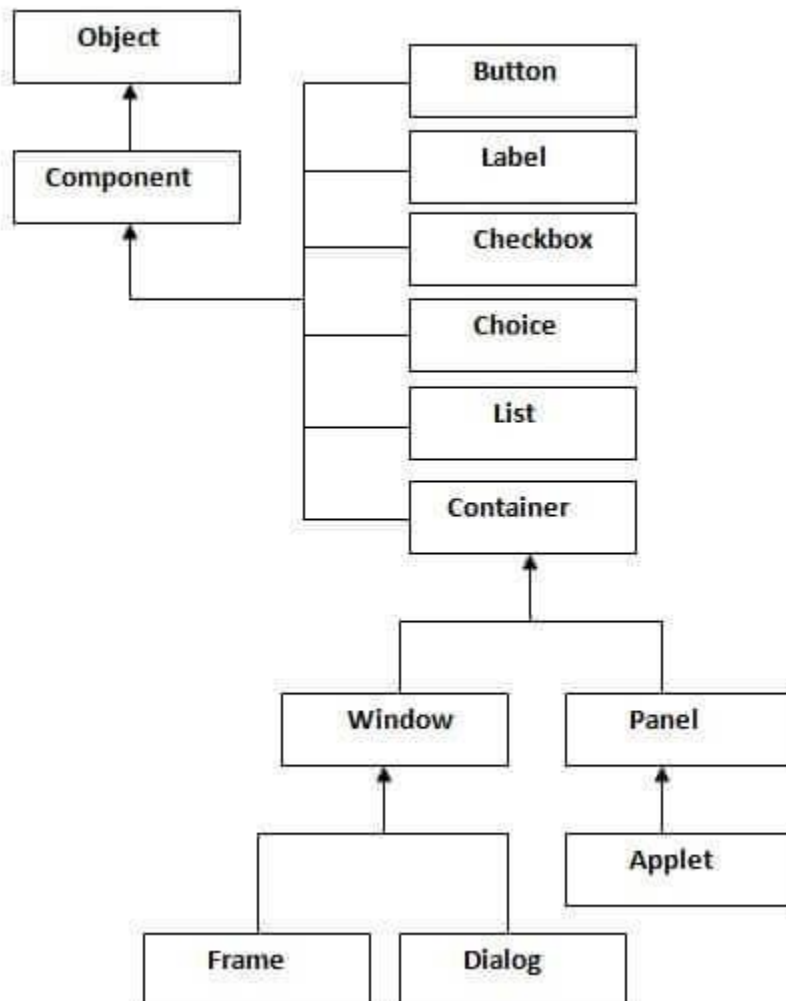
**Java AWT** (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.

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

The java.awt package provides classes for AWT api such as [TextField](#), [Label](#), [TextArea](#), [RadioButton](#), [CheckBox](#), [Choice](#), [List](#) etc.

## Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



## 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.

---

## 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.

---

## Panel

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

---

## Frame

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

---

## Useful Methods of Component class

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

# Java AWT Example

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

- By extending Frame class (inheritance)
  - 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.

```
1. import java.awt.*;
2. class First extends Frame{
3. First(){
4. Button b=new Button("click me");
5. b.setBounds(30,100,80,30);// setting button position
6. add(b);//adding button into frame
7. setSize(300,300);//frame size 300 width and 300 height
8. setLayout(null);//no layout manager
9. setVisible(true);//now frame will be visible, by default not visible
10. }
11. public static void main(String args[]){
12. First f=new First();
13. }}
```

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

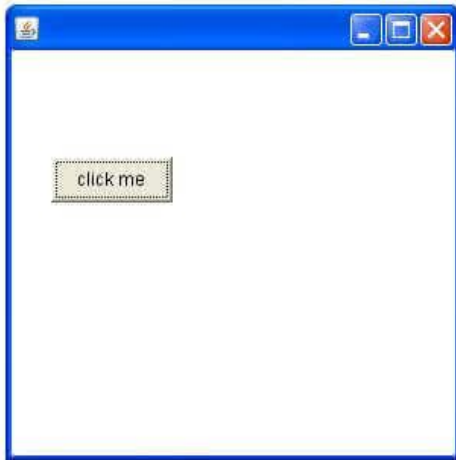


---

## AWT Example by Association

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

```
1. import java.awt.*;
2. class First2{
3. First2(){
4. Frame f=new Frame();
5. Button b=new Button("click me");
6. b.setBounds(30,50,80,30);
7. f.add(b);
8. f.setSize(300,300);
9. f.setLayout(null);
10. f.setVisible(true);
11. }
12. public static void main(String args[]){
13. First2 f=new First2();
14. }}
```



## Basic Terminologies

Term	Description
Component	Component is an object having a graphical representation that can be displayed on the screen and that can interact with the user. For examples buttons, checkboxes, list and scrollbars of a graphical user interface.
Container	Container object is a component that can contain other components. Components added to a container are tracked in a list. The order of the list will define the components' front-to-back stacking order within the container. If no index is specified when adding a component to a container, it will be added to the end of the list.
Panel	Panel provides space in which an application can attach any other components, including other panels.
Window	Window is a rectangular area which is displayed on the screen. In different window we can execute different program and display different data. Window provide us with multitasking environment. A window must have either a frame, dialog, or another window defined as its owner when it's constructed.
Frame	A Frame is a top-level window with a title and a border. The size of the frame includes any area designated for the border. Frame encapsulates <b>window</b> . It and has a title bar, menu bar, borders, and resizing corners.

Canvas	Canvas component represents a blank rectangular area of the screen onto which the application can draw. Application can also trap input events from the use from that blank area of Canvas component.
--------	---

## Examples of GUI based Applications

Following are some of the examples for GUI based applications.

- Automated Teller Machine (ATM)
- Airline Ticketing System
- Information Kiosks at railway stations
- Mobile Applications
- Navigation Systems

## AWT Controls

Every user interface considers the following three main aspects:

- **UI elements** : These are the core visual elements the user eventually sees and interacts with. GWT provides a huge list of widely used and common elements varying from basic to complex which we will cover in this tutorial.
- **Layouts**: They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface). This part will be covered in Layout chapter.
- **Behavior**: These are events which occur when the user interacts with UI elements. This part will be covered in Event Handling chapter.

## Event and Listener (Java Event Handling)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. 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

MouseEvent	MouseListener
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**

Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

- 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

```

1. import java.awt.*;
2. import java.awt.event.*;
3. class AEvent extends Frame implements ActionListener{
4. TextField tf;
5. AEvent(){
6.
7. //create components
8. tf=new TextField();
9. tf.setBounds(60,50,170,20);
10. Button b=new Button("click me");
11. b.setBounds(100,120,80,30);
12.
13. //register listener
14. b.addActionListener(this); //passing current instance
15.
16. //add components and set size, layout and visibility
17. add(b);add(tf);
18. setSize(300,300);
19. setLayout(null);
20. setVisible(true);
21. }
22. public void actionPerformed(ActionEvent e){

```

Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

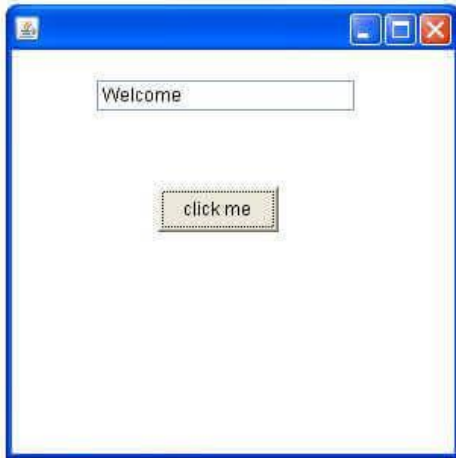


```

23. tf.setText("Welcome");
24. }
25. public static void main(String args[]){
26. new AEvent();
27. }
28. }

```

**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.



## 2) Java event handling by outer class

```

1. import java.awt.*;
2. import java.awt.event.*;
3. class AEvent2 extends Frame{
4. TextField tf;
5. AEvent2(){
6. //create components
7. tf=new TextField();
8. tf.setBounds(60,50,170,20);
9. Button b=new Button("click me");
10. b.setBounds(100,120,80,30);
11. //register listener
12. Outer o=new Outer(this);
13. b.addActionListener(o);//passing outer class instance
14. //add components and set size, layout and visibility
15. add(b);add(tf);
16. setSize(300,300);

```

Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

```

17. setLayout(null);
18. setVisible(true);
19. }
20. public static void main(String args[]){
21. new AEvent2();
22. }
23. }
1. import java.awt.event.*;
2. class Outer implements ActionListener{
3. AEvent2 obj;
4. Outer(AEvent2 obj){
5. this.obj=obj;
6. }
7. public void actionPerformed(ActionEvent e){
8. obj.tf.setText("welcome");
9. }
10. }

```

---

### 3) Java event handling by anonymous class

```

1. import java.awt.*;
2. import java.awt.event.*;
3. class AEvent3 extends Frame{
4. TextField tf;
5. AEvent3(){
6. tf=new TextField();
7. tf.setBounds(60,50,170,20);
8. Button b=new Button("click me");
9. b.setBounds(50,120,80,30);
10.
11. b.addActionListener(new ActionListener(){
12. public void actionPerformed(){
13. tf.setText("hello");
14. }
15. });
16. add(b);add(tf);
17. setSize(300,300);
18. setLayout(null);
19. setVisible(true);
20. }
21. public static void main(String args[]){
22. new AEvent3();
23. }

```

Every AWT controls inherits properties from Component class.

Sr. No.	Control & Description
1	<p><u>Component</u></p> <p>A Component is an abstract super class for GUI controls and it represents an object with graphical representation.</p>

## AWT UI Elements:

Following is the list of commonly used controls while designed GUI using AWT.

Sr. No.	Control & Description
1	<p><u>Label</u></p> <p>A Label object is a component for placing text in a container.</p>
2	<p><u>Button</u></p> <p>This class creates a labeled button.</p>
3	<p><u>Check Box</u></p> <p>A check box is a graphical component that can be in either an <b>on</b> (true) or <b>off</b> (false) state.</p>
4	<p><u>Check Box Group</u></p> <p>The CheckboxGroup class is used to group the set of checkbox.</p>
5	<p><u>List</u></p> <p>The List component presents the user with a scrolling list of text items.</p>
6	<p><u>Text Field</u></p> <p>A TextField object is a text component that allows for the editing of a single line of text.</p>

7	<u>Text Area</u> A TextArea object is a text component that allows for the editing of a multiple lines of text.
8	<u>Choice</u> A Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu.
9	<u>Canvas</u> A Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.
10	<u>Image</u> An Image control is superclass for all image classes representing graphical images.
11	<u>Scroll Bar</u> A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
12	<u>Dialog</u> A Dialog control represents a top-level window with a title and a border used to take some form of input from the user.
13	<u>File Dialog</u> A FileDialog control represents a dialog window from which the user can select a file.

## AWT Event Handling

### What is an Event?

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.

### Types of Event

The events can be broadly classified into two categories:

- **Foreground Events** - Those events which require the direct interaction of user. They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.
- **Background Events** - Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

## What is Event Handling?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as event handler that is executed when an event occurs. Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events. Let's have a brief introduction to this model.

The Delegation Event Model has the following key participants namely:

- **Source** - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to its handler. Java provides as with classes for source object.
- **Listener** - It is also known as event handler. Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received, the listener processes the event and then returns.

The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to the separate piece of code. In this model, Listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listeners that want to receive them.

## Steps involved in event handling

- The User clicks the button and the event is generated.
- Now the object of concerned event class is created automatically and information about the source and the event gets populated with in same object.
- Event object is forwarded to the method of registered listener class.
- the method is now get executed and returns.

## Points to remember about listener

- In order to design a listener class we have to develop some listener interfaces. These Listener interfaces forecast some public abstract callback methods which must be implemented by the listener class.
- If you do not implement the any if the predefined interfaces then your class can not act as a listener class for a source object.

## Callback Methods

These are the methods that are provided by API provider and are defined by the application programmer and invoked by the application developer. Here the callback methods represents an event method. In response to an event java jre will fire callback method. All such callback methods are provided in listener interfaces.

If a component wants some listener will listen to it's events the the source must register itself to the listener.

## AWT Event Classes

The Event classes represent the event. Java provides us various Event classes but we will discuss those which are more frequently used.

### EventObject class

It is the root class from which all event state objects shall be derived. All Events are constructed with a reference to the object, the **source**, that is logically deemed to be the object upon which the Event in question initially occurred upon. This class is defined in java.util package.

### Class declaration

Following is the declaration for **java.util.EventObject** class:

```
public class EventObject
    extends Object
    implements Serializable
```

### Field

Following are the fields for **java.util.EventObject** class:

- **protected Object source** -- The object on which the Event initially occurred.

### Class constructors

S.N.	Constructor & Description
1	<b>EventObject(Object source)</b> Constructs a prototypical Event.

### Class methods

S.N.	Method & Description
------	----------------------

Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

1	<b>Object getSource()</b> The object on which the Event initially occurred.
2	<b>String toString()</b> Returns a String representation of this EventObject.

## Methods inherited

This class inherits methods from the following classes:

- java.lang.Object

## AWT Event Classes:

Following is the list of commonly used event classes.

Sr. No.	Control & Description
1	<u>AWTEvent</u> It is the root event class for all AWT events. This class and its subclasses supercede the original java.awt.Event class.
2	<u>ActionEvent</u> The ActionEvent is generated when button is clicked or the item of a list is double clicked.
3	<u>InputEvent</u> The InputEvent class is root event class for all component-level input events.
4	<u>KeyEvent</u> On entering the character the Key event is generated.
5	<u>MouseEvent</u> This event indicates a mouse action occurred in a component.
6	<u>TextEvent</u> The object of this class represents the text events.

Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

7	<u>WindowEvent</u> The object of this class represents the change in state of a window.
8	<u>AdjustmentEvent</u> The object of this class represents the adjustment event emitted by Adjustable objects.
9	<u>ComponentEvent</u> The object of this class represents the change in state of a window.
10	<u>ContainerEvent</u> The object of this class represents the change in state of a window.
11	<u>MouseEvent</u> The object of this class represents the change in state of a window.
12	<u>PaintEvent</u> The object of this class represents the change in state of a window.

## Java ActionListener Interface

The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event [package](#). It has only one method: actionPerformed().

### actionPerformed() method

The actionPerformed() method is invoked automatically whenever you click on the registered component.

1. **public abstract void** actionPerformed(ActionEvent e);

## How to write ActionListener

The common approach is to implement the ActionListener. If you implement the ActionListener class, you need to follow 3 steps:

1) Implement the ActionListener interface in the class:

1. **public class** ActionListenerExample Implements ActionListener  
Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)



2) Register the component with the Listener:

```
1. component.addActionListener(instanceOfListenerclass);
```

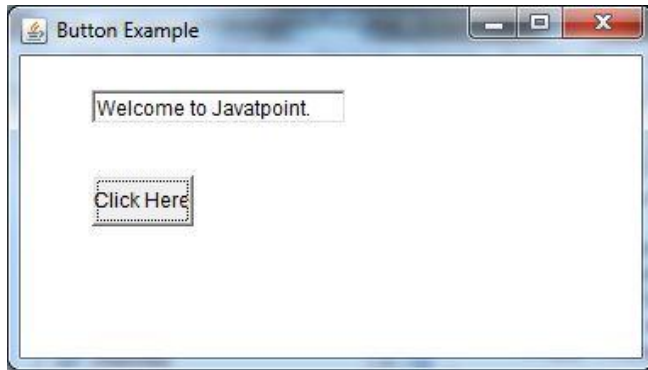
3) Override the actionPerformed() method:

```
1. public void actionPerformed(ActionEvent e){  
2.     //Write the code here  
3. }
```

## Java ActionListener Example: On Button click

```
1. import java.awt.*;  
2. import java.awt.event.*;  
3. //1st step  
4. public class ActionListenerExample implements ActionListener{  
5. public static void main(String[] args) {  
6.     Frame f=new Frame("ActionListener Example");  
7.     final TextField tf=new TextField();  
8.     tf.setBounds(50,50, 150,20);  
9.     Button b=new Button("Click Here");  
10.    b.setBounds(50,100,60,30);  
11.    //2nd step  
12.    b.addActionListener(this);  
13.    f.add(b);f.add(tf);  
14.    f.setSize(400,400);  
15.    f.setLayout(null);  
16.    f.setVisible(true);  
17. }  
18. //3rd step  
19. public void actionPerformed(ActionEvent e){  
20.     tf.setText("Welcome to Javatpoint.");  
21. }  
22. }
```

Output:



## Java ActionListener Example: Using Anonymous class

We can also use the anonymous class to implement the ActionListener. It is the shorthand way, so you do not need to follow the 3 steps:

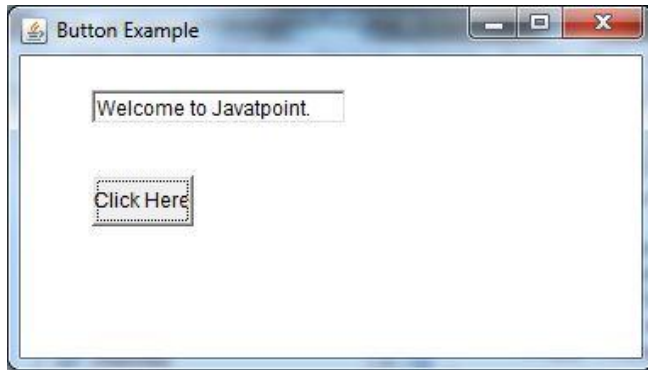
1. `b.addActionListener(new ActionListener(){`
2. `public void actionPerformed(ActionEvent e){`
3. `tf.setText("Welcome to Javatpoint.");`
4. `}`
5. `});`

Let us see the full code of ActionListener using anonymous class.

1. `import java.awt.*;`
2. `import java.awt.event.*;`
3. `public class ActionListenerExample {`
4. `public static void main(String[] args) {`
5. `Frame f=new Frame("ActionListener Example");`
6. `final TextField tf=new TextField();`
7. `tf.setBounds(50,50, 150,20);`
8. `Button b=new Button("Click Here");`
9. `b.setBounds(50,100,60,30);`
10.
11. `b.addActionListener(new ActionListener(){`
12. `public void actionPerformed(ActionEvent e){`
13. `tf.setText("Welcome to Javatpoint.");`
14. `}`
15. `});`
16. `f.add(b);f.add(tf);`
17. `f.setSize(400,400);`
18. `f.setLayout(null);`
19. `f.setVisible(true);`
20. `}`
21. `}`

Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

Output:



## Java AWT Label

The [object](#) of 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 an application but a user cannot edit it directly.

## AWT Label Class Declaration

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

## Java AWT Label Example with ActionListener

1. **import** java.awt.\*;
2. **import** java.awt.event.\*;
3. **public class** LabelExample **extends** Frame **implements** ActionListener{
4.     TextField tf; Label l; Button b;
5.     LabelExample(){
6.         tf=**new** TextField();
7.         tf.setBounds(50,50, 150,20);
8.         l=**new** Label();
9.         l.setBounds(50,100, 250,20);
10.         b=**new** Button("Find IP");
11.         b.setBounds(50,150,60,30);
12.         b.addActionListener(**this**);
13.         add(b);add(tf);add(l);
14.         setSize(400,400);
15.         setLayout(**null**);
16.         setVisible(**true**);
17.     }
18.     **public void** actionPerformed(ActionEvent e) {
19.         **try**{

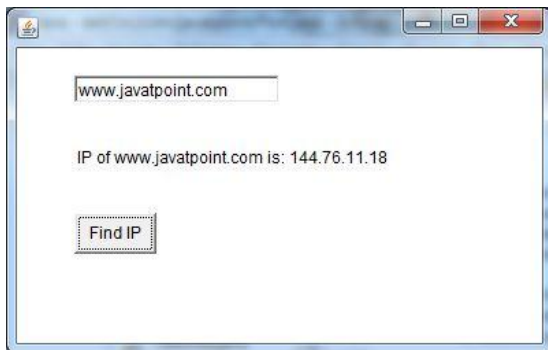
Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

```

20.     String host=tf.getText();
21.     String ip=java.net.InetAddress.getByName(host).getHostAddress();
22.     l.setText("IP of "+host+" is: "+ip);
23.     }catch(Exception ex){System.out.println(ex);}
24. }
25. public static void main(String[] args) {
26.     new LabelExample();
27. }
28. }

```

Output:



## Java AWT TextField

The [object](#) of a TextField class is a text component that allows the editing of a single line text. It inherits TextComponent class.

## AWT TextField Class Declaration

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

## Java AWT TextField Example with ActionListener

```

1. import java.awt.*;
2. import java.awt.event.*;
3. public class TextFieldExample extends Frame implements ActionListener{
4.     TextField tf1,tf2,tf3;
5.     Button b1,b2;
6.     TextFieldExample(){
7.         tf1=new TextField();
8.         tf1.setBounds(50,50,150,20);
9.         tf2=new TextField();
10.        tf2.setBounds(50,100,150,20);
11.        tf3=new TextField();

```

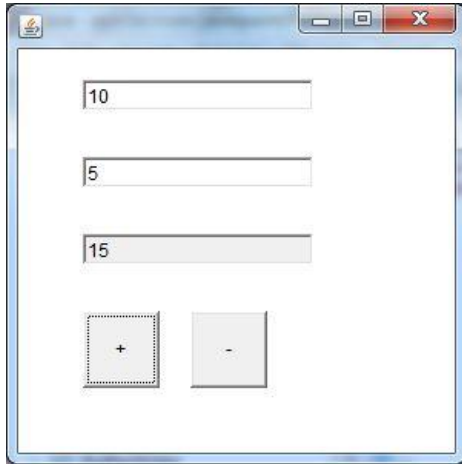
Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

```

12.    tf3.setBounds(50,150,150,20);
13.    tf3.setEditable(false);
14.    b1=new Button("+");
15.    b1.setBounds(50,200,50,50);
16.    b2=new Button("-");
17.    b2.setBounds(120,200,50,50);
18.    b1.addActionListener(this);
19.    b2.addActionListener(this);
20.    add(tf1);add(tf2);add(tf3);add(b1);add(b2);
21.    setSize(300,300);
22.    setLayout(null);
23.    setVisible(true);
24. }
25. public void actionPerformed(ActionEvent e) {
26.     String s1=tf1.getText();
27.     String s2=tf2.getText();
28.     int a=Integer.parseInt(s1);
29.     int b=Integer.parseInt(s2);
30.     int c=0;
31.     if(e.getSource()==b1){
32.         c=a+b;
33.     }else if(e.getSource()==b2){
34.         c=a-b;
35.     }
36.     String result=String.valueOf(c);
37.     tf3.setText(result);
38. }
39. public static void main(String[] args) {
40.     new TextFieldExample();
41. }
42. }

```

Output:



## Java AWT TextArea

The [object](#) of a TextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits TextComponent class.

## AWT TextArea Class Declaration

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

## Java AWT TextArea Example with ActionListener

1. **import** java.awt.\*;
2. **import** java.awt.event.\*;
3. **public class** TextAreaExample **extends** Frame **implements** ActionListener{
4. Label l1,l2;
5. TextArea area;
6. Button b;
7. TextAreaExample(){
8.   l1=**new** Label();
9.   l1.setBounds(50,50,100,30);
10.   l2=**new** Label();
11.   l2.setBounds(160,50,100,30);
12.   area=**new** TextArea();
13.   area.setBounds(20,100,300,300);
14.   b=**new** Button("Count Words");
15.   b.setBounds(100,400,100,30);
16.   b.addActionListener(**this**);
17.   add(l1);add(l2);add(area);add(b);
18.   setSize(400,450);
19.   setLayout(**null**);
20.   setVisible(**true**);

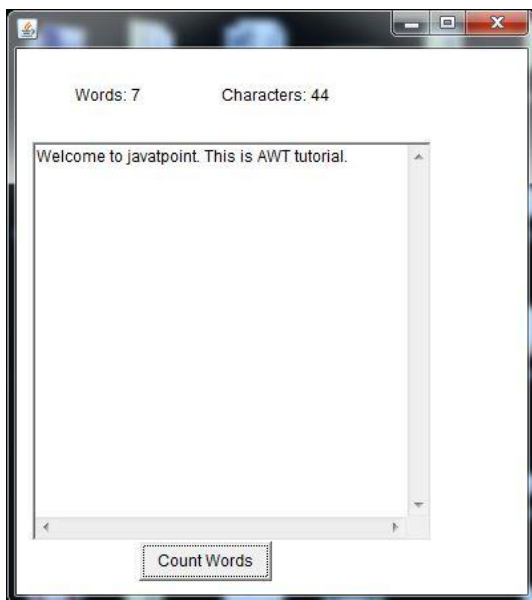
Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

```

21. }
22. public void actionPerformed(ActionEvent e){
23.     String text=area.getText();
24.     String words[]=text.split("\\s");
25.     l1.setText("Words: "+words.length);
26.     l2.setText("Characters: "+text.length());
27. }
28. public static void main(String[] args) {
29.     new TextAreaExample();
30. }
31. }

```

Output:




---

## Java MouseListener Interface

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 are given below:

1. **public abstract void** mouseClicked(MouseEvent e);
2. **public abstract void** mouseEntered(MouseEvent e);

Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

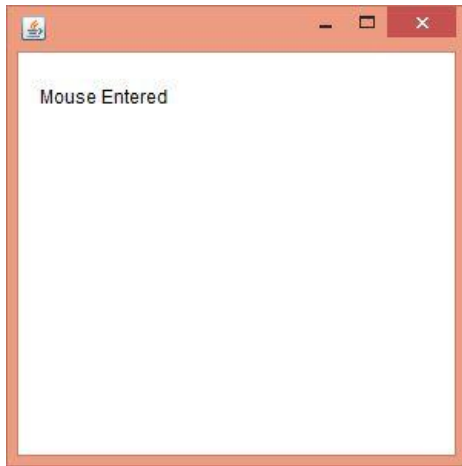
3. **public abstract void** mouseExited(MouseEvent e);
4. **public abstract void** mousePressed(MouseEvent e);
5. **public abstract void** mouseReleased(MouseEvent e);

## Java MouseListener Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseListenerExample extends Frame implements MouseListener{
4.     Label l;
5.     MouseListenerExample(){
6.         addMouseListener(this);
7.
8.         l=new Label();
9.         l.setBounds(20,50,100,20);
10.        add(l);
11.        setSize(300,300);
12.        setLayout(null);
13.        setVisible(true);
14.    }
15.    public void mouseClicked(MouseEvent e) {
16.        l.setText("Mouse Clicked");
17.    }
18.    public void mouseEntered(MouseEvent e) {
19.        l.setText("Mouse Entered");
20.    }
21.    public void mouseExited(MouseEvent e) {
22.        l.setText("Mouse Exited");
23.    }
24.    public void mousePressed(MouseEvent e) {
25.        l.setText("Mouse Pressed");
26.    }
27.    public void mouseReleased(MouseEvent e) {
28.        l.setText("Mouse Released");
29.    }
30.    public static void main(String[] args) {
31.        new MouseListenerExample();
32.    }
33. }
```

Output:

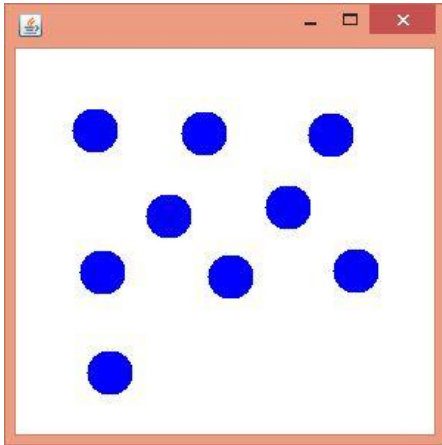




## Java MouseListener Example 2

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseListenerExample2 extends Frame implements MouseListener{
4.     MouseListenerExample2(){
5.         addMouseListener(this);
6.
7.         setSize(300,300);
8.         setLayout(null);
9.         setVisible(true);
10.    }
11.    public void mouseClicked(MouseEvent e) {
12.        Graphics g=getGraphics();
13.        g.setColor(Color.BLUE);
14.        g.fillOval(e.getX(),e.getY(),30,30);
15.    }
16.    public void mouseEntered(MouseEvent e) {}
17.    public void mouseExited(MouseEvent e) {}
18.    public void mousePressed(MouseEvent e) {}
19.    public void mouseReleased(MouseEvent e) {}
20.
21.    public static void main(String[] args) {
22.        new MouseListenerExample2();
23.    }
24. }
```

Output:



## Java MouseMotionListener Interface

The Java MouseMotionListener is notified whenever you move or drag mouse. It is notified against MouseEvent. The MouseMotionListener interface is found in java.awt.event package. It has two methods.

### Methods of MouseMotionListener interface

The signature of 2 methods found in MouseMotionListener interface are given below:

1. **public abstract void** mouseDragged(MouseEvent e);
2. **public abstract void** mouseMoved(MouseEvent e);

### Java MouseMotionListener Example

1. **import** java.awt.\*;
2. **import** java.awt.event.\*;
3. **public class** MouseMotionListenerExample **extends** Frame **implements** MouseMotionListener{
4.     MouseMotionListenerExample(){
5.         addMouseMotionListener(**this**);
6.     }
7.     setSize(**300,300**);
8.     setLayout(**null**);
9.     setVisible(**true**);
10. }  
11. **public void** mouseDragged(MouseEvent e) {
12.     Graphics g=getGraphics();
13.     g.setColor(Color.BLUE);
14.     g.fillOval(e.getX(),e.getY(),**20,20**);
15. }

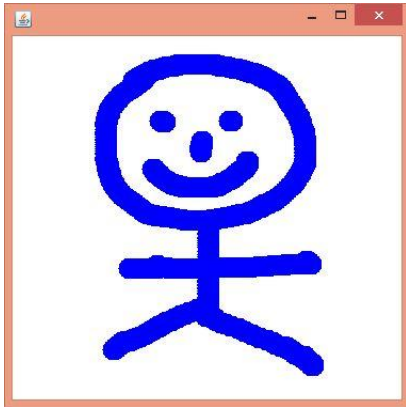
Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

```

16. public void mouseMoved(MouseEvent e) {}
17.
18. public static void main(String[] args) {
19.     new MouseMotionListenerExample();
20. }
21. }

```

Output:



## Java MouseMotionListener Example 2

```

1. import java.awt.*;
2. import java.awt.event.MouseEvent;
3. import java.awt.event.MouseMotionListener;
4. public class Paint extends Frame implements MouseMotionListener{
5.     Label l;
6.     Color c=Color.BLUE;
7.     Paint(){
8.         l=new Label();
9.         l.setBounds(20,40,100,20);
10.        add(l);
11.
12.        addMouseMotionListener(this);
13.
14.        setSize(400,400);
15.        setLayout(null);
16.        setVisible(true);
17. }
18. public void mouseDragged(MouseEvent e) {
19.     l.setText("X="+e.getX()+" Y="+e.getY());
20.     Graphics g=getGraphics();

```

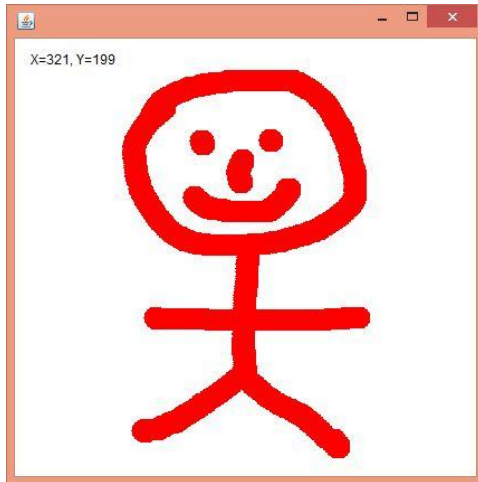
Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

```

21. g.setColor(Color.RED);
22. g.fillOval(e.getX(),e.getY(),20,20);
23. }
24. public void mouseMoved(MouseEvent e) {
25.     l.setText("X="+e.getX()+" , Y="+e.getY());
26. }
27. public static void main(String[] args) {
28.     new Paint();
29. }
30. }

```

Output:



## Java ItemListener Interface

The Java ItemListener is notified whenever you click on the checkbox. It is notified against ItemEvent. The ItemListener interface is found in java.awt.event package. It has only one method: itemStateChanged().

### itemStateChanged() method

The itemStateChanged() method is invoked automatically whenever you click or unclick on the registered checkbox component.

```

1. public abstract void itemStateChanged(ItemEvent e);

```

## Java ItemListener Example

```

1. import java.awt.*;
2. import java.awt.event.*;
3. public class ItemListenerExample implements ItemListener{
4.     Checkbox checkBox1,checkBox2;

```

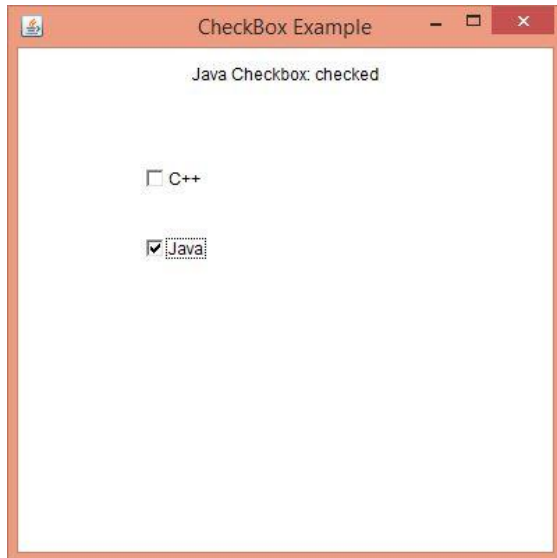
Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

```

5.   Label label;
6.   ItemListenerExample(){
7.       Frame f= new Frame("CheckBox Example");
8.       label = new Label();
9.       label.setAlignment(Label.CENTER);
10.      label.setSize(400,100);
11.      checkBox1 = new Checkbox("C++");
12.      checkBox1.setBounds(100,100, 50,50);
13.      checkBox2 = new Checkbox("Java");
14.      checkBox2.setBounds(100,150, 50,50);
15.      f.add(checkBox1); f.add(checkBox2); f.add(label);
16.      checkBox1.addItemListener(this);
17.      checkBox2.addItemListener(this);
18.      f.setSize(400,400);
19.      f.setLayout(null);
20.      f.setVisible(true);
21.  }
22.  public void itemStateChanged(ItemEvent e) {
23.      if(e.getSource()==checkBox1)
24.          label.setText("C++ Checkbox: "
25.          + (e.getStateChange()==1?"checked":"unchecked"));
26.      if(e.getSource()==checkBox2)
27.          label.setText("Java Checkbox: "
28.          + (e.getStateChange()==1?"checked":"unchecked"));
29.  }
30. public static void main(String args[])
31. {
32.     new ItemListenerExample();
33. }
34. }

```

Output:



## Java KeyListener Interface

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. It has three methods.

### Methods of KeyListener interface

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

1. **public abstract void** keyPressed(KeyEvent e);
2. **public abstract void** keyReleased(KeyEvent e);
3. **public abstract void** keyTyped(KeyEvent e);

### Java KeyListener Example

1. **import** java.awt.\*;
2. **import** java.awt.event.\*;
3. **public class** KeyListenerExample **extends** Frame **implements** KeyListener{
4.     Label l;
5.     TextArea area;
6.     KeyListenerExample(){
- 7.
8.         l=**new** Label();
9.         l.setBounds(20,50,100,20);
10.        area=**new** TextArea();
11.        area.setBounds(20,80,300, 300);
12.        area.addKeyListener(**this**);
- 13.

Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

```

14.     add(l);add(area);
15.     setSize(400,400);
16.     setLayout(null);
17.     setVisible(true);
18. }
19. public void keyPressed(KeyEvent e) {
20.     l.setText("Key Pressed");
21. }
22. public void keyReleased(KeyEvent e) {
23.     l.setText("Key Released");
24. }
25. public void keyTyped(KeyEvent e) {
26.     l.setText("Key Typed");
27. }
28.
29. public static void main(String[] args) {
30.     new KeyListenerExample();
31. }
32. }

```

Output:



## Java KeyListener Example 2: Count Words & Characters

1. **import** java.awt.\*;
2. **import** java.awt.event.\*;
3. **public class** KeyListenerExample **extends** Frame **implements** KeyListener{

Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

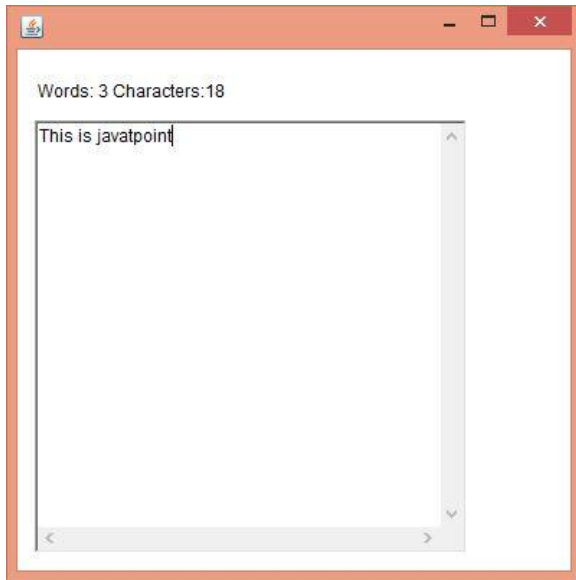
```

4.   Label l;
5.   TextArea area;
6.   KeyListenerExample(){
7.
8.       l=new Label();
9.       l.setBounds(20,50,200,20);
10.      area=new TextArea();
11.      area.setBounds(20,80,300, 300);
12.      area.addKeyListener(this);
13.
14.      add(l);add(area);
15.      setSize(400,400);
16.      setLayout(null);
17.      setVisible(true);
18.  }
19.  public void keyPressed(KeyEvent e) {}
20.  public void keyReleased(KeyEvent e) {
21.      String text=area.getText();
22.      String words[]=text.split("\\s");
23.      l.setText("Words: "+words.length+" Characters:"+text.length());
24.  }
25.  public void keyTyped(KeyEvent e) {}
26.
27.  public static void main(String[] args) {
28.      new KeyListenerExample();
29.  }
30. }

```

Output:





## Java WindowListener Interface

The Java WindowListener is notified whenever you change the state of window. It is notified against WindowEvent. The WindowListener interface is found in java.awt.event package. It has three methods.

## Methods of WindowListener interface

The signature of 7 methods found in WindowListener interface are given below:

1. **public abstract void** windowActivated(WindowEvent e);
2. **public abstract void** windowClosed(WindowEvent e);
3. **public abstract void** windowClosing(WindowEvent e);
4. **public abstract void** windowDeactivated(WindowEvent e);
5. **public abstract void** windowDeiconified(WindowEvent e);
6. **public abstract void** windowIconified(WindowEvent e);
7. **public abstract void** windowOpened(WindowEvent e);

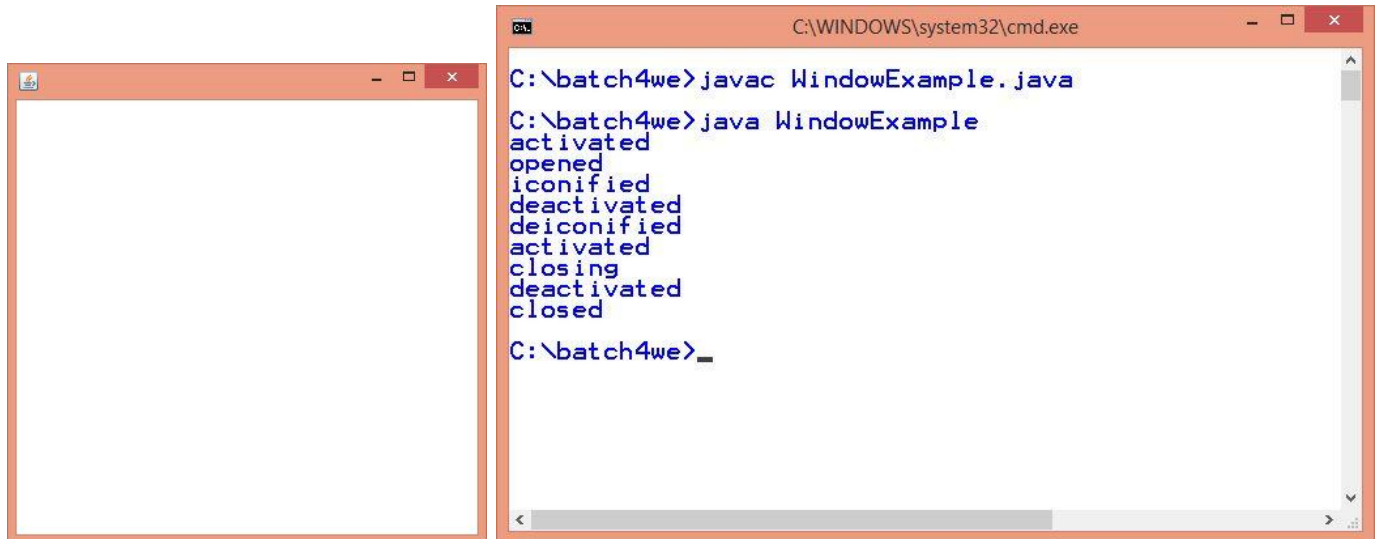
## Java WindowListener Example

1. **import** java.awt.\*;
2. **import** java.awt.event.WindowEvent;
3. **import** java.awt.event.WindowListener;
4. **public class** WindowExample **extends** Frame **implements** WindowListener{
5.     WindowExample(){
6.         addWindowListener(**this**);
7.     }
8.     setSize(400,400);
9.     setLayout(**null**);

Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

```
10.     setVisible(true);
11. }
12.
13. public static void main(String[] args) {
14.     new WindowExample();
15. }
16. public void windowActivated(WindowEvent arg0) {
17.     System.out.println("activated");
18. }
19. public void windowClosed(WindowEvent arg0) {
20.     System.out.println("closed");
21. }
22. public void windowClosing(WindowEvent arg0) {
23.     System.out.println("closing");
24.     dispose();
25. }
26. public void windowDeactivated(WindowEvent arg0) {
27.     System.out.println("deactivated");
28. }
29. public void windowDeiconified(WindowEvent arg0) {
30.     System.out.println("deiconified");
31. }
32. public void windowIconified(WindowEvent arg0) {
33.     System.out.println("iconified");
34. }
35. public void windowOpened(WindowEvent arg0) {
36.     System.out.println("opened");
37. }
38. }
```

Output:



```
C:\WINDOWS\system32\cmd.exe
C:\batch4we>javac WindowExample.java
C:\batch4we>java WindowExample
activated
opened
iconified
deactivated
deiconified
activated
closing
deactivated
closed
C:\batch4we>_
```

## Java Adapter Classes

Java adapter classes *provide the default implementation of listener [interfaces](#)*. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

The adapter classes are found in **java.awt.event**, **java.awt.dnd** and **javax.swing.event** [packages](#). The Adapter classes with their corresponding listener interfaces are given below.

### java.awt.event Adapter classes

Adapter class	Listener <a href="#">interface</a>
WindowAdapter	<a href="#">WindowListener</a>
KeyAdapter	<a href="#">KeyListener</a>
MouseAdapter	<a href="#">MouseListener</a>
MouseMotionAdapter	<a href="#">MouseMotionListener</a>
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener

HierarchyBoundsAdapter

HierarchyBoundsListener

## java.awt.dnd Adapter classes

Adapter class	Listener interface
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

## javax.swing.event Adapter classes

Adapter class	Listener interface
MouseInputAdapter	MouseListener
InternalFrameAdapter	InternalFrameListener

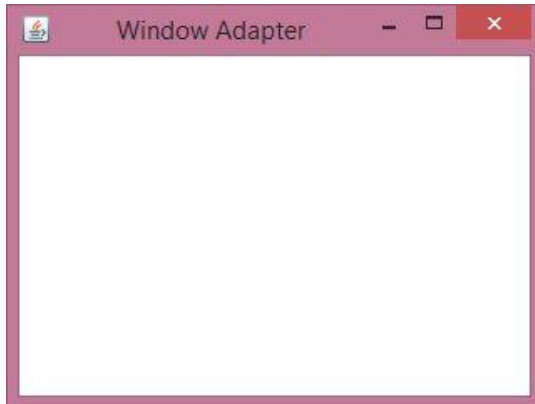
## Java WindowAdapter Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class AdapterExample{
4.     Frame f;
5.     AdapterExample(){
6.         f=new Frame("Window Adapter");
7.         f.addWindowListener(new WindowAdapter(){
8.             public void windowClosing(WindowEvent e) {
9.                 f.dispose();
10.            }
11.        });
12.
13.        f.setSize(400,400);
14.        f.setLayout(null);
15.        f.setVisible(true);
16.    }
17. public static void main(String[] args) {
18.     new AdapterExample();
19. }
```

Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

20. }

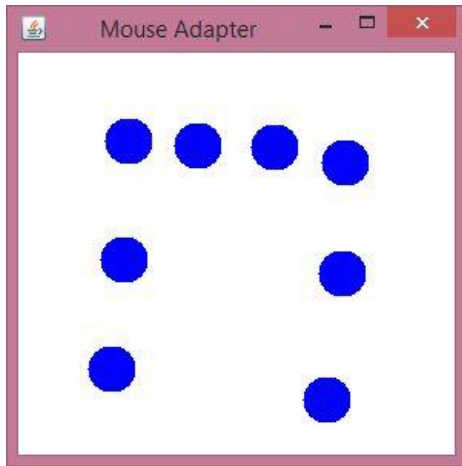
Output:



## Java MouseAdapter Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseAdapterExample extends MouseAdapter{
4.     Frame f;
5.     MouseAdapterExample(){
6.         f=new Frame("Mouse Adapter");
7.         f.addMouseListener(this);
8.
9.         f.setSize(300,300);
10.        f.setLayout(null);
11.        f.setVisible(true);
12.    }
13.    public void mouseClicked(MouseEvent e) {
14.        Graphics g=f.getGraphics();
15.        g.setColor(Color.BLUE);
16.        g.fillOval(e.getX(),e.getY(),30,30);
17.    }
18.
19. public static void main(String[] args) {
20.     new MouseAdapterExample();
21. }
22. }
```

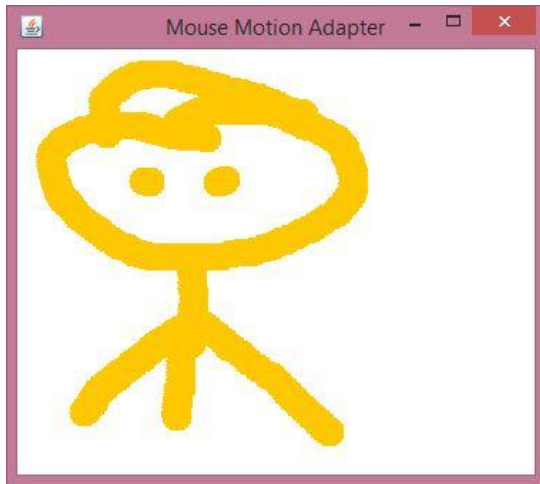
Output:



## Java MouseMotionAdapter Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseMotionAdapterExample extends MouseMotionAdapter{
4.     Frame f;
5.     MouseMotionAdapterExample(){
6.         f=new Frame("Mouse Motion Adapter");
7.         f.addMouseListener(this);
8.
9.         f.setSize(300,300);
10.        f.setLayout(null);
11.        f.setVisible(true);
12.    }
13. public void mouseDragged(MouseEvent e) {
14.     Graphics g=f.getGraphics();
15.     g.setColor(Color.ORANGE);
16.     g.fillOval(e.getX(),e.getY(),20,20);
17. }
18. public static void main(String[] args) {
19.     new MouseMotionAdapterExample();
20. }
21. }
```

Output:



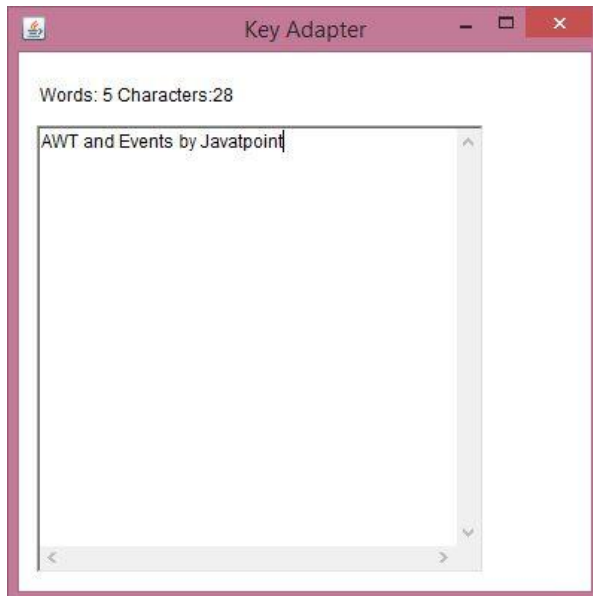
## Java KeyAdapter Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class KeyAdapterExample extends KeyAdapter{
4.     Label l;
5.     TextArea area;
6.     Frame f;
7.     KeyAdapterExample(){
8.         f=new Frame("Key Adapter");
9.         l=new Label();
10.        l.setBounds(20,50,200,20);
11.        area=new TextArea();
12.        area.setBounds(20,80,300, 300);
13.        area.addKeyListener(this);
14.
15.        f.add(l);f.add(area);
16.        f.setSize(400,400);
17.        f.setLayout(null);
18.        f.setVisible(true);
19.    }
20.    public void keyReleased(KeyEvent e) {
21.        String text=area.getText();
22.        String words[]=text.split("\\s");
23.        l.setText("Words: "+words.length+" Characters:"+text.length());
24.    }
25.
26.    public static void main(String[] args) {
27.        new KeyAdapterExample();
28.    }
```

Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

29. }

Output:



## How to close AWT Window in Java

We can close the AWT Window or Frame by calling *dispose()* or *System.exit()* inside *windowClosing()* method. The *windowClosing()* method is found in **WindowListener** interface and **WindowAdapter** class.

The *WindowAdapter* class implements *WindowListener* interfaces. It provides the default implementation of all the 7 methods of *WindowListener* interface. To override the *windowClosing()* method, you can either use *WindowAdapter* class or *WindowListener* interface.

If you implement the *WindowListener* interface, you will be forced to override all the 7 methods of *WindowListener* interface. So it is better to use *WindowAdapter* class.

## Different ways to override *windowClosing()* method

There are many ways to override *windowClosing()* method:

- By anonymous class
- By inheriting *WindowAdapter* class
- By implementing *WindowListener* interface

## Close AWT Window Example 1: Anonymous class

1. **import** java.awt.\*;
2. **import** java.awt.event.WindowEvent;
3. **import** java.awt.event.WindowListener;

Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

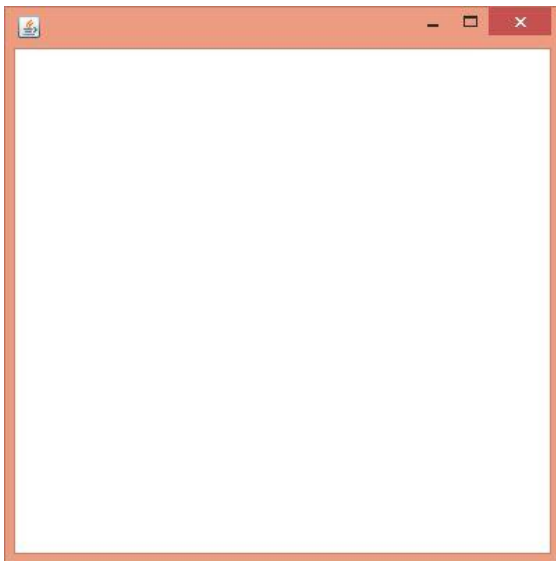


```

4. public class WindowExample extends Frame{
5.     WindowExample(){
6.         addWindowListener(new WindowAdapter(){
7.             public void windowClosing(WindowEvent e) {
8.                 dispose();
9.             }
10.        });
11.        setSize(400,400);
12.        setLayout(null);
13.        setVisible(true);
14.    }
15. public static void main(String[] args) {
16.     new WindowExample();
17. }

```

Output:



## Close AWT Window Example 2: extending WindowAdapter

```

1. import java.awt.*;
2. import java.awt.event.*;
3. public class AdapterExample extends WindowAdapter{
4.     Frame f;
5.     AdapterExample(){
6.         f=new Frame();
7.         f.addWindowListener(this);
8.
9.         f.setSize(400,400);

```

Reference:- [www.javatpoint.com](http://www.javatpoint.com) and [www.tutorialspoint.com](http://www.tutorialspoint.com)

```

10.     f.setLayout(null);
11.     f.setVisible(true);
12. }
13. public void windowClosing(WindowEvent e) {
14.     f.dispose();
15. }
16. public static void main(String[] args) {
17.     new AdapterExample();
18. }
19. }

```

## Close AWT Window Example 3: implementing WindowListener

```

1. import java.awt.*;
2. import java.awt.event.WindowEvent;
3. import java.awt.event.WindowListener;
4. public class WindowExample extends Frame implements WindowListener{
5.     WindowExample(){
6.         addWindowListener(this);
7.
8.         setSize(400,400);
9.         setLayout(null);
10.        setVisible(true);
11.    }
12.
13. public static void main(String[] args) {
14.     new WindowExample();
15. }
16. public void windowActivated(WindowEvent e) {}
17. public void windowClosed(WindowEvent e) {}
18. public void windowClosing(WindowEvent e) {
19.     dispose();
20. }
21. public void windowDeactivated(WindowEvent e) {}
22. public void windowDeiconified(WindowEvent e) {}
23. public void windowIconified(WindowEvent e) {}
24. public void windowOpened(WindowEvent arg0) {}
25. }

```