//MultiThreading

class mythreadA extends Thread

{

public void run()

{

for(int i=0;i<5;i++)

{

System.out.println("i="+i);

if(i==0)

{

yield();

}

}

}

}

class mythreadB extends Thread

{

public void run()

{

for(int j=0;j<5;j++)

{

System.out.println("j="+j);

If(j==3)

{

try

{Thread.sleep(1000);}

catch(Exception e){}

}

}

}

}

class mythreadC extends Thread

{

public void run()

{

for(int k=0;k<5;k++)

{

System.out.println("k= "+k);

if(k==2)

{

stop();

}

}

}

}

class threaddemo

{

public static void main(String args[])

{

mythreadA a1=new mythreadA();

mythreadB b1=new mythreadB();

mythreadC c1=new mythreadC();

c1.start();

a1.start();

b1.start();

}

}

// End of multithreading

class per

{

//synchronized

void display(String s)

{

System.out.print("(");

System.out.print(s);

try

{

Thread.sleep(1000);

}

catch(Exception e)

{

}

System.out.print(")");

}

}

class mt extends Thread

{

String sx;

per p1;

public mt(per p2,String s2)

{

p1=p2;

sx=s2;

start();

}

public void run()

{

p1.display(sx);

}

}

class thrsyn

{

public static void main(String arg[])

{

per px=new per();

mt t1=new mt(px,"hello");

mt t2=new mt(px,"test");

}

}

==============

class test

{

public static void main(String[] args)

{

String s="PUsoftech";

String s2="ucoepatiala";

String s3=s2.substring(2,5);

String s4=s.concat(s2);

int i;

int j;

System.out.println("hello "+s);

System.out.println("length of s "+s.length());

System.out.println("s2.subString(2,5) "+s3);

System.out.println("s2.charAt(2) "+s2.charAt(2));

i=s.compareTo(s2);

System.out.println("s.compareTo(s2) "+i);

System.out.println("concate "+s4);

j=s.indexOf('o');

System.out.println("index of o "+j);

String s5=s.replace('s','S');

System.out.println("after replace "+s5);

String s6=s.toUpperCase();

System.out.println("s6 uppercase "+s6);

String s7=s6.toLowerCase();

System.out.println("s6 lowercase "+s7);

int m=123;

String s8=String.valueOf(m);

System.out.println("valueof "+s8);

}}

class test2

{

public static void main(String[] args)

{

StringBuffer st=new StringBuffer(args[0]);

String s=String.valueOf(st);

st.reverse();

//System.out.println("\nString reverse: "+st);

if(s.compareTo(String.valueOf(st))==0)

System.out.println("\npal");

else

System.out.println("\nnot pal");

} }

class stringfun3

{public static void main(String[] args)

{StringBuffer strBuf1 = new StringBuffer("PUsoftech");

StringBuffer strBuf2 = new StringBuffer(100);

StringBuffer strBuf3 = new StringBuffer();

System.out.println("strBuf1 : " + strBuf1);

System.out.println("strBuf1 capacity :"+ strBuf1.capacity());

System.out.println("strBuf2 capacity :"+ strBuf2.capacity());

System.out.println("strBuf3 capacity :"+ strBuf3.capacity());

System.out.println("strBuf1 length :" + strBuf1.length());

System.out.println("strBuf1 charAt 2 :"+ strBuf1.charAt(2));

strBuf1.setCharAt(1, 't');

System.out.println("strBuf1 setCharAt 1 to t is:"+ strBuf1);

}}

import java.io.\*;

class stringfun4

{

public static void main(String[] args)

{

String str = "PUSOFTECH ";

//Create a object of StringBuffer class

StringBuffer strbuf = new StringBuffer("hello");

strbuf.append(str);

System.out.println(strbuf);

// delete string from the stringbuffer

strbuf.delete(0,str.length());

//append()

strbuf.append(" Hello");

strbuf.append(" World");

//print HelloWorld

System.out.println(strbuf);

//insert()

strbuf.insert(5,"\_Java ");

//print Hello\_Java World

System.out.println(strbuf);

//reverse()

strbuf.reverse();

System.out.print("Reversed string : "+ strbuf);

//print dlroW avaJ\_olleH

strbuf.reverse();

System.out.println("Back from reverse "+ strbuf);

//print Hello\_Java World

//setCharAt()

strbuf.setCharAt(5,' ');

System.out.println(strbuf);

//prit Hello Java World

//charAt()

System.out.print("Char at 6th position:"+strbuf.charAt(6));

//print J

//substring()

System.out.print("Substring from position 3 to 6 : ");

System.out.println(strbuf.substring(3,7));

//print lo J

//deleteCharAt()

strbuf.deleteCharAt(3);

System.out.println(strbuf);

//print Helo java World

//capacity()

System.out.print("Capacity of StringBuffer object : ");

System.out.println(strbuf.capacity());

//delete() and length()

strbuf.delete(6,strbuf.length());

System.out.println(strbuf);

}}

An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following −

An applet is a Java class that extends the java.applet.Applet class.

A main() method is not invoked on an applet, and an applet class will not define main().

Applets are designed to be embedded within an HTML page.

When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.

A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.

The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.

Applets have strict security rules that are enforced by the Web browser.

### Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

### java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

**public void init():** is used to initialized the Applet. It is invoked only once.

**public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.

**public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.

**public void destroy():** is used to destroy the Applet. It is invoked only once.

### java.awt.Component class

The Component class provides 1 life cycle method of applet.

**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.

Example

import java.applet.\*;

import java.awt.\*;

public class HelloWorldApplet extends Applet

{

public void paint (Graphics g) {

g.drawString ("Hello World", 25, 50);

-------

-------

}

}

**The Applet Class**

Every applet is an extension of the *java.applet.Applet class*. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context.

These include methods that do the following −

Get applet parameters

Get the network location of the HTML file that contains the applet

Get the network location of the applet class directory

Print a status message in the browser

Fetch an image

Fetch an audio clip

Play an audio clip

Resize the applet

Additionally, the Applet class provides an interface by which the viewer or browser obtains information about the applet and controls the applet's execution. The viewer may Perform Following Actions −

Request information about the author, version, and copyright of the applet

Request a description of the parameters the applet recognizes

Initialize the applet

Destroy the applet

Start the applet's execution

Stop the applet's execution

Invoking an Applet

An applet may be invoked by embedding directives in an HTML file and viewing the file through an applet viewer or Java-enabled browser.

The <applet> tag is the basis for embedding an applet in an HTML file. Following is an example that invokes the "Hello, World" applet −

<html>

-----

<applet code = "HelloWorldApplet.class" width = "320" height = "120">

</applet>

-----

</html>

If an applet takes parameters, values may be passed for the parameters by adding <param> tags between <applet> and </applet>. The browser ignores text and other tags between the applet tags.

<html>

<title>Checkerboard Applet</title>

<hr>

<applet code = "CheckerApplet.class" width = "480" height = "320">

<param name = "color" value = "blue">

<param name = "squaresize" value = "30">

</applet>

<hr>

</html>

**Note** − Parameter names are not case sensitive.

mport java.applet.\*;

import java.awt.\*;

public class CheckerApplet extends Applet

{

public void init ()

{

String colorParam = getParameter ("color");

Color fg = parseColor (colorParam);

setBackground (fg);

}

}

Commonly used methods of Graphics class:

**public abstract void drawString(String str, int x, int y):** is used to draw the specified string.

**public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.

**public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.

**public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.

**public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.

**public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).

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

**public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.

**public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.

**public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.

**public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

**import** java.applet.Applet;

**import** java.awt.\*;

**public** **class** GraphicsDemo **extends** Applet{

**public** **void** paint(Graphics g){

g.setColor(Color.red);

g.drawString("Welcome",50, 50);

g.drawLine(20,30,20,300);

g.drawRect(70,100,30,30);

g.fillRect(170,100,30,30);

g.drawOval(70,200,30,30);

g.setColor(Color.pink);

g.fillOval(170,200,30,30);

g.drawArc(90,150,30,30,30,270);

g.fillArc(270,150,30,30,0,180);

}

}

### myapplet.html

<html>

<body>

<applet code="GraphicsDemo.class" width="300" height="300">

</applet>

</body>

</html>

# Java AWT Tutorial

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

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

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

### Container

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

### Window

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

### Panel

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

### Frame

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

### Useful Methods of Component class

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

## AWT UI Elements:

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

|  |  |
| --- | --- |
| **Sr. No.** | **Control & Description** |
| 1 | [**Label**](https://www.tutorialspoint.com/awt/awt_label.htm)  A Label object is a component for placing text in a container. |
| 2 | [**Button**](https://www.tutorialspoint.com/awt/awt_button.htm)  This class creates a labeled button. |
| 3 | [**CheckBox**](https://www.tutorialspoint.com/awt/awt_checkbox.htm)  A check box is a graphical component that can be in either an **on** (true) or **off** (false) state. |
| 4 | [**CheckBoxGroup**](https://www.tutorialspoint.com/awt/awt_checkboxgroup.htm)  The CheckboxGroup class is used to group the set of checkbox. |
| 5 | [**List**](https://www.tutorialspoint.com/awt/awt_list.htm)  The List component presents the user with a scrolling list of text items. |
| 6 | [**Text Field**](https://www.tutorialspoint.com/awt/awt_textfield.htm)  A TextField object is a text component that allows for the editing of a single line of text. |
| 7 | [**Text Area**](https://www.tutorialspoint.com/awt/awt_textarea.htm)  A TextArea object is a text component that allows for the editing of a multiple lines of text. |
| 8 | [**Choice**](https://www.tutorialspoint.com/awt/awt_choice.htm)  A Choice control is used to show pop up menu of choices. Selected choice is shown on the top of the menu. |
| 9 | [**Canvas**](https://www.tutorialspoint.com/awt/awt_canvas.htm)  A Canvas control represents a rectangular area where application can draw something or can receive inputs created by user. |
| 10 | [**Image**](https://www.tutorialspoint.com/awt/awt_image.htm)  An Image control is superclass for all image classes representing graphical images. |
| 11 | [**Scroll Bar**](https://www.tutorialspoint.com/awt/awt_scrollbar.htm)  A Scrollbar control represents a scroll bar component in order to enable user to select from range of values. |
| 12 | [**Dialog**](https://www.tutorialspoint.com/awt/awt_dialog.htm)  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 | [**File Dialog**](https://www.tutorialspoint.com/awt/awt_filedialog.htm)  A FileDialog control represents a dialog window from which the user can select a file. |

**class** First **extends** Frame{

First(){

Button b=**new** Button("click me");

setLayout(**null**);//no layout manager

b.setBounds(30,100,80,30);// setting button position

add(b);//adding button into frame

}

**public** **static** **void** main(String args[]){

First f=**new** First();

f.setSize(300,300);//frame size 300 width and 300 height

f.setVisible(**true**);//now frame will be visible, by default not visible

}}

# 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. 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:

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 Swing

Java Foundation Classes (JFC) 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 in java

|  |  |
| --- | --- |
| **AWT** | **Swing** |
| AWT stands for Abstract Window Toolkit. | Swing is a part of Java Foundation Class (JFC). |
| AWT components are heavy weight. | Swing components are light weight. |
| AWT components are platform dependent so there look and feel changes according to OS. | Swing components are platform independent so there look and feel remains constant. |
| AWT components are not very good in look and feel as compared to Swing components.  Difference between AWT and Swing | Swing components are better in look and feel as compared to AWT.  Difference between AWT and Swing |
| AWT provides less components than Swing. | Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc. |
| 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.

### Event Handling

Any program that uses GUI (graphical user interface) such as Java application written for windows, is event driven. Event describes the change of state of any object. **Example :** Pressing a button, Entering a character in Textbox.

#### Components of Event Handling

Event handling has three main components,

**Events :** An event is a change of state of an object.

**Events Source :** Event source is an object that generates an event.

**Listeners :** A listener is an object that listens to the event. A listener gets notified when an event occurs.

#### How Events are handled ?

A source generates an Event and send it to one or more listeners registered with the source. Once event is received by the listener, they processe the event and then return. Events are supported by a number of Java packages, like **java.util**, **java.awt** and **java.awt.event**.

#### Important Event Classe and Interface

|  |  |  |
| --- | --- | --- |
| **Event Classe** | **Description** | **Listener Interface** |
| **ActionEvent** | generated when button is pressed, menu-item is selected, list-item is double clicked | ActionListener |
| **MouseEvent** | generated when mouse is dragged, moved,clicked,pressed or released also when the enters or exit a component | MouseListener |
| **KeyEvent** | generated when input is received from keyboard | KeyListener |
| **ItemEvent** | generated when check-box or list item is clicked | ItemListener |
| **TextEvent** | generated when value of textarea or textfield is changed | TextListener |
| **MouseWheelEvent** | generated when mouse wheel is moved | MouseWheelListener |
| **WindowEvent** | generated when window is activated, deactivated, deiconified, iconified, opened or closed | WindowListener |
| **ComponentEvent** | generated when component is hidden, moved, resized or set visible | ComponentEventListener |
| **ContainerEvent** | generated when component is added or removed from container | ContainerListener |
| **AdjustmentEvent** | generated when scroll bar is manipulated | AdjustmentListener |
| **FocusEvent** | generated when component gains or loses keyboard focus | FocusListener |

# AWT MouseEvent Class

This event indicates a mouse action occurred in a component. This low-level event is generated by a component object for Mouse Events and Mouse motion events.

a mouse button is pressed

a mouse button is released

a mouse button is clicked (pressed and released)

a mouse is moved

the mouse is dragged

## Class declaration

Following is the declaration for **java.awt.event.MouseEvent** class:

public class MouseEvent

extends InputEvent

## Class constructors

|  |  |
| --- | --- |
| **S.N.** | **Constructor & Description** |
| 1 | **MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int clickCount, boolean popupTrigger)**  Constructs a MouseEvent object with the specified source component, type, modifiers, coordinates, and click count. |
| 2 | **MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int clickCount, boolean popupTrigger, int button)**  Constructs a MouseEvent object with the specified source component, type, modifiers, coordinates, and click count. |
| 3 | **MouseEvent(Component source, int id, long when, int modifiers, int x, int y, int xAbs, int yAbs, int clickCount, boolean popupTrigger, int button)**  Constructs a MouseEvent object with the specified source component, type, modifiers, coordinates, absolute coordinates, and click count. |

## Class methods

|  |  |
| --- | --- |
| **S.N.** | **Method & Description** |
| 1 | **int getButton()**  Returns which, if any, of the mouse buttons has changed state. |
| 2 | **int getClickCount()**  Returns the number of mouse clicks associated with this event. |
| 3 | **Point getLocationOnScreen()**  Returns the absolute x, y position of the event. |
| 4 | **static String getMouseModifiersText(int modifiers)**  Returns a String describing the modifier keys and mouse buttons that were down during the event, such as "Shift", or "Ctrl+Shift". |
| 5 | **Point getPoint()**  Returns the x,y position of the event relative to the source component. |
| 6 | **int getX()**  Returns the horizontal x position of the event relative to the source component. |
| 7 | **int getXOnScreen()**  Returns the absolute horizontal x position of the event. |
| 8 | **int getY()**  Returns the vertical y position of the event relative to the source component. |
| 9 | **int getYOnScreen()**  Returns the absolute vertical y position of the event. |
| 10 | **boolean isPopupTrigger() Returns whether or not this mouse event is the popup menu trigger event for the platform.** |
| 11 | **String paramString()**  Returns a parameter string identifying this event. |
| 12 | **void translatePoint(int x, int y)**  Translates the event's coordinates to a new position by adding specified x (horizontal) and y (vertical) offsets. |

# AWT KeyEvent Class

On entering the character the Key event is generated.There are three types of key events which are represented by the integer constants. These key events are following

KEY\_PRESSED

KEY\_RELASED

KEY\_TYPED

## Class declaration

Following is the declaration for **java.awt.event.KeyEvent** class:

public class KeyEvent

extends InputEvent

## Field

Following are the fields for **java.awt.InputEvent** class:

**static char CHAR\_UNDEFINED**--KEY\_PRESSED and KEY\_RELEASED events which do not map to a valid Unicode character use this for the keyChar value.

**static int KEY\_FIRST**--The first number in the range of ids used for key events.

**static int KEY\_LAST**--The last number in the range of ids used for key events.

**static int KEY\_LOCATION\_LEFT**--A constant indicating that the key pressed or released is in the left key location (there is more than one possible location for this key).

**static int KEY\_LOCATION\_NUMPAD** --A constant indicating that the key event originated on the numeric keypad or with a virtual key corresponding to the numeric keypad.

**static int KEY\_LOCATION\_RIGHT**-- A constant indicating that the key pressed or released is in the right key location (there is more than one possible location for this key).

**static int KEY\_LOCATION\_STANDARD**--A constant indicating that the key pressed or released is not distinguished as the left or right version of a key, and did not originate on the numeric keypad (or did not originate with a virtual key corresponding to the numeric keypad).

**static int KEY\_LOCATION\_UNKNOWN** -- A constant indicating that the keyLocation is indeterminate or not relevant.

**static int KEY\_PRESSED** --The "key pressed" event.

**static int KEY\_RELEASED**--The "key released" event.

**static int KEY\_TYPED**--The "key typed" event.

static int VK\_0 --VK\_0 thru VK\_9 are the same as ASCII '0' thru '9' (0x30 - 0x39)

## Class constructors

|  |  |
| --- | --- |
| **S.N.** | **Constructor & Description** |
| 1 | **KeyEvent(Component source, int id, long when, int modifiers, int keyCode)**  Deprecated. as of JDK1.1 |
| 2 | **KeyEvent(Component source, int id, long when, int modifiers, int keyCode, char keyChar)**  Constructs a KeyEvent object. |
| 3 | **KeyEvent(Component source, int id, long when, int modifiers, int keyCode, char keyChar, int keyLocation)** |

## Class methods

|  |  |
| --- | --- |
| **S.N.** | **Method & Description** |
| 1 | **char getKeyChar()**  Returns the character associated with the key in this event. |
| 2 | **int getKeyCode()**  Returns the integer keyCode associated with the key in this event. |
| 3 | **int getKeyLocation()**  Returns the location of the key that originated this key event. |
| 4 | **static String getKeyModifiersText(int modifiers)**  Returns a String describing the modifier key(s), such as "Shift", or "Ctrl+Shift". |
| 5 | **static String getKeyText(int keyCode)**  Returns a String describing the keyCode, such as "HOME", "F1" or "A". |
| 6 | **boolean isActionKey()**  Returns whether the key in this event is an "action" key. |
| 7 | **String paramString()**  Returns a parameter string identifying this event. |
| 8 | **void setKeyChar(char keyChar)**  Set the keyChar value to indicate a logical character. |
| 9 | **void setKeyCode(int keyCode)**  Set the keyCode value to indicate a physical key. |
| 10 | **void setModifiers(int modifiers)**  Deprecated. as of JDK1.1.4 |

## Methods inherited

This class inherits methods from the following classes:

java.awt.event.InputEvent

java.awt.event.ComponentEvent

java.awt.AWTEvent

java.util.EventObject

java.lang.Object

import java.awt.\*;

import java.awt.event.\*;

import java.applet.\*;

import java.applet.\*;

import java.awt.event.\*;

import java.awt.\*;

public class Test extends Applet implements KeyListener

{

String msg="";

public void init()

{

addKeyListener(this);

}

public void keyPressed(KeyEvent k)

{

showStatus("KeyPressed");

}

public void keyReleased(KeyEvent k)

{

showStatus("KeyRealesed");

}

public void keyTyped(KeyEvent k)

{

msg = msg+k.getKeyChar();

repaint();

}

public void paint(Graphics g)

{

g.drawString(msg, 20, 40);

}

}

==

## AWT Event Classes:

Following is the list of commonly used event classes.

|  |  |
| --- | --- |
| **Sr. No.** | **Control & Description** |
| 1 | [**AWTEvent**](https://www.tutorialspoint.com/awt/awt_awt_event.htm)  It is the root event class for all AWT events. This class and its subclasses supercede the original java.awt.Event class. |
| 2 | [**ActionEvent**](https://www.tutorialspoint.com/awt/awt_action_event.htm)  The ActionEvent is generated when button is clicked or the item of a list is double clicked. |
| 3 | [**InputEvent**](https://www.tutorialspoint.com/awt/awt_input_event.htm)  The InputEvent class is root event class for all component-level input events. |
| 4 | [**KeyEvent**](https://www.tutorialspoint.com/awt/awt_key_event.htm)  On entering the character the Key event is generated. |
| 5 | [**MouseEvent**](https://www.tutorialspoint.com/awt/awt_mouse_event.htm)  This event indicates a mouse action occurred in a component. |
| 6 | [**TextEvent**](https://www.tutorialspoint.com/awt/awt_text_event.htm)  The object of this class represents the text events. |
| 7 | [**WindowEvent**](https://www.tutorialspoint.com/awt/awt_window_event.htm)  The object of this class represents the change in state of a window. |
| 8 | [**AdjustmentEvent**](https://www.tutorialspoint.com/awt/awt_adjustment_event.htm)  The object of this class represents the adjustment event emitted by Adjustable objects. |
| 9 | [**ComponentEvent**](https://www.tutorialspoint.com/awt/awt_component_event.htm)  The object of this class represents the change in state of a window. |
| 10 | [**ContainerEvent**](https://www.tutorialspoint.com/awt/awt_container_event.htm)  The object of this class represents the change in state of a window. |
| 11 | [**MouseMotionEvent**](https://www.tutorialspoint.com/awt/awt_mousemotion_event.htm)  The object of this class represents the change in state of a window. |
| 12 | [**PaintEvent**](https://www.tutorialspoint.com/awt/awt_paint_event.htm)  The object of this class represents the change in state of a window. |

# AWT Layouts

## Introduction

Layout means the arrangement of components within the container. In other way we can say that placing the components at a particular position within the container. The task of layouting the controls is done automatically by the Layout Manager.

## Layout Manager

The layout manager automatically positions all the components within the container. If we do not use layout manager then also the components are positioned by the default layout manager. It is possible to layout the controls by hand but it becomes very difficult because of the following two reasons.

It is very tedious to handle a large number of controls within the container.

Oftenly the width and height information of a component is not given when we need to arrange them.

Java provide us with various layout manager to position the controls. The properties like size,shape and arrangement varies from one layout manager to other layout manager. When the size of the applet or the application window changes the size, shape and arrangement of the components also changes in response i.e. the layout managers adapt to the dimensions of appletviewer or the application window.

The layout manager is associated with every Container object. Each layout manager is an object of the class that implements the LayoutManager interface.

Following are the interfaces defining functionalities of Layout Managers.

|  |  |
| --- | --- |
| **Sr. No.** | **Interface & Description** |
| 1 | [**LayoutManager**](https://www.tutorialspoint.com/awt/awt_layoutmanager.htm)  The LayoutManager interface declares those methods which need to be implemented by the class whose object will act as a layout manager. |
| 2 | [**LayoutManager2**](https://www.tutorialspoint.com/awt/awt_layoutmanager2.htm)  The LayoutManager2 is the sub-interface of the LayoutManager.This interface is for those classes that know how to layout containers based on layout constraint object. |

## AWT Layout Manager Classes:

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

|  |  |
| --- | --- |
| **Sr. No.** | **LayoutManager & Description** |
| 1 | [**BorderLayout**](https://www.tutorialspoint.com/awt/awt_borderlayout.htm)  The borderlayout arranges the components to fit in the five regions: east, west, north, south and center. |
| 2 | [**CardLayout**](https://www.tutorialspoint.com/awt/awt_cardlayout.htm)  The CardLayout object treats each component in the container as a card. Only one card is visible at a time. |
| 3 | [**FlowLayout**](https://www.tutorialspoint.com/awt/awt_flowlayout.htm)  The FlowLayout is the default layout.It layouts the components in a directional flow. |
| 4 | [**GridLayout**](https://www.tutorialspoint.com/awt/awt_gridlayout.htm)  The GridLayout manages the components in form of a rectangular grid. |
| 5 | [**GridBagLayout**](https://www.tutorialspoint.com/awt/awt_gridbaglayout.htm)  This is the most flexible layout manager class.The object of GridBagLayout aligns the component vertically,horizontally or along their baseline without requiring the components of same size. |