What is Java

Java is a **programming language** and a **platform**.

Java is a high level, robust, secured and object-oriented programming language.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform.

Java Example

Let's have a quick look at java programming example. A detailed description of hello java example is given in next page.

```java
class Simple{
    public static void main(String args[]){
     System.out.println("Hello Java");
    }
}
```

Where it is used?

According to Sun, 3 billion devices run java. There are many devices where java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus etc.
2. Web Applications such as irctc.co.in, javatpoint.com etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games etc.

Types of Java Applications

There are mainly 4 type of applications that can be created using java programming:

1) Standalone Application

It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.

4) Mobile Application

An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.

History of Java

Java history is interesting to know. The history of java starts from Green Team. Java team members (also known as **Green Team**), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.

For the green team members, it was an advance concept at that time. But, it was suited for internet programming. Later, Java technology as incorporated by Netscape.

James Gosling

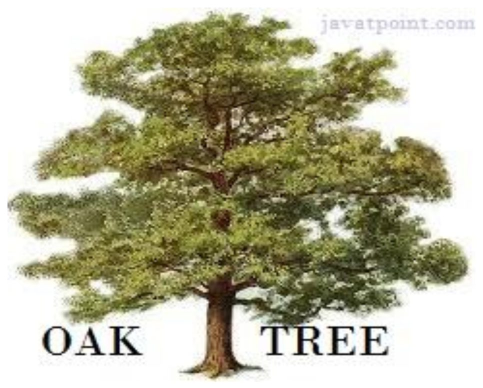
Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. There are given the major points that describes the history of java.

1) **James Gosling**, **Mike Sheridan**, and **Patrick Naughton**initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

2) Originally designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called **"Greentalk"** by James Gosling and file extension was .gt.

4) After that, it was called **Oak** and was developed as a part of the Green project.



Why Oak name for java language?

5) **Why Oak?** Oak is a symbol of strength and choosen as a national tree of many countries like U.S.A., France, Germany, Romania etc.

6) In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

Why Java name for java language?

7) **Why they choosed java name for java language?** The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA" etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

According to James Gosling "Java was one of the top choices along with **Silk**". Since java was so unique, most of the team members preferred java.

8) Java is an island of Indonesia where first coffee was produced (called java coffee).

9) Notice that Java is just a name not an acronym.

10) Originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

11) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.

12) JDK 1.0 released in(January 23, 1996).

Java Version History

There are many java versions that has been released. Current stable release of Java is Java SE 8.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan, 1996)
3. JDK 1.1 (19th Feb, 1997)
4. J2SE 1.2 (8th Dec, 1998)
5. J2SE 1.3 (8th May, 2000)
6. J2SE 1.4 (6th Feb, 2002)
7. J2SE 5.0 (30th Sep, 2004)
8. Java SE 6 (11th Dec, 2006)
9. Java SE 7 (28th July, 2011)
10. Java SE 8 (18th March, 2014)

Features of Java

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral

7. Portable
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

Simple

According to Sun, Java language is simple because:

syntax is based on C++ (so easier for programmers to learn it after C++).

removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.

No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

Object-oriented

Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.

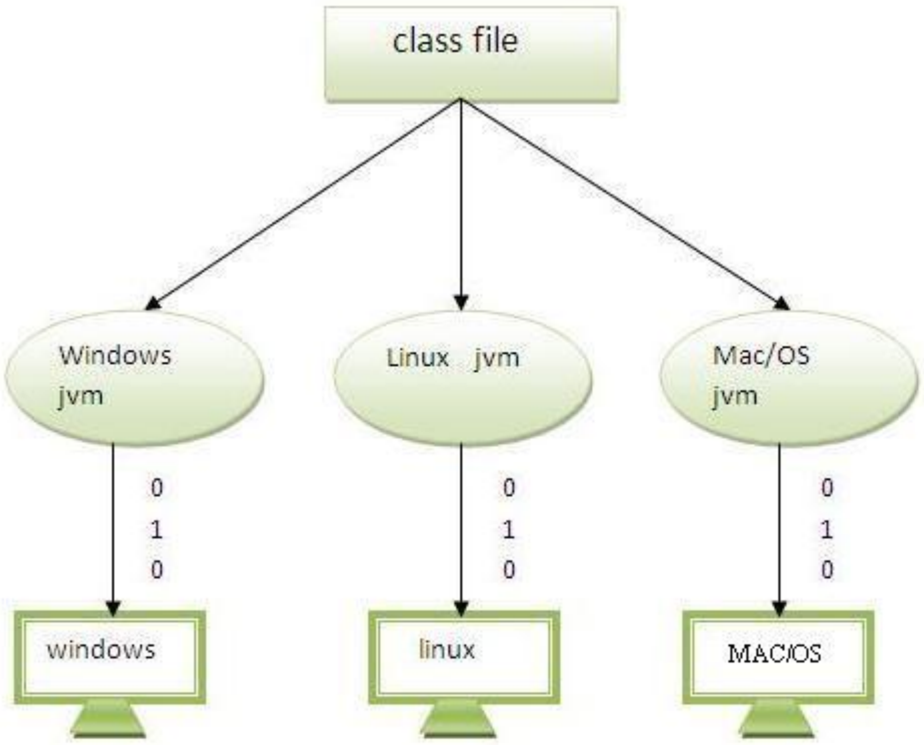
Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Platform Independent

A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms.It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

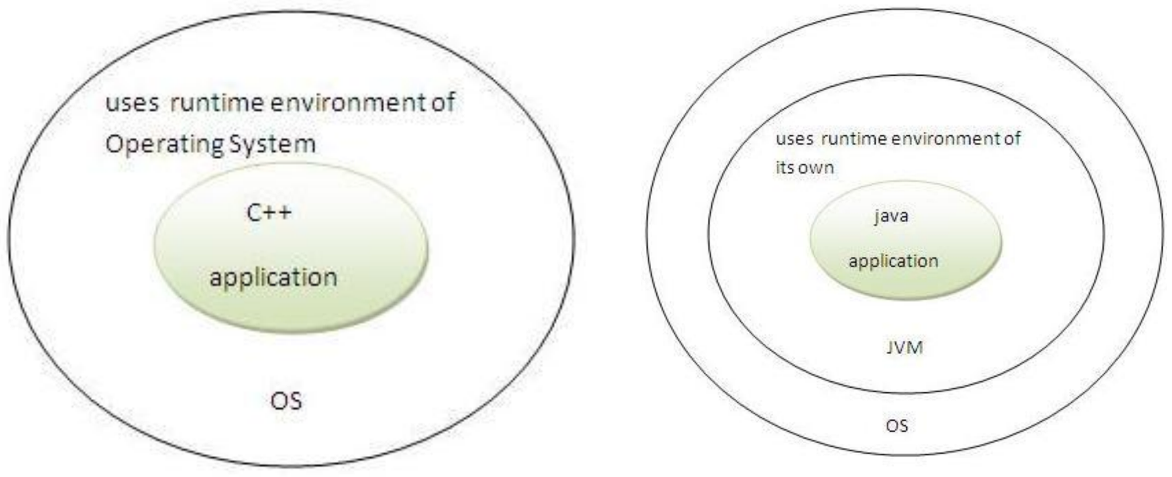


Java code can be run on multiple platforms e.g.Windows,Linux,Sun Solaris,Mac/OS etc. Java code is compiled by the compiler and converted into bytecode.This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

Secured

Java is secured because:

- No explicit pointer
- Programs run inside virtual machine sandbox.


uses runtime environment of Operating System
C++
application
OS
uses runtime environment of its own
java
application
JVM
OS

- **Classloader-** adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier-** checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager-** determines what resources a class can access such as reading and writing to the local disk.

These security are provided by java language. Some security can also be provided by application developer through SSL,JAAS,cryptography etc.

Robust

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

Architecture-neutral

There is no implementation dependent features e.g. size of primitive types is set.

Portable

We may carry the java bytecode to any platform.

High-performance

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

Distributed

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

What are the main differences between Java and C++?

Both Java and C++ support object oriented programming, yet there are differences between them. To begin with, Java is a pure object oriented programming language; therefore, everything is an object in Java (single root hierarchy as everything gets derived fromjava.lang.Object). On the contrary, in C++ there is no such root hierarchy. C++ supports both procedural and object oriented programming; therefore, it is called a hybrid language.

Following are the major differences between Java and C++:

Java	**C++**
Java does not support pointers, templates, unions, operator overloading, structures etc. The Java language promoters initially said "No pointers!", but when many programmers questioned how you can work without pointers, the promoters began saying "Restricted pointers." Java supports what it calls "references". References act a lot like pointers in C/C++ languages but you cannot perform arithmetic on pointers in Java. References have types, and they're type-safe. These references cannot be interpreted as raw address and unsafe conversion is not allowed.	C++ supports structures, unions, templates, operator overloading, pointers and pointer arithmetic.
Java support automatic garbage collection. It does not support destructors as C++ does.	C++ support destructors, which is automatically invoked when the object is destroyed.
Java does not support conditional compilation and inclusion.	Conditional inclusion (#ifdef #ifndef type) is one of the main features of C/C++.
Java has built in support for threads. In Java, there is a Thread class that you inherit to create a new thread	C++ has no built in support for threads. C++ relies on non-standard

and override the run() method.	third-party libraries for thread support.
Java does not support default arguments. There is no scope resolution operator (::) in Java. The method definitions must always occur within a class, so there is no need for scope resolution there either.	C++ supports default arguments. C++ has scope resolution operator (::) which is used to to define a method outside a class and to access a global variable within from the scope where a local variable also exists with the same name.
There is no *goto* statement in Java. The keywords const and goto are reserved, even though they are not used.	C++ has *goto* statement. However, it is not considered good practice to use of*goto* statement.
Java doesn't provide multiple inheritance, at least not in the same sense that C++ does.	C++ does support multiple inheritance. The keyword virtual is used to resolve ambiguities during multiple inheritance if there is any.
Exception handling in Java is different because there are no destructors. Also, in Java, try/catch must be defined if the function declares that it may throw an exception.	While in C++, you may not include the try/catch even if the function throws an exception.
Java has method overloading, but no operator overloading. The String class does use the + and += operators to concatenate strings and Stringexpressions use automatic type conversion, but that's a special built-in case.	C++ supports both method overloading and operator overloading.
Java has built-in support for documentation comments (/** ... */); therefore, Java source files can contain their own documentation, which is read	C++ does not support documentation comments.

by a separate tool usuallyjavadoc and reformatted into HTML. This helps keeping documentation maintained in easy way.	
Java is interpreted for the most part and hence platform independent.	C++ generates object code and the same code may not run on different platforms.

There are many other subtle differences and similarities in Java and C++. Here, we included the major ones. To read more see the references section at the end.

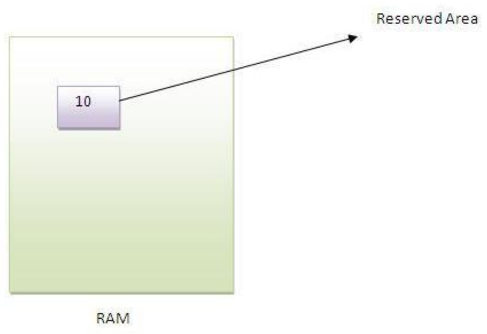
Hope you have enjoyed reading differences between Java and C++. Please do write us if you have any suggestion/comment or come across any error on this page. Thanks for reading!

Variable and Datatype in Java

In this page, we will learn about the variable and java data types. Variable is a name of memory location. There are three types of variables: local, instance and static. There are two types of datatypes in java, primitive and non-primitive.

Variable

Variable is name of reserved area allocated in memory.

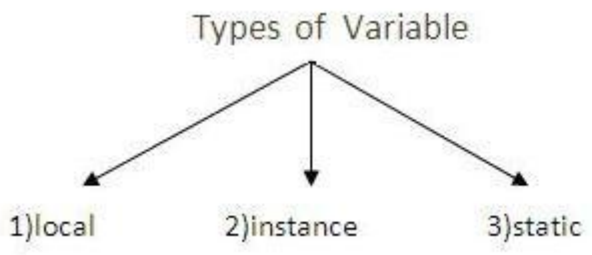


```
1. int data=50;//Here data is variable
```

Types of Variable

There are three types of variables in java

- local variable
- instance variable
- static variable



Local Variable

A variable that is declared inside the method is called local variable.

Instance Variable

A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static.

Static variable

A variable that is declared as static is called static variable. It cannot be local.

We will have detailed learning of these variables in next chapters.

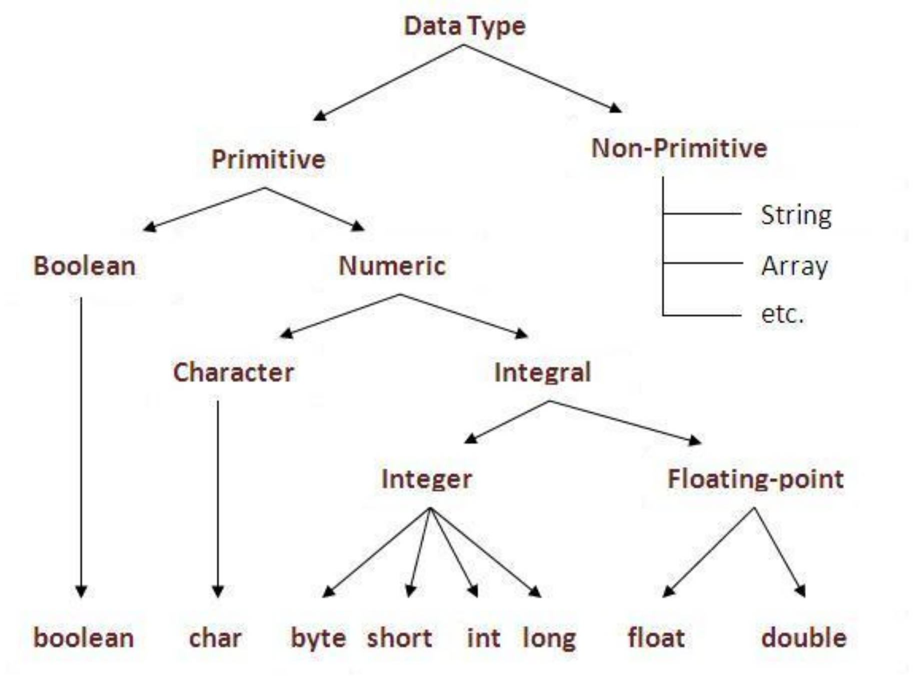
Example to understand the types of variables

```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

Data Types in Java

In java, there are two types of data types

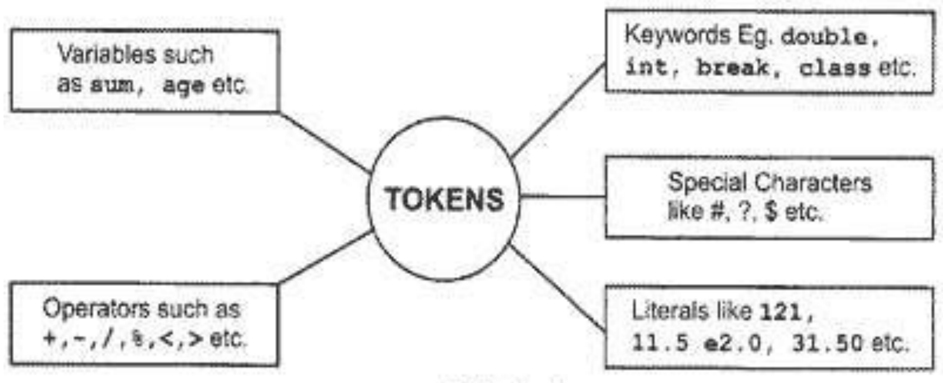
- primitive data types
- non-primitive data types



Data Type	**Default Value**	**Default size**
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Java Tokens:- A java Program is made up of Classes and Methods and in the Methods are the Container of the various Statements And a Statement is made up of Variables, Constants, operators etc .

Tokens are the various Java program elements which are identified by the compiler. A token is the smallest element of a program that is meaningful to the compiler. Tokens supported in Java include keywords, variables, constants, special characters, operations etc.


Variables such as sum, age etc.
Keywords Eg. double, int, break, class etc.
TOKENS
Special Characters like #, ?, $ etc.
Operators such as +,-,/,%,<,> etc.
Literals like 121, 11.5 e2.0, 31.50 etc.

Tokens in Java

When you compile a program, the compiler scans the text in your source code and extracts individual tokens. While tokenizing the source file, the compiler recognizes and subsequently removes whitespaces (spaces, tabs, newline and form feeds) and the text enclosed within comments. Now let us consider a program

```
//Print Hello
Public class Hello
{
Public static void main(String args[])
{
System.out.println("Hello Java");
}
}
```

The source code contains tokens such as public, class, Hello, {, public, static, void, main, (, String, [], args, {, System, out, println, (, "Hello Java", }, }. The resulting tokens· are compiled into Java bytecodes that is capable of being run from within an interpreted java environment. Token are useful for compiler to detect errors. When tokens are not arranged in a particular sequence, the compiler generates an error message.

Tokens are the smallest unit of Program There is Five Types of Tokens

1) Reserve Word or Keywords
2) Identifier
3) Literals
4) Operators
5) Separators

Operators in java

Operator in java is a symbol that is used to perform operations. There are many types of operators in java such as unary operator, arithmetic operator, relational operator, shift operator, bitwise operator, ternary operator and assignment operator.

Operators	**Precedence**
postfix	`expr++ expr--`
unary	`++expr --expr +expr -expr ~ !`
multiplicative	`* / %`
additive	`+ -`
shift	`<< >> >>>`
relational	`< > <= >= instanceof`
equality	`== !=`
bitwise AND	`&`
bitwise exclusive OR	`^`
bitwise inclusive OR	`|`
logical AND	`&&`
logical OR	`||`
ternary	`? :`
assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

Simple Program of Java

In this page, we will learn how to write the simple program of java. We can write a simple hello java program easily after installing the JDK.

To create a simple java program, you need to create a class that contains main method. Let's understand the requirement first.

Requirement for Hello Java Example

For executing any java program, you need to

- install the JDK if you don't have installed it, download the JDK and install it.
- set path of the jdk/bin directory. http://www.javatpoint.com/how-to-set-path-in-java
- create the java program
- compile and run the java program

Creating hello java example

Let's create the hello java program:

```java
class Simple{
    public static void main(String args[]){
     System.out.println("Hello Java");
    }
}
```

Test it Now

save this file as Simple.java

To compile: javac Simple.java

To execute: java Simple

```
Output:Hello Java
```

JAVA Branching statements

Branching statements

The branching statements are also called decision making statement which is used to see whether a particular condition has occurred or not and then tell the compiler to execute certain statements accordingly.

C language supports the following branching statements:

1. if statement
2. switch statement
3. conditional statement

JAVA- if statement

if statement: The if statement is use to control the flow of the execution of statement. If the expression is true then it transfers the control to the particular statement.

There are different forms of if statements, these are as followings:

A. simple if statement:

Syntax:

```
if(expression)

{                    //statement block

Statement-A;

}

Statement-B;
```

There may b more than one statement in a block. If the expression is true then the statement-A will be execute, otherwise it skips the statement and will jump to the statement-B. if statement-A will be executed after that statement-B will executed in a sequence.

}

B. If-else statement:

Syntax:

```
if(expression)

{

True-block

}

else

{

False-block

}
```

If the condition is true at first then the true block will be executed else false block will be executed.

C. Nesting of if-else statements:

Syntax:

```
if(condition-1)

{

If(condition-2)

{

Statement-1;

}
```

else

{

Statement-2;

}

}

else

{

Statement-3;

}

In this if the condition-1 is true then it will check the condition-2 and if it true then statement-1

will be executed else it will executes the statement-2. And if the condition-1 is fail then the control will goes to the else part and will executes the statement-3.

JAVA- Switch statement

Switch statement:

The problem in if-statement is the complexity to read and follow. Even it may be confuse the programmer who designed it. The switch statement tests the value of the variable with a list of case values and when a match is found, and then the block of that particular case will be executed.

Syntax:

switch(expression)

```
{

case: 1

block-1

break;

case: 2

block-2

break;

default:

default-block

break;

}
```

As is the above syntax we will give the expression in the form of variable. And according to that given value it will executes the case. If any of the case is not matched then the default case will be executed. And we have used the break statement which will help to exit from the switch statement. The break will skip the rest of the cases for comparisons, and this saves execution time.

JAVA- Conditional operator statement

Conditional operator statement:

The conditional operator statement is used with the combination of ? and : and this is known as conditional operator.

Syntax: conditional expression ? expression1 : expression2

The conditional expression is evaluated first and if the result is nonzero then the expression1 is returned as the value of the expression, otherwise the value of expression2 is returned.

For example:

```
If(a<1)

t=0;

else

t=1;
```

Here the above code can be written as:

```
t = (a<1) ? 0:1;
```

Object and Class in Java

. In object-oriented programming technique, we design a program using objects and classes.

Object is the physical as well as logical entity whereas class is the logical entity only.

Object in Java

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tengible and intengible). The example of integible object is banking system.

An object has three characteristics:

- **state:** represents data (value) of an object.
- **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
- **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But,it is used internally by the JVM to identify each object uniquely.

For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior.

Object is an instance of a class. Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.

Class in Java

A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.

A class in java can contain:

- data member
- method
- constructor
- block
- class and interface

Syntax to declare a class:

```
class <class_name>{
    data member;
    method;
}
```

Simple Example of Object and Class

In this example, we have created a Student class that have two data members id and name. We are creating the object of the Student class by new keyword and printing the objects value.

```
class Student1{
 int id;//data member (also instance variable)
 String name;//data member(also instance variable)

 public static void main(String args[]){
  Student1 s1=new Student1();//creating an object of Student
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

Test it Now

```
Output:0 null
```

Instance variable in Java

A variable that is created inside the class but outside the method, is known as instance variable.Instance variable doesn't get memory at compile time.It gets memory at runtime when object(instance) is created.That is why, it is known as instance variable.

Method in Java

In java, a method is like function i.e. used to expose behaviour of an object.

Advantage of Method

- Code Reusability
- Code Optimization

. Constructor in Java

Constructor in java is a *special type of method* that is used to initialize the object.

Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating java constructor

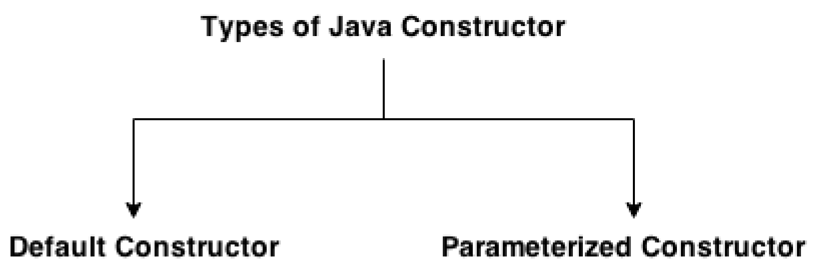
There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors

There are two types of constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



Java Default Constructor

A constructor that have no parameter is known as default constructor.

Syntax of default constructor:

```
<class_name>(){}
```

Example of default constructor

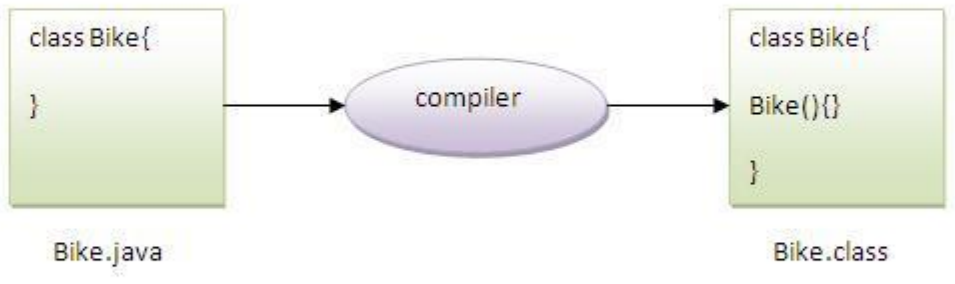
In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```java
class Bike1{
Bike1(){System.out.println("Bike is created");}
public static void main(String args[]){
Bike1 b=new Bike1();
}
}
```

Test it Now

Output:

```
Bike is created
```



Q) What is the purpose of default constructor?

Default constructor provides the default values to the object like 0, null etc. depending on the type.

Example of default constructor that displays the default values

```
class Student3{
int id;
String name;

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
Student3 s1=new Student3();
Student3 s2=new Student3();
s1.display();
s2.display();
}
}
```

Test it Now

Output:

```
0 null
0 null
```

Explanation: In the above class,you are not creating any constructor so compiler provides you a default constructor.Here 0 and null values are provided by default constructor.

Java parameterized constructor

A constructor that have parameters is known as parameterized constructor.

Why use parameterized constructor?

Parameterized constructor is used to provide different values to the distinct objects.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
class Student4{
    int id;
    String name;

    Student4(int i,String n){
    id = i;
    name = n;
    }
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    s1.display();
    s2.display();
  }
}
```

Test it Now

Output:

```
111 Karan
222 Aryan
```

The Class Destructor:

A **destructor** is a special member function of a class that is executed whenever an object of it's class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.

A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc.

Following example explains the concept of destructor:

```
import std.stdio;


class Line
{
   public:
      this()
      {
         writeln("Object is being created");
      }
      ~this()
      {
         writeln("Object is being deleted");
      }
      void setLength( double len )
      {
         length = len;
      }
      double getLength()
      {
         return length;
      }
   private:
      double length;
}
// Main function for the program
void main( )
{
   Line line = new Line();
   // set line length
   line.setLength(6.0);
   writeln("Length of line : ", line.getLength());
}
```

When the above code is compiled and executed, it produces the following result:

```
Object is being created
Length of line : 6
Object is being deleted
```

Inheritance in Java

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

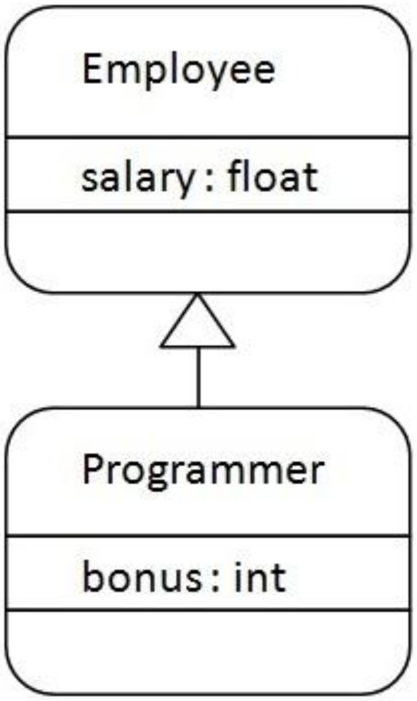
Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class.

In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.

Understanding the simple example of inheritance



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. Relationship between two classes is **Programmer IS-A Employee.**It means that Programmer is a type of Employee.

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

Test it Now

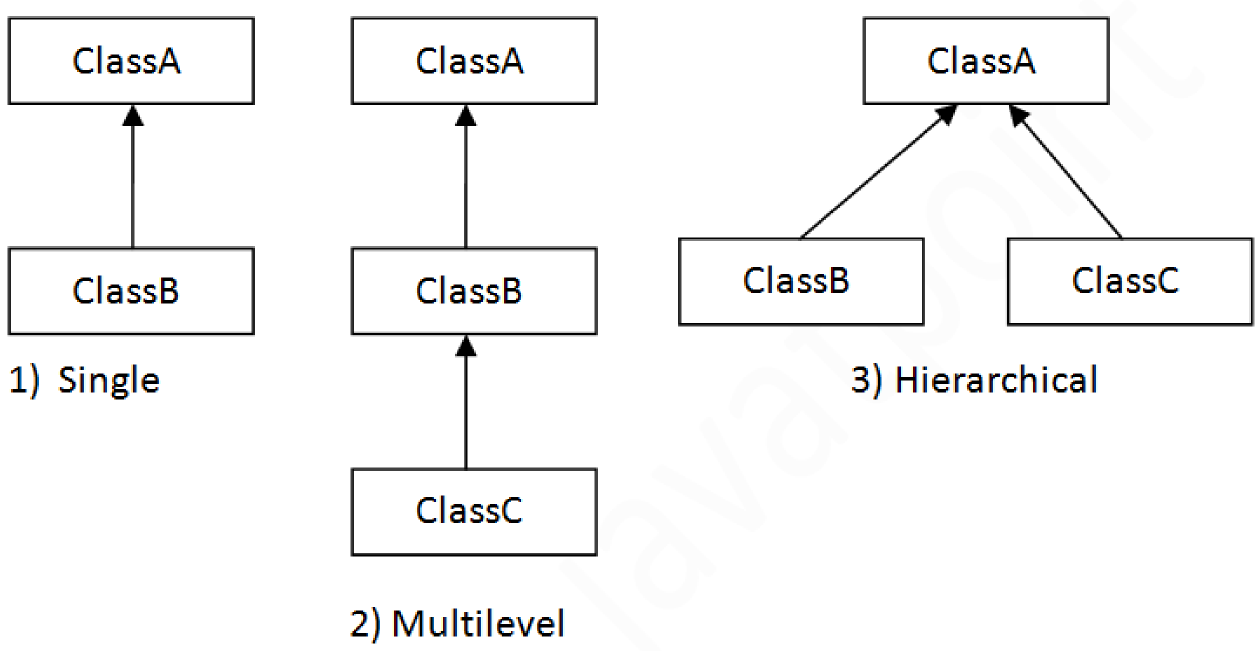
```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

Types of inheritance in java

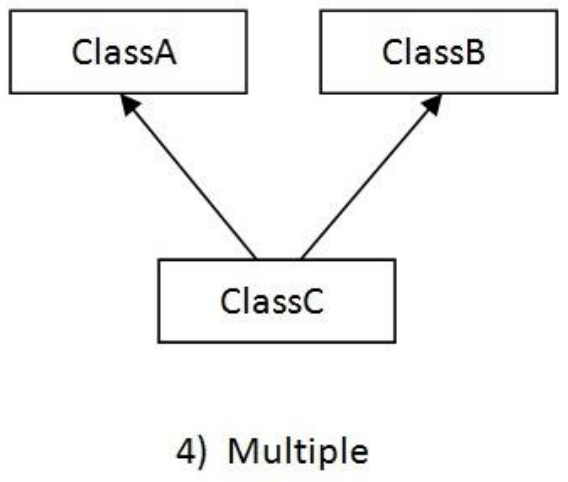
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

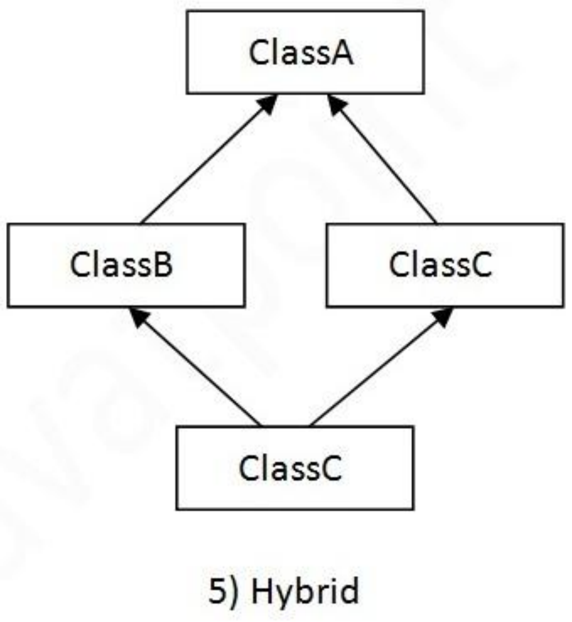
In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



Note: Multiple inheritance is not supported in java through class.

When a class extends multiple classes i.e. known as multiple inheritance. For Example:





.

Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

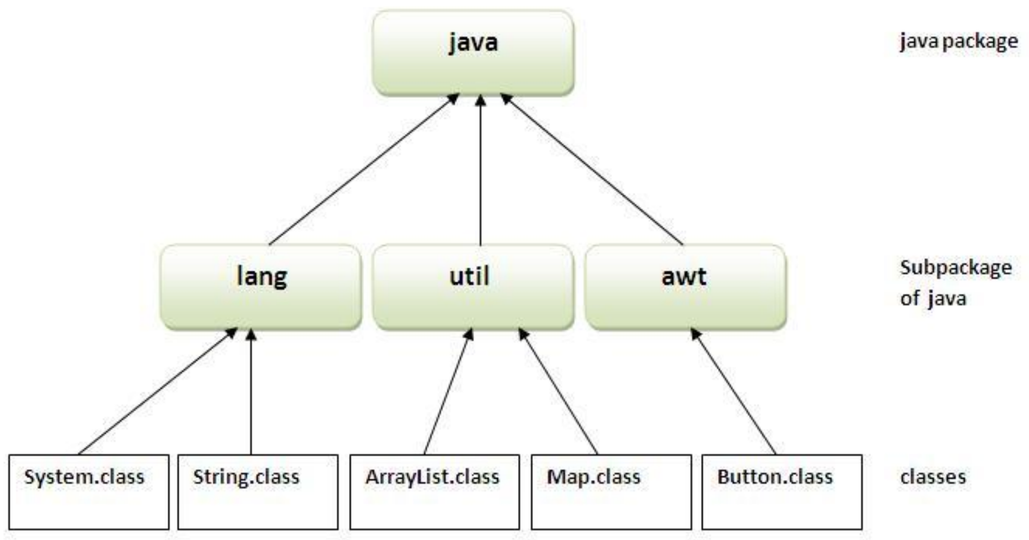
Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
   }
}
```

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

```
javac -d directory javafilename
```

For **example**

```
javac -d . Simple.java
```

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

How to run java package program

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

To Compile: javac -d . Simple.java

To Run: java mypack.Simple

```
Output:Welcome to package
```

The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder.

How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

1) Using packagename.*

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

```
Output:Hello
```

---

2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//save by A.java

package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.A;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

```
Output:Hello
```

---

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

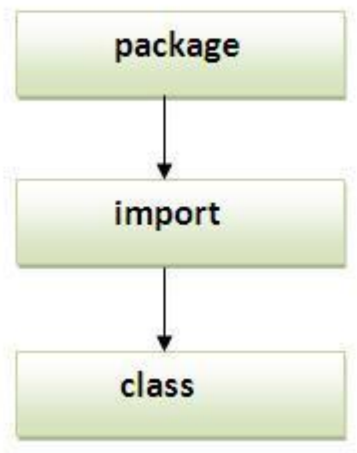
```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();
  }
}
```

```
Output:Hello
```

If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the subpackage as well.

Note: Sequence of the program must be package then import then class.



Subpackage in java

Package inside the package is called the **subpackage**. It should be created **to categorize the package further**.

Let's take an example, Sun Microsystem has definded a package named java that contains many classes like System, String, Reader, Writer, Socket etc. These classes represent a particular group e.g. Reader and Writer classes are for Input/Output operation, Socket and ServerSocket classes are for networking etc and so on. So, Sun has subcategorized the java package into subpackages such as lang, net, io etc. and put the Input/Output related classes in io package, Server and ServerSocket classes in net packages and so on.

Example of Subpackage

```java
package com.javatpoint.core;
class Simple{
  public static void main(String args[]){
   System.out.println("Hello subpackage");
  }
}
```

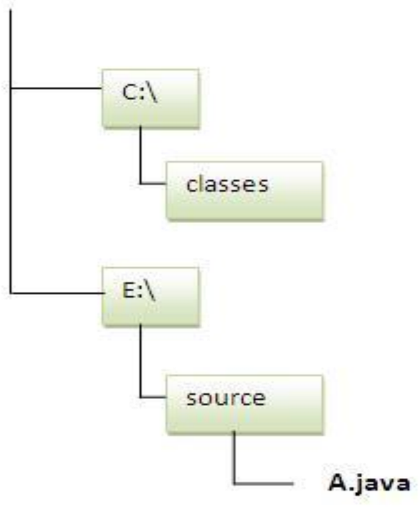
To Compile: javac -d . Simple.java

To Run: java com.javatpoint.core.Simple

```
Output:Hello subpackage
```

How to send the class file to another directory or drive?

There is a scenario, I want to put the class file of A.java source file in classes folder of c: drive. For example:



```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
   }
}
```

To Compile:

e:\sources> javac -d c:\classes Simple.java

To Run:

To run this program from e:\source directory, you need to set classpath of the directory where the class file resides.

e:\sources> set classpath=c:\classes;.;

e:\sources> java mypack.Simple

Another way to run this program by -classpath switch of java:

The -classpath switch can be used with javac and java tool.

To run this program from e:\source directory, you can use -classpath switch of java that tells where to look for class file. For example:

e:\sources> java -classpath c:\classes mypack.Simple

```
Output:Welcome to package
```

Ways to load the class files or jar files

There are two ways to load the class files temporary and permanent.

- Temporary
  - By setting the classpath in the command prompt
  - By -classpath switch
- Permanent
  - By setting the classpath in the environment variables
  - By creating the jar file, that contains all the class files, and copying the jar file in the jre/lib/ext folder.

Rule: There can be only one public class in a java source file and it must be saved by the public class name.

```java
//save as C.java otherwise Compilte Time Error

class A{}
class B{}
public class C{}
```

How to put two public classes in a package?

If you want to put two public classes in a package, have two java source files containing one public class, but keep the package name same. For example:

```java
//save as A.java

package javatpoint;
public class A{}
```

```java
//save as B.java

package javatpoint;
public class B{}
```

Java System Packages

Java System Packages / Java API Packages are collection of classes, interfaces and enumerated types. Different Java System Packages / Java API Packages are :

- **java.lang Package –** This java.lang package is a Java system package / Java API package that contains core classes related to language functionality and runtime system.
- **java.util Package –** The java.util package is a Java system package / Java API package that includes collection of data structure related classes.
- **java.io Package –** This java.io package is a Java system package / Java API package that offers the input / output operations and other file operations.
- **java.math Package** – This java.math package is a Java system package / Java API package that performs multi-precision arithmetic in your java program.
- **java.nio Package** – The word nio in this java.nio package refers to “new i/o”. The java.nio package is a Java system package / Java API package that contains classes and interfaces related to new input / output framework.
- **java.net Package –** The java.net package is a Java system package / Java API package that is imported to perform network operations, socket programming, DNS lookup and much more.
- **java.security Package –**This java.security package is a Java system package / Java API package that performs security related activities like encryption, decryption and key generation.
- **java.sql Package –** This java.sql package is a Java system package / Java API package that is imported to establish connectivity between Java and SQL through Java Database Connectivity (JDBC).

- **java.awt Package –** This java.awt package is a Java system package / Java API package that is used to create user interfaces and perform graphics programming on it.
- **java.applet package –** This java.applet package is a Java system package / Java API package that is used to create applets.

What is Thread in java

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.

Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.

As shown in the above figure, a thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS, and one process can have multiple threads.

How to create thread

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Commonly used methods of Thread class:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(depricated).
16. **public void resume():** is used to resume the suspended thread(depricated).
17. **public void stop():** is used to stop the thread(depricated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

public void run(): is used to perform action for a thread.

Starting a thread:

start() method of Thread class is used to start a newly created thread. It performs following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

---

1) Java Thread Example by extending Thread class

```
class Multi extends Thread{
public void run(){
```

```java
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

```
Output:thread is running...
```

2) Java Thread Example by implementing Runnable interface

```java
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 }
}
```

```
Output:thread is running...
```

If you are not extending the Thread class,your class object would not be treated as a thread object.So you need to explicitely create Thread class object.We are passing the object of your class that implements Runnable so that your class run() method may execute.

Multithreading in Java

1. Multithreading
2. Multitasking
3. Process-based multitasking
4. Thread-based multitasking
5. What is Thread

Multithreading in java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

- Process-based Multitasking (Multiprocessing)
- Thread-based Multitasking (Multithreading)

1) Process-based Multitasking (Multiprocessing)

- Each process has an address in memory. In other words, each process allocates a separate memory area.
- A process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

2) Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- A thread is lightweight.
- Cost of communication between the thread is low.

Life cycle of a Thread (Thread States)

1. Life cycle of a thread
   1. New
   2. Runnable
   3. Running
   4. Non-Runnable (Blocked)
   5. Terminated

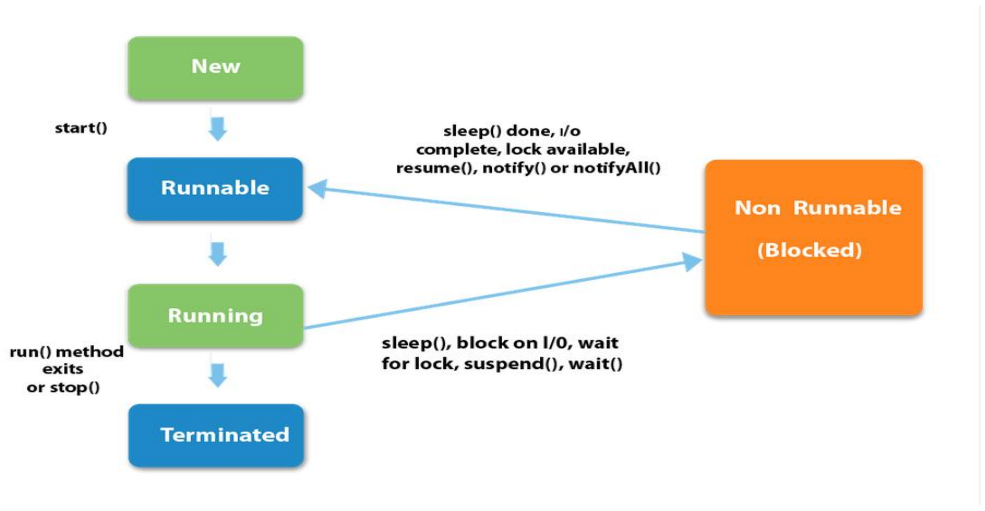
A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. New
2. Runnable

3. Running
4. Non-Runnable (Blocked)
5. Terminated


New
start()
Runnable
sleep() done, I/o complete, lock available, resume(), notify() or notifyAll()
Non Runnable (Blocked)
Running
sleep(), block on I/0, wait for lock, suspend(), wait()
run() method exits or stop()
Terminated

1) New

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

2) Runnable

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3) Running

The thread is in running state if the thread scheduler has selected it.

4) Non-Runnable (Blocked)

This is the state when the thread is still alive, but is currently not eligible to run.

5) Terminated

A thread is in terminated or dead state when its run() method exits.

Priority of a Thread (Thread Priority):

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defiend in Thread class:

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example of priority of a Thread:

```
class TestMultiPriority1 extends Thread{
 public void run(){
   System.out.println("running thread name is:"+Thread.currentThread().getName());
   System.out.println("running thread priority is:"+Thread.currentThread().getPriority());

  }
 public static void main(String args[]){
  TestMultiPriority1 m1=new TestMultiPriority1();
  TestMultiPriority1 m2=new TestMultiPriority1();
  m1.setPriority(Thread.MIN_PRIORITY);
  m2.setPriority(Thread.MAX_PRIORITY);
  m1.start();
  m2.start();

 }
}
```

```
Output:running thread name is:Thread-0
       running thread priority is:10
       running thread name is:Thread-1
       running thread priority is:1
```

Java Applet

Java is a programming language.
Developed in the years 1991 to 1994 by Sun Microsystems.

Programs written in Java are called applets.
The first browser that could show applets was introduced in 1994, as "WebRunner" - later known as "The HotJava Browser".

An applet is a small Internet-based program written in Java, a programming language for the Web, which can be downloaded by any computer. The applet is also able to run in HTML. The applet is usually embedded in an HTML page on a Web site and can be executed from within a browser.

Advantage of Applet

There are many advantages of applet. They are as follows:

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

Drawback of Applet

- Plugin is required at client browser to execute applet.

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.

```
//First.java
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>
```

Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{

public void paint(Graphics g){
g.drawString("welcome to applet",150,150);
}

}
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

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.

Example of Graphics in applet:

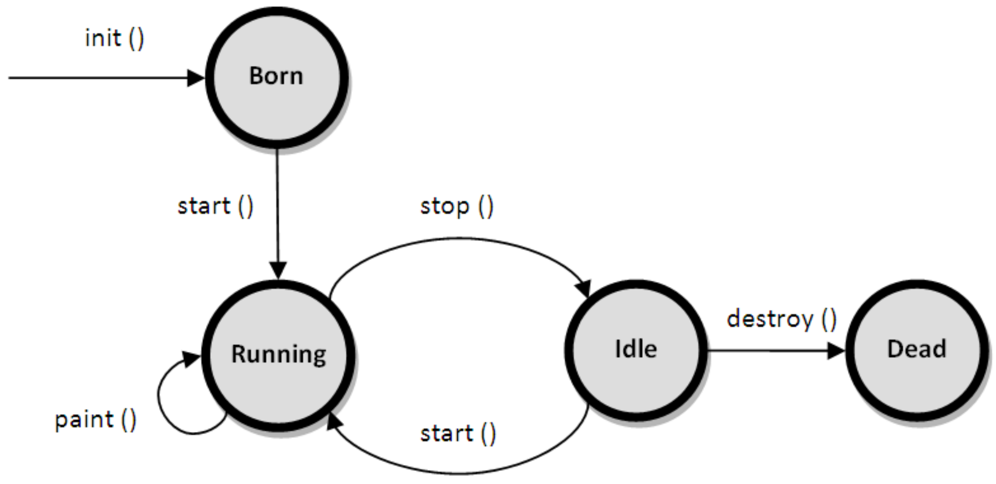
```
import java.applet.Applet;
import java.awt.*;

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>
```

Applet Life Cycle



There are four java.Applet class methods that define the applet life cycle.

They are:

- **public void init()**: This method initializes the Applet and is invoked only once in the Applet life cycle. It helps to initialize variables and instantiate the objects and load the GUI of the applet. This is invoked when the page containing the applet is loaded.

```
public void init( )
//Action to be performed
}
```

- **public void start():** This method is invoked after the init() method. It starts the execution of Applet. In this state, the applet becomes active.

```
public void start( )
{
//Action to be performed
}
```

- **public void stop():** It stops the Applet execution. It's invoked when the Applet stops or when the browser is minimized. This makes the Applet temporarily inactive. The Applet frequently comes to this state in its life cycle and can go back to its start state.

```
public void stope
{
//Action to be performed
}
```

- **public void destroy():** This destroys the Applet and is also invoked only once when the active browser page containing the applet is closed.

```
public void destroy( )

{

//Action to be performed

}
```

- **public void paint() :** This method helps to create Applet's GUI such as a colored background, drawing and writing. It is a method of java.awt.Graphics package.

Example

```
import java.applet.Applet;



import java.awt.Graphics;



import java.awt.*;



class MyfirstApplet extends Applet



{



String str = "";



public void init() // init method

{

str = str + "I am Init ";
}
public void start() // start method
{
str = str + "I am start : ";
}
public void stop() // stop method and holds the applet on pause state
 {
str = str + "I am stop : ";
}
public void destroy() // destroy method, destroys the applet
{
System.out.println("Applet Destroyed.");
}
public void paint(Graphics g) // GUI design of the applet
{
g.drawString(str,500,700);
}
}
```

Java AWT Tutorial

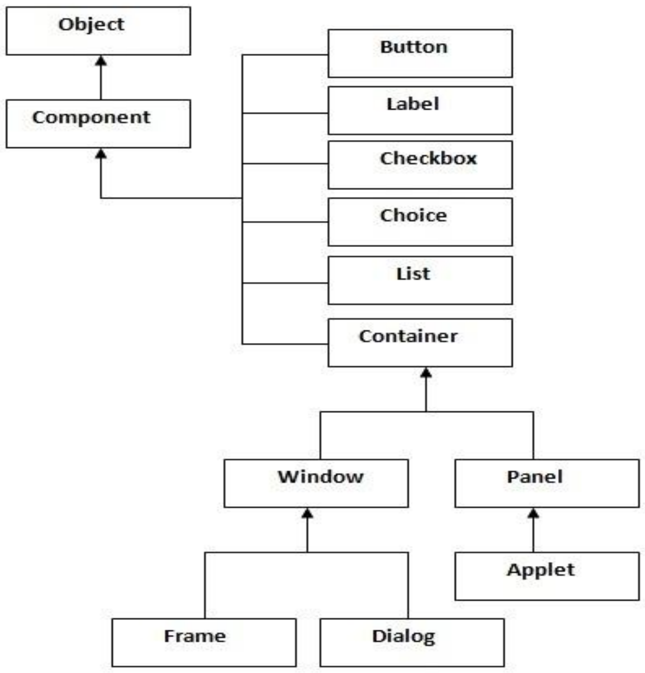
Java AWT (Abstract Windowing Toolkit) is *an API to develop GUI or window-based application in java*.

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 uses the resources of system.

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 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.

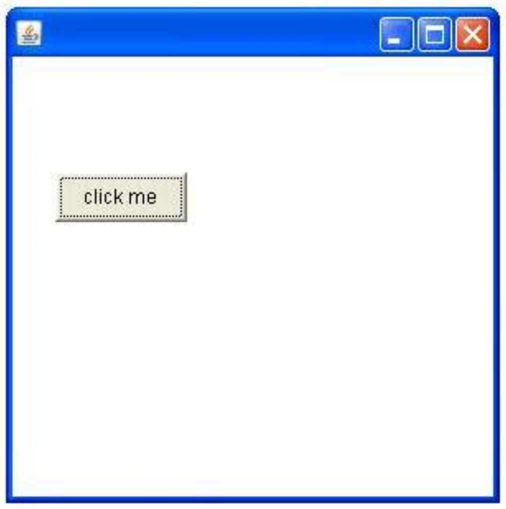
Java AWT Example

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)

Simple example of AWT by inheritance

```
import java.awt.*;
class First extends Frame{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position

add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[]){
First f=new First();
}}

```



Simple example of AWT by association

```
import java.awt.*;
class First2{
```

```java
First2(){
Frame f=new Frame();

Button b=new Button("click me");
b.setBounds(30,50,80,30);

f.add(b);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[]){
First2 f=new First2();
}}
```