**JAVA**

## 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.

**PRODUCT**: JAVA was product of Sun Microsoft, but now it is product of Oracle

## Where it is used?

According to Oracle, 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

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.

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

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

7) **Why had they choosen 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).

# 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. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

Java Features

### 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. Where Garbage Collection means unreferenced objects.  Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects  To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.  Advantage of Garbage Collection   * It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory. * It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.  Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).How can an object be unreferenced? There are many ways:   * By nulling the reference * By assigning a reference to another * By anonymous object etc.  1) By nulling a reference:  1. Employee e=**new** Employee(); 2. e=**null**;  2) By assigning a reference to another:  1. Employee e1=**new** Employee(); 2. Employee e2=**new** Employee(); 3. e1=e2;//now the first object referred by e1 is available for garbage collection  3) By annonymous object:  1. **new** Employee();  finalize() method The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:   1. **protected** **void** finalize(){}  Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).gc() method The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.   1. **public** **static** **void** gc(){}  Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.Simple Example of garbage collection in java  1. **public** **class** TestGarbage1{ 2. **public** **void** finalize(){System.out.println("object is garbage collected");} 3. **public** **static** **void** main(String args[]){ 4. TestGarbage1 s1=**new** TestGarbage1(); 5. TestGarbage1 s2=**new** TestGarbage1(); 6. s1=**null**; 7. s2=**null**; 8. System.gc(); 9. } 10. } |

### Object-oriented

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

java is 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 is a software-based platform that runs on the 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**
* **Java Programs run inside virtual machine sandbox**

how java is secured

* **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 fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

### 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 doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications etc.

## JAVA Installation:

**Download JDK version 7 or any latest version**

**Step 1:** Right Click->Run as Administrator->Install

Step 2: Check where JAVA Installed in c folder-> C:>program Files86

Step 3: My Computer->rightClick->Advance System Setting->Env. Variables->Click on New->

Variable Name - JAVA\_HOME

Variable Value – C:\Program Files (x86)\Java\jdk1.7.0\_40

Step 4: Create on more new Variable :

Variable Name – JRE\_HOME (in always capital letter)

Variable Value – C:\Program Files (x86)\Java\jre7

Step 5: Edit default **path** variable and append the text:

;%JAVA\_HOME%\bin;%JRE\_HOME%\bin; (defined in global variable)

Step 5: open cmd: java –version ( to check the java version and java has been installed properly)

Step 6: Install Eclipse IDE

# What are environment variables in Windows?

Environment variables are, in short, variables that describe the environment in which programs run.

In Windows, **environment variables have a name and value**. For example, the variable *windir* (short for "Windows directory") may have the value *"C:\Windows"* or another path where Windows is installed.

# Types of environment variables: User and System

User environment variables have values that differ from user to user. As their name implies, they are specific to your user account, and they store user specific data, like the location of your user profile, the folder where temporary files are stored for your account, the location of your OneDrive folder, and so on. They can be edited by that user account but not by other user accounts. These variables can be set by the user, by Windows or by different programs working with user specific locations.

System variables are global and cannot be changed by any user. Their values are the same for all user accounts. They refer to locations of critical system resources, like the folder where Windows is installed, the location of Program Files, and so on. These variables are set by Windows, by different programs and drivers.

# How to Set Path and Classpath in Java

Setting path and classpath in Java is very simple but before do this process you need to know about path variable and classpath variable. Here I will show you **how to set path and classpath in Java** in very simple and easy way.

### Path Variable

Path variable is set for providing path for all Java tools like java, javac, javap, javah, jar, appletviewer which are used in java programming. All these tools are available in **bin** folders so we set path upto bin folders.

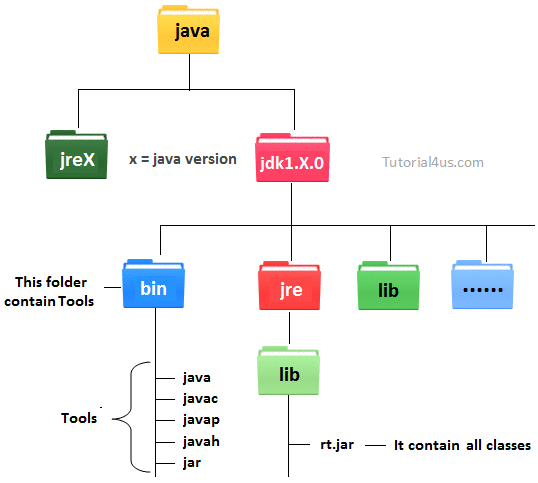
### Classpath Variable

**Classpath** variable is set for providing a path for predefined Java classes which is used in our application. All classes are available in **lib/rt.jar** so we set classpath upto lib/rt.jar.

### Video Tutorial

[How to Set Path and ClassPath in Java](https://www.youtube.com/watch?v=tUD5ilcqkiM)

## JDK Folder Hierarchy



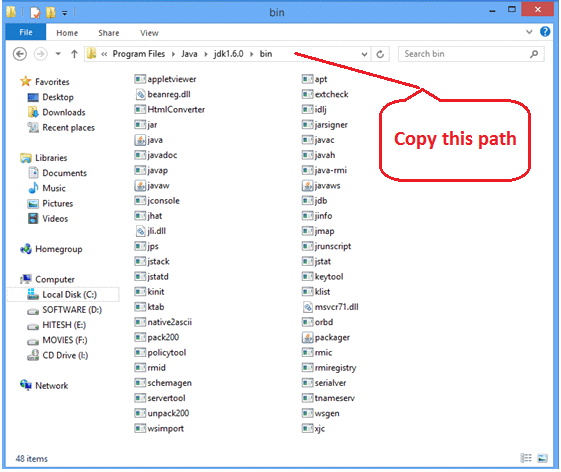
## Why set path ?

The following programming error is generally for all Java programmers when they compile any Java program.

**'javac' is not recognized as an internal or external command, operable program or batch file.**

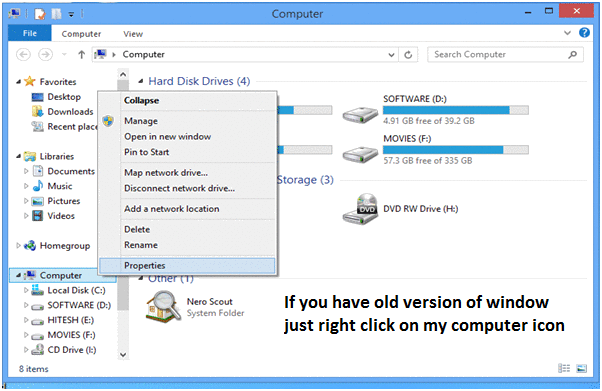
When you get this type of error, then your operating system cannot find the Java compiler **(javac)**. To solve this error you need to set the PATH variable.

**Javac** is a tool which is available in bin folder so you must set the PATH upto bin folder. In a **bin**folder all tools are available like **javap, javah, jar, javac, java, appletviewer** etc. All these tools are used for different-different purpose.

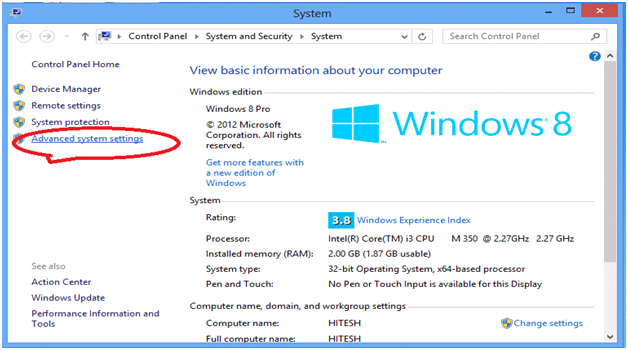


## set the path and classpath

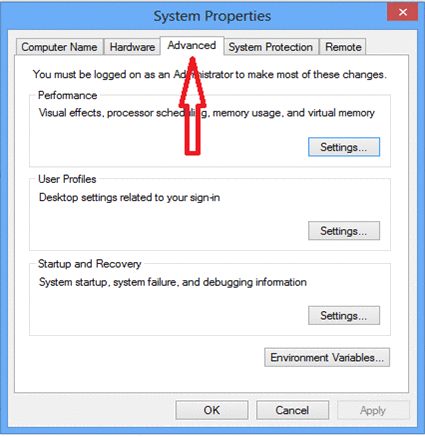
Go on my computer icon and right click, after that click on properties option.



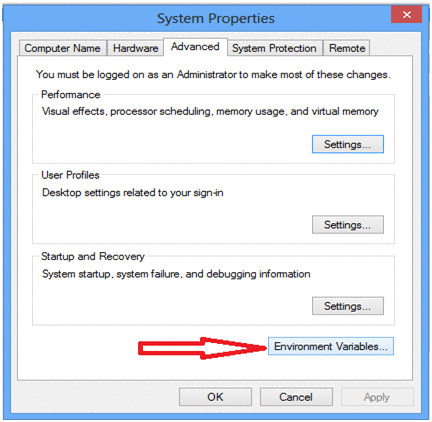
Now click on advance setting



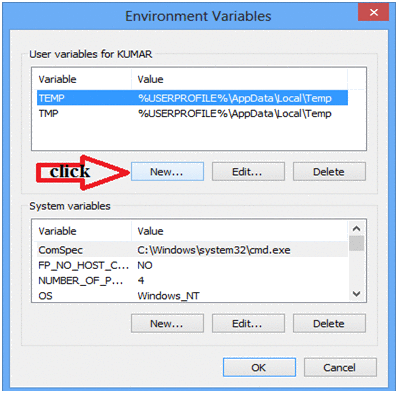
Click on advance



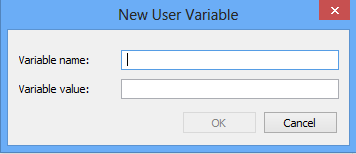
Click on Advance variables



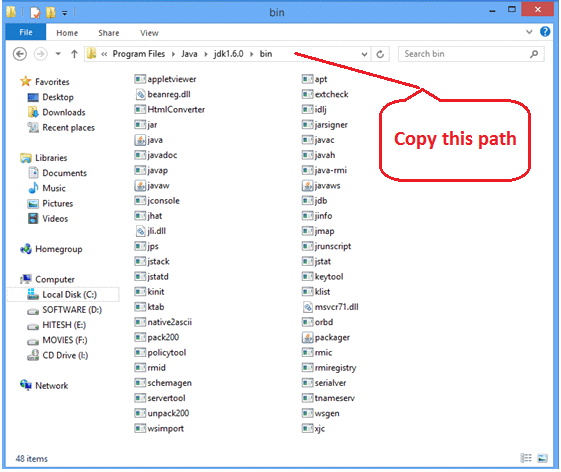
Click on new button which is below the first box.



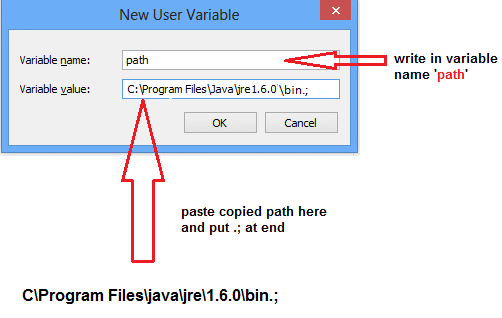
Now one dilog box is appear, now ignore this but do not close.



Now open my computer open c:/ > Programs Files > java > java1.6.0 > bin copy this path

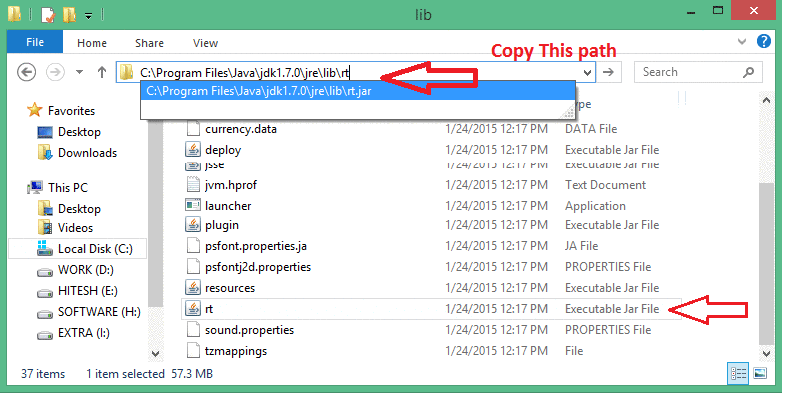


Now come back on previous open dilogbox and write variable name **'path'** and for variable value paste all copied path upto the bin folder. Put .; at the end. It (.) selects all the tools from the bin folder.

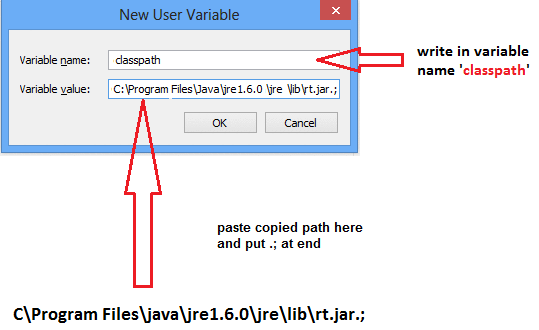
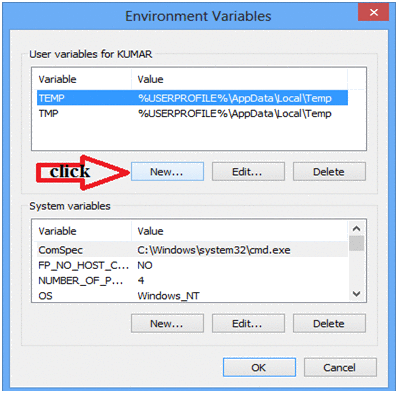


Now open my computer open c:/ > Programs Files > java > java1.6.0 > jre > lib > rt.jar copy this path

**Note:**rt.jar is available in lib folder this jar files contains all classes of jdk.



Now again come back on Environment variable dilogbox and click on new. Now one box is open and write path variable as **'classpath'** and for variable value paste all copied paths upto rt.jar. Put .; at the end. It (.) selects all the classes from lib folder.



**Note:**Finally after set classpath Restart your system, or you can re-open command prompt.

# C++ vs Java

There are many differences and similarities between C++ programming language and Java. A list of top differences between C++ and Java are given below:

|  |  |  |
| --- | --- | --- |
| **Comparison Index** | **C++** | **Java** |
| Platform-independent | C++ is platform-dependent. | Java is platform-independent. |
| Mainly used for | C++ is mainly used for system programming. | Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications. |
| Goto | C++ supports goto statement. | Java doesn't support goto statement. |
| Multiple inheritance | C++ supports multiple inheritance. | Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java. |
| Operator Overloading | C++ supports operator overloading. | Java doesn't support operator overloading. |
| Pointers | C++ supports pointers. You can write pointer program in C++. | Java supports pointer internally. But you can't write the pointer program in java. It means java has restricted pointer support in java. |
| Compiler and Interpreter | C++ uses compiler only. | Java uses compiler and interpreter both. |
| Call by Value and Call by reference | C++ supports both call by value and call by reference. | Java supports call by value only. There is no call by reference in java. |
| Structure and Union | C++ supports structures and unions. | Java doesn't support structures and unions. |
| Thread Support | C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support. | Java has built-in thread support. |
| Documentation comment | C++ doesn't support documentation comment. | Java supports documentation comment (/\*\* ... \*/) to create documentation for java source code. |
| Virtual Keyword | C++ supports virtual keyword so that we can decide whether or not override a function. | Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default. |
| unsigned right shift >>> | C++ doesn't support >>> operator. | Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator. |
| Inheritance Tree | C++ creates a new inheritance tree always. | Java uses single inheritance tree always because all classes are the child of Object class in java. Object class is the root of inheritance tree in java. |

**TYPES OF JAVA:**

Core JAVA

J2SE

J2EE ( Enterprise Edition)

**SYNTAX: simple.java-----🡪**

Public class java{

public static void main (String[] args){

}

}

**Public – modifier | class – keyword | java-classname**

**Classname Naming Convention:**

1. Class name should not starts with number , space, specialcharacters

public static void main (String[] args){

}

**Public – modifier | static – we can access the method without creating any object of that class because main execution starts from here | void – no return type | main- method,main execution start from here | string[] args – pass the arguments of string array**

**Program will not be execute with this**

JAVAC – JAVA COMPILER

JVM - Java Virtual Machine

Firstly java file compiled by the JAVA complier and create .class file, then .class file execute on JVM.

In Editor Ecllipse , program complie along with code and displays the error and when click on RUN, file directly run on JVM.

|  |  |
| --- | --- |
| **To compile:** | javac Simple.java |
| **To execute:** | java Simple |

It creates the .classpath file automatically when program execute. And compile file save in bin folder.

## Understanding first java program

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

* **class** keyword is used to declare a class in java.
* **public** keyword is an access modifier which represents visibility, it means it is visible to all.
* **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
* **void** is the return type of the method, it means it doesn't return any value.
* **main** represents startup of the program.
* **String[] args** is used for command line argument. We will learn it later.
* **System.out.println()** is used print statement. We will learn about the internal working of System.out.println statement later

**WHY NEED PSVM ?**

When file execute on JVM, firstly it finds the PSVM (public static void main) in file,if it doesnot find the PSVM in file, it will not execute any code.PSVM execute automatically.

Eg. If in one project have 100 classes, then atleast one class should have one PSVM to execute the file.

Its not necessary to write PSVM in each file.

To Print on Screen - System.out.println() - for new line.

System.out.print() - in sam eline

Concatination in JAVA with ‘+’ .

**COMMENTS**: - Multiline Comment: ctrl+shift+/ (comment) | (uncomment – ctrl+shift+\)

Single Line – ctrl+/ (comment/uncomment)

## How many ways can we write a java program

There are many ways to write a java program. The modifications that can be done in a java program are given below:

**1) By changing sequence of the modifiers, method prototype is not changed.**

Let's see the simple code of main method.

1. static public void main(String args[])

**2) subscript notation in java array can be used after type, before variable or after variable.**

Let's see the different codes to write the main method.

1. public static void main(String[] args)
2. public static void main(String []args)
3. public static void main(String args[])

**3) You can provide var-args support to main method by passing 3 ellipses (dots)**

Let's see the simple code of using var-args in main method. We will learn about var-args later in Java New Features chapter.

1. public static void main(String... args)

**4) Having semicolon at the end of class in java is optional.**

Let's see the simple code.

class A{

static public void main(String... args){

System.out.println("hello java4");

}

};

## Valid java main method signature

1. public static void main(String[] args)
2. public static void main(String args[])
3. public static void main(String... args)
4. static public void main(String[] args)
5. public static final void main(String[] args)
6. final public static void main(String[] args)
7. final strictfp public static void main(String[] args)

## Invalid java main method signature

1. public void main(String[] args)
2. static void main(String[] args)
3. public void static main(String[] args)
4. abstract public static void main(String[] args)

# Internal Details of Hello Java Program

## What happens at compile time?

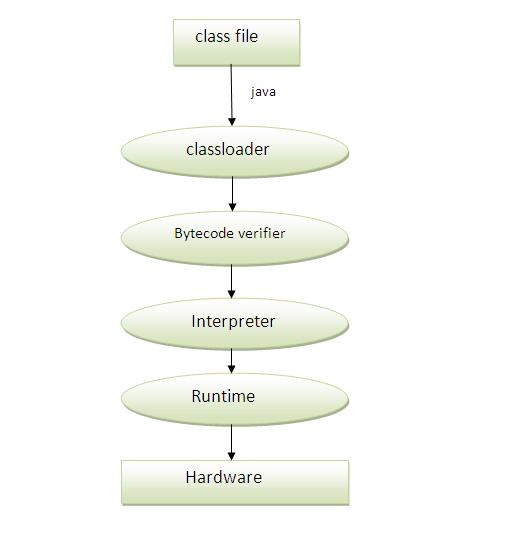
At compile time, java file is compiled by Java Compiler (It does not interact with OS) and converts the java code into bytecode.

compilation of simple java program

## What happens at runtime?

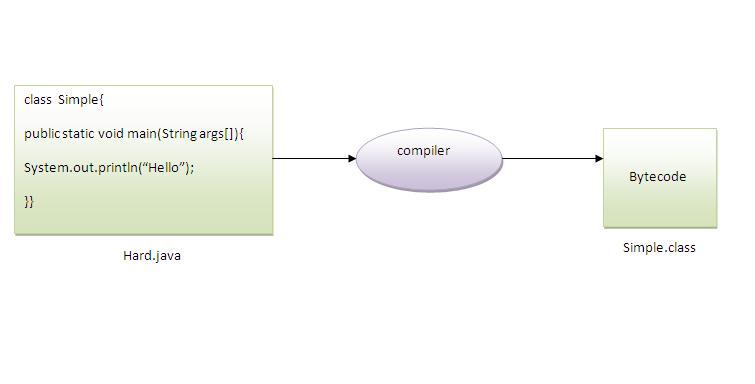
At runtime, following steps are performed:

|  |
| --- |
| **Classloader:** is the subsystem of JVM that is used to load class files. |
| **Bytecode Verifier:** checks the code fragments for illegal code that can violate access right to objects. |
| **Interpreter:** read bytecode stream then execute the instructions. |



## Can you save a java source file by other name than the class name?

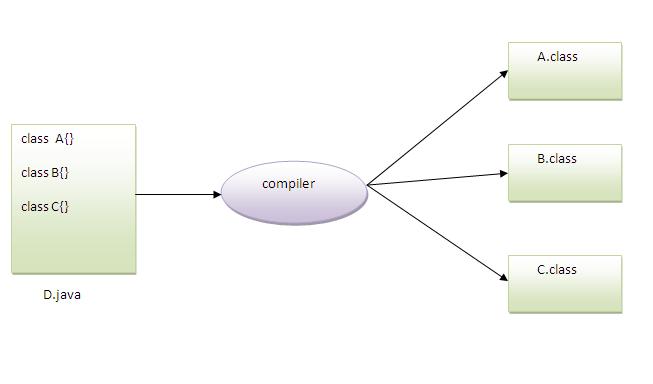
Yes, if the class is not public. It is explained in the figure given below:



|  |  |
| --- | --- |
| **To compile:** | javac Hard.java |
| **To execute:** | java Simple |

## Q)Can you have multiple classes in a java source file?

Yes, like the figure given below illustrates:



# Difference between JDK, JRE and JVM

**JVM**

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

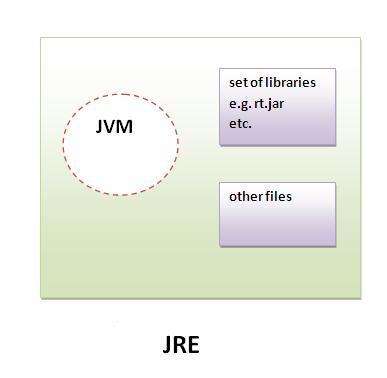
The JVM performs following main tasks:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

### JRE

JRE is an acronym for Java Runtime Environment. It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.

Implementation of JVMs is also actively released by other companies besides Sun Micro Systems.



### JDK

JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools.



# JVM (Java Virtual Machine)

1. [Java Virtual Machine](https://www.javatpoint.com/internal-details-of-jvm)
2. [Internal Architecture of JVM](https://www.javatpoint.com/internal-details-of-jvm#jvminternalarch)

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java byte code can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

# What is JVM

It is:

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Sun and other companies.
2. **An implementation** its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

# What it does

The JVM performs following operation:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

JVM provides definitions for the:

* Memory area
* Class file format
* Register set
* Garbage-collected heap
* Fatal error reporting etc.

# Internal Architecture of JVM

|  |
| --- |
| Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc. |



### 1) Classloader

Classloader is a subsystem of JVM that is used to load class files.

### 2) Class(Method) Area

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

### 3) Heap

It is the runtime data area in which objects are allocated.

### 4) Stack

|  |
| --- |
| Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return. |
| Each thread has a private JVM stack, created at the same time as thread. |
| A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes. |

### 5) Program Counter Register

PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

### 6) Native Method Stack

It contains all the native methods used in the application.

### 7) Execution Engine

|  |
| --- |
| It contains: |
| **1) A virtual processor** |
| **2) Interpreter:** Read bytecode stream then execute the instructions. |
| **3) Just-In-Time(JIT) compiler:** It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU. |

# Does JVM create object of Main class (the class with main())?

Consider following program.

|  |
| --- |
| class Main {      public static void main(String args[])      {          System.out.println("Hello");      }  } |

Hello

Does JVM create an object of class Main?  
The answer is “No”. We have studied that the reason for main() static in Java is to make sure that the main() can be called without any instance. To justify the same, we can see that the following program compiles and runs fine.

|  |
| --- |
| // Not Main is abstract  abstract class Main {      public static void main(String args[])      {          System.out.println("Hello");      }  } |

Output:

Hello

Since we can’t create object of [abstract classes in Java](http://www.geeksforgeeks.org/abstract-classes-in-java/), it is guaranteed that object of class with main() is not created by JVM.

# Is main method compulsory in Java?

The answer to this question depends on version of java you are using. Prior to JDK 5, main method was not mandatory in a java program.

* You could write your full code under [static block](http://www.geeksforgeeks.org/g-fact-79/) and it ran normally.
  + The static block is first executed as soon as the class is loaded before the main(); method is invoked and therefore before the main() is called. main is usually declared as static method and hence [Java doesn’t need an object to call main method.](http://www.geeksforgeeks.org/jvm-create-object-main-class-class-contains-main/)

However, From JDK6 main method is mandatory. If your program doesn’t contain main method, then you will get a run-time error “main method not found in the class”. Note that your program will successfully compile in this case, but at run-time, it will throw error.

|  |
| --- |
| // This program will successfully run  // prior to JDK 5  public class Test  {      // static block      static      {          System.out.println("program is running without main() method");      }  } |

Run on IDE

Output:

* If run prior to JDK 5
* program is running without main() method
* If run on JDK 6,7,8
* Error: Main method not found in class Test, please define the main method as:

public static void main(String[] args)

# Myth about the file name and class name in Java

[**1.6**](http://www.geeksforgeeks.org/basic/)

The first lecture note given during java class is “In java file name and class name should be the same”. When the above law is violated a compiler error message will appear as below

|  |
| --- |
| /\*\*\*\*\* File name: Trial.java \*\*\*\*\*\*/  public class Geeks  {     public static void main(String[] args) {          System.out.println("Hello world");     }  } |

Run on IDE

Output:

javac Trial.java

Trial.java:9: error: class Geeks is public, should be

declared in a file named Geeks.java

public class Geeks

^

1 error

But the myth can be violated in such a way to compile the above file.

|  |
| --- |
| /\*\*\*\*\* File name: Trial.java \*\*\*\*\*\*/  class Geeks  {      public static void main(String[] args) {          System.out.println("Hello world");      }  } |

Run on IDE

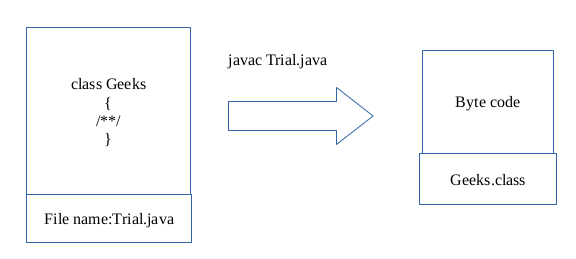
Step 1: javac Trial.java

Step1 will create a Geeks.class (byte code) without any error message since the class is not public.

Step 2: java Geeks

Now the output will be **Hello world**

The myth about the file name and class name should be same only when the class is declared in  
**public**.

The above program works as follows :  
[](http://www.geeksforgeeks.org/wp-content/uploads/javaex.png)

Now this .class file can be executed. By the above features some more miracles can be done. It is possible to have many classes in a java file. For debugging purposes this approach can be used. Each class can be executed separately to test their functionalities(only on one condition: Inheritance concept should not be used).

But in general it is good to follow the myth.

For example:

|  |
| --- |
| /\*\*\* File name: Trial.java \*\*\*/  class ForGeeks  {     public static void main(String[] args){        System.out.println("For Geeks class");     }  }  class GeeksTest  {     public static void main(String[] args){        System.out.println("Geeks Test class");     }  } |

Run on IDE

When the above file is compiled as **javac Trial.java**will create two .class files as **ForGeeks.class and GeeksTest.class**.  
Since each class has separate main() stub they can be tested individually.  
When **java ForGeeks** is executed the output is **For Geeks class**.  
When **java GeeksTest** is executed the output is **Geeks Test class**.

# How to run java class file which is in different directory?

[**2.8**](http://www.geeksforgeeks.org/easy/)

In this article we will learn about how to use other project’s utilities, classes and members. Before proceeding let’s learn about some keywords.

**classpath**

Classpath is the location from where jvm starts execution of a program. Similar to the classic dynamic loading behavior, when executing Java programs, the Java Virtual Machine finds and loads classes lazily (it loads the bytecode of a class only when the class is first used). The classpath tells Java where to look in the filesystem for files defining these classes. Variables and methods which are accessible and available at classpath are known as classpath variables. By default JVM always access the classpath classes while executing a program. JVM always go into the deep of classpath to search a class or resource.

**The JVM searches for and loads classes in this order:**

1. **bootstrap classes:** the classes that are fundamental to the Java Platform (comprising the public classes of the Java Class Library, and the private classes that are necessary for this library to be functional).
2. **extension classes:** packages that are in the extension directory of the JRE or JDK, jre/lib/ext/ user-defined packages and libraries

**Using import keyword**

import keyword is used in Java to import classes from current project’s classpath. You can import classes from different packages but from same classpath. It is to be remembered that packaging of a class starts from classpath. Suppose you have directory structure as follows:

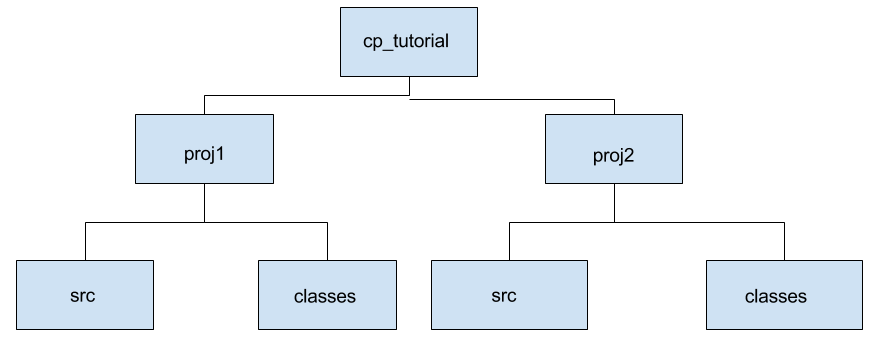
a > b > c > d > class A

and your classpath starts from c, **then your class should be in package d not in a.b.c.d**.

**Using classpath -cp option**

import keyword can import classes from current classpath, outside the classpath import can’t be used. Now suppose you already have a project in which you have used some utility classes, which you need in your second project also. Then in this situation import keyword doesn’t work because your first project is at another classpath. In that case you can use **-cp** command while compiling and executing your program.

Let’s proceed with following example. Create a directory structure as shown in figure below.

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/cp_tutorial.png)

Here you have 2 projects proj1 and proj2. proj1 contains src and classes. In src directory, we will keep .java files that is source files and in classes directory we will keep .classes files that is files generated after compiling the project.  
**Following are the steps to run java class file which is in different directory:**

* **Step 1 (Create utility class):** Create A.java in src directory containing following code.

|  |
| --- |
| //java utility class  public class A  {      public void test()      {          System.out.println("Test() method of class A");      }  } |

* Run on IDE
* Here, We have one utility class that is A.
* **Step 2 (Compile utility class):** Open terminal at proj1 location and execute following commands.
* cp\_tutorial/proj1>cd src
* cp\_tutorial/proj1/src>javac -d ../classes A.java

**-d option:** It is used to store the output to different directory. If we don’t use this option then class file will be created in src directory. But it’s a good practice to keep source and class files seperately. after -d option we provide the location of directory in which class files should be stored.  
If there is any compile time error please resolve it before going further.

* **Step 3 (Check whether A.java is successfuly compiled):** Check in classes directory of proj1 whether class file is created or not. It will be certainly Yes if your program was Compiled successfully.
* **Step 4 (Write main class and compile it):** Move to your proj2 directory. Here are also 2 directories for the same reasons. Create MainClass.java in src directory having following content and try to compile it.

|  |
| --- |
| //java class to execute program  public class MainClass{      public static void main(String[] args){          System.out.println("In main class");          A a1 = new A();          a1.test();      }  } |

* Run on IDE
* cp\_tutorial/proj2>cd src
* cp\_tutorial/proj2/src>javac -d ../classes MainClass.java
* MainClass.java:4: error: cannot find symbol
* A a1 = new A();
* ^
* symbol: class A
* location: class MainClass
* MainClass.java:4: error: cannot find symbol
* A a1 = new A();
* ^
* symbol: class A
* location: class MainClass
* 2 errors
* As you see, there is a compile time error that symbol A is not found. If, we want to use class A of proj1 then we have to use -cp option to include proj1’s resources as shown in next step.
* **Step 5 (Compile with -cp option):**
* cp\_tutorial/proj2>cd src
* cp\_tutorial/proj2/src>javac -d ../classes -cp
* ../../proj1/classes MainClass.java

Now, your code will be compiled successfully and MainClass.class is created in classes directory. -cp stands for classpath and it includes the path given with current classpath and once it is included jvm recognizes the symbol A that it is a class and compiles the file successfully.

* **Step 6 (Execute the program):** Try excuting the program.

execute following commands to run your program.

cp\_tutorial/proj2/src>cd ../classes

cp\_tutorial/proj2/classes>java MainClass

Exception in thread "main" java.lang.NoClassDefFoundError: A

at MainClass.main(MainClass.java:4)

Caused by: java.lang.ClassNotFoundException: A

at java.net.URLClassLoader$1.run(Unknown Source)

at java.net.URLClassLoader$1.run(Unknown Source)

at java.security.AccessController.doPrivileged(Native Method)

at java.net.URLClassLoader.findClass(Unknown Source)

at java.lang.ClassLoader.loadClass(Unknown Source)

at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)

at java.lang.ClassLoader.loadClass(Unknown Source)

... 1 more

Oops, we got an error that class A is not found. This is because you tell JVM about A’s path at compile time only. While running the MainClass, he doesn’t know that there is a class A in other project.

* **Step 7 (Execute with -cp option):** We have to again provide the path of class A.
* cp\_tutorial/proj2/classes>java -cp ../../proj1/classes; MainClass
* In main class
* Test() method of class A

Now, you have successfuly run your program. Don’t forget to include ; after provided classpath.

**How to run a java class with a jar in the classpath?**

You can also use jar file instead of class files from different classpath. The process will be same, you just have to replace classes folder with jar folder and class name with jar name.  
Suppose you have jar file in lib directory, then to compile you can use

cp\_tutorial/proj2/src>javac -d ../classes -cp ../../proj1/lib MainClass.java

and to execute

cp\_tutorial/proj2/classes>java -cp ../../proj1/lib; MainClass

# Using predefined class name as Class or Variable name in Java

[**2.6**](http://www.geeksforgeeks.org/easy/)

In Java, Using predefined class name as Class or Variable name is allowed. However, According to [Java Specification Language(§3.9)](http://docs.oracle.com/javase/specs/jls/se8/html/jls-3.html#jls-3.9) the basic rule for naming in Java is that you cannot use a keyword as name of a class, name of a variable nor the name of a folder used for package.  
Using any predefined class in Java won’t cause such compiler error as Java predefined classes are not keywords.

**Following are some invalid class names in Java:**

boolean break = false; // not allowed as break is keyword

int boolean = 8; // not allowed as boolean is keyword

boolean goto = false; // not allowed as goto is keyword

String final = "hi"; // not allowed as final is keyword

**Using predefined class name as User defined class name**

1. **Question :**Can we have our class name as one of the predefined class name in our program?  
   **Answer :**Yes we can have it. Below is example of using **Number** as user defined class

|  |
| --- |
| // Number is predefined class name in java.lang package  // Note : java.lang package is included in every java program by default  public class Number  {      public static void main (String[] args)      {          System.out.println("It works");      }  } |

1. Run on IDE
2. Output:
3. It works
4. **Using String as User Defined Class:**

|  |
| --- |
| // String is predefined class name in java.lang package  // Note : java.lang package is included in every java program by default  public class String  {      public static void main (String[] args)      {          System.out.println("I got confused");      }  } |

1. Run on IDE
2. However, in this case you will get run-time error like this :
3. Error: Main method not found in class String, please define
4. the main method as:
5. public static void main(String[] args)
6. **Explanation :**This is because [Main thread](http://www.geeksforgeeks.org/main-thread-java/)is looking for main method() with predefined **String class** array argument. But here, it got main method() with user defined String class. Whenever Main thread will see a class name, it tries to search that class scope by scope. First it will see in your program, then in your package.If not found, then [JVM](http://www.geeksforgeeks.org/jvm-works-jvm-architecture/) follows delegation hierarchy principle to load that class.Hence you will get run-time error.  
   To run above program, we can also provide full path of String class, i.e. java.lang.String .

|  |
| --- |
| // String is predefined class name in java.lang package  // Note : java.lang package is included in every java program by default  public class String  {      public static void main (java.lang.String[] args)      {          System.out.println("I got confused");      }  } |

1. Run on IDE
2. Output:
3. I got confused

**Using predefined class name as User defined Variable name**

**Question :**Can we have a variable name as one of the predefined class name in our program?  
**Answer :**Yes we can have it.

|  |
| --- |
| // Number is predefined class name in java.lang package  // Note : java.lang package is included in every java program by default  public class Number  {      // instance variable      int Number = 20;        public static void main (String[] args)      {          // reference variable          Number Number = new Number();            // printing reference          System.out.println(Number);            // printing instance variable          System.out.println(Number.Number);      }  } |

Run on IDE

Output:

Number@15db9742

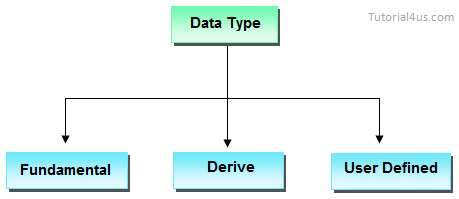
20

## Data Type in Java

**Datatype** is a spacial keyword used to allocate sufficient memory space for the data, in other words Data type is used for representing the data in main memory (RAM) of the computer.

In general every programming language is containing three categories of data types. They are

* Fundamental or primitive data types
* Derived data types
* User defined data types.



## Primitive data types

**Primitive** data types are those whose variables allows us to store only one value but they never allows us to store multiple values of same type. This is a data type whose variable can hold maximum one value at a time.

## Example

**int** a; // valid

a=10; // valid

a=10, 20, 30; // invalid

Here "a" store only one value at a time because it is primitive type variable.

## Derived data types

**Derived** data types are those whose variables allow us to store multiple values of same type. But they never allows to store multiple values of different types. These are the data type whose variable can hold more than one value of similar type. In general derived data type can be achieve using array.

## Example

**int** a[] = {10,20,30}; // valid

**int** b[] = {100, 'A', "ABC"}; // invalid

Here derived data type store only same type of data at a time not store integer, character and string at same time.

## User defined data types

User defined data types are those which are developed by programmers by making use of appropriate features of the language.

User defined data types related variables allows us to store multiple values either of same type or different type or both. This is a data type whose variable can hold more than one value of dissimilar type, in java it is achieved using class concept.

**Note:**In java both derived and user defined data type combined name as reference data type.

In C language, user defined data types can be developed by using struct, union, enum etc. In java programming user defined datatype can be developed by using the features of classes and interfaces.

## Example

Student s = **new** Student();

In java we have eight data type which are organized in four groups. They are

* Integer category data types
* Character category data types
* Float category data types
* Boolean category data types

## Integer category data types

These category data types are used for storing integer data in the main memory of computer by allocating sufficient amount of memory space.

Integer category data types are divided into four types which are given in following table

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Data Type** | **Size** | **Range** |
| 1 | Byte | 1 | + 127 to -128 |
| 2 | Short | 2 | + 32767 to -32768 |
| 3 | Int | 4 | + x to - (x+1) |
| 4 | Long | 8 | + y to - (y+1) |

### Character category data types

A character is an identifier which is enclosed within single quotes. In java to represent character data, we use a data type called char. This data type takes two byte since it follows Unicode character set.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Size(Byte)** | **Range** |
| Char | 2 | 232767 to -32768 |

## Why Java take 2 byte of memory for store character ?

Java support more than 18 international languages so java take 2 byte for characters, because for 18 international language 1 byte of memory is not sufficient for storing all characters and symbols present in 18 languages. Java supports Unicode but c support ascii code. In ascii code only English language are present, so for storing all English latter and symbols 1 byte is sufficient. Unicode character set is one which contains all the characters which are available in 18 international languages and it contains 65536 characters

## Float category data types

Float category data type are used for representing float values. This category contains two data types, they are in the given table

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Type** | **Size** | **Range** | **Number of decimal places** |
| Float | 4 | +2147483647 to -2147483648 | 8 |
| Double | 8 | + 9.223\*1018 | 16 |

## Boolean category data types

Boolean category data type is used for representing or storing logical values is true or false. In java programming to represent Boolean values or logical values, we use a data type called Boolean.

### Why Boolean data types take zero byte of memory ?

Boolean data type takes zero bytes of main memory space because Boolean data type of java implemented by Sun Micro System with a concept of flip - flop. A flip - flop is a general purpose register which stores one bit of information (one true and zero false).

**Note:**In C, C++ (Turbo) Boolean data type is not available for representing true false values but a true value can be treated as non-zero value and false values can be represented by zero

|  |  |  |
| --- | --- | --- |
| **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 |

# CHAPTER 2 - Operators in java

**Operator** in java is a symbol that is used to perform operations. For example: +, -, \*, / etc.

There are many types of operators in java which are given below:

* Unary Operator,
* Arithmetic Operator,
* shift Operator,
* Relational Operator,
* Bitwise Operator,
* Logical Operator,
* Ternary Operator and
* Assignment Operator.

## Java Operator Precedence

|  |  |  |
| --- | --- | --- |
| **Operator Type** | **Category** | **Precedence** |
| Unary | postfix | *expr*++ *expr*-- |
| prefix | ++*expr* --*expr* +*expr* -*expr* ~ ! |
| Arithmetic | multiplicative | \* / % |
| additive | + - |
| Shift | shift | << >> >>> |
| Relational | comparison | < > <= >= instanceof |
| equality | == != |
| Bitwise | bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | | |
| Logical | logical AND | && |
| logical OR | || |
| Ternary | ternary | ? : |
| Assignment | assignment | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

# Java Operators

### Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

### Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

***public******class*** *operators {*

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

***int*** *x=20;*

***int*** *y = x++;*

***int*** *z = ++x;*

*System.****out****.println(z);*

***int*** *w = x++ ;*

*System.****out****.println(w);*

*System.****out****.println(x);*

***int*** *u = x++ + ++x; //23 + 25*

*System.****out****.println(u);*

***int*** *a=50;*

***int*** *b=-10;*

***boolean*** *c=****true****;*

***boolean*** *d=****false****;*

*System.****out****.println(~a);//-11 (minus of total positive value which starts from 0)*

*System.****out****.println(~b);//9 (positive of total minus, positive starts from 0)*

*System.****out****.println(!c);//false (opposite of boolean value)*

*System.****out****.println(!d);//true*

*System.****out****.println(a+b);//15*

*System.****out****.println(a-b);//5*

*System.****out****.println(a\*b);//50*

*System.****out****.println(a/b);//2*

*System.****out****.println(a%b);//0*

*}*

*}*

### Java Ternary Operator Example

*class OperatorExample{*

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

*int a=2;*

*int b=5;*

*int min=(a<b)?a:b;*

*System.out.println(min);*

*}}*

### Java Assignment Operator Example

***class OperatorExample{***

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

***int a=10;***

***int b=20;***

***a+=4;//a=a+4 (a=10+4)***

***b-=4;//b=b-4 (b=20-4)***

***System.out.println(a);***

***System.out.println(b);***

***}}***

### Java Assignment Operator Example: Adding short

***class OperatorExample{***

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

***short a=10;***

***short b=10;***

***//a+=b;//a=a+b internally so fine***

***a=a+b;//Compile time error because 10+10=20 now int***

***System.out.println(a);***

***}}***

# CHAPTER - 3 Variables and Data Types in Java

## Variable

**Variable** is name of reserved area allocated in memory. In other words, it is a name of memory location. It is a combination of "vary + able" that means its value can be changed.

variables in java

#### int data=50;//Here data is variable

## Types of Variable

There are three types of variables in java:

* local variable
* instance variable
* static variable

types of variables in java

#### 1) Local Variable

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

#### 2) Instance Variable

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

#### 3) Static variable

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

**Example to understand the types of variables in java**

***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

Data types represent the different values to be stored in the variable. 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 |

## Why char uses 2 byte in java and what is \u0000 ?

It is because java uses Unicode system than ASCII code system. The \u0000 is the lowest range of Unicode system.

## Java Variable Example: Add Two Numbers

***class Simple{***

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

***int a=10;***

***int b=10;***

***int c=a+b;***

***System.out.println(c);***

***}}***

## Java Variable Example: Widening

***class Simple{***

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

***int a=10;***

***float f=a;***

***System.out.println(a);***

***System.out.println(f);***

***}}***

## Java Variable Example: Narrowing (Typecasting)

***public class simple {***

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

***int a=10;***

***int b=20;***

***int c = a+b;***

***System.out.println(c);***

***int e=30;***

***float f=a;***

***System.out.println(f);***

***float x=(float) 50.1;***

***float x1=100.1f;***

***int y= (int) x;***

***System.out.println(x1);***

***System.out.println(y);***

***}***

***}***

## Java Variable Example: Adding Lower Type

***class Simple{***

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

***byte a=10;***

***byte b=10;***

***//byte c=a+b;//Compile Time Error: because a+b=20 will be int***

***byte c=(byte)(a+b);***

***System.out.println(c);***

***}}***

# Unicode System

|  |
| --- |
| Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages. |

# Why java uses Unicode System?

|  |
| --- |
| Before Unicode, there were many language standards: |
| * **ASCII** (American Standard Code for Information Interchange) for the United States. * **ISO 8859-1** for Western European Language. * **KOI-8** for Russian. * **GB18030 and BIG-5** for chinese, and so on. |

## Problem

|  |
| --- |
| **This caused two problems:**   1. A particular code value corresponds to different letters in the various language standards. 2. The encodings for languages with large character sets have variable length.Some common characters are encoded as single bytes, other require two or more byte. |

## Solution

|  |
| --- |
| To solve these problems, a new language standard was developed i.e. Unicode System. |
| In unicode, character holds 2 byte, so java also uses 2 byte for characters. |
| **lowest value:**\u0000 |
| **highest value:**\uFFFF |

## CHAPTER 4 - CONTROL Statements

The Java *if statement* is used to test the condition. It checks boolean condition: *true* or *false*. There are various types of if statement in java.

* if statement
* if-else statement
* nested if statement
* if-else-if ladder

## Java IF Statement

The Java if statement tests the condition. It executes the if block if condition is true.

**Syntax:**

***if(condition){***

***//code to be executed***

***}***

***Example:***

***public class IfExample {***

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

***int age=20;***

***if(age>18){***

***System.out.print("Age is greater than 18");***

***}***

***}***

***}***

## Java IF-else Statement

The Java if-else statement also tests the condition. It executes the if block if condition is true otherwise else block is executed.

**Syntax:**

***if(condition){***

***//code if condition is true***

***}else{***

***//code if condition is false***

***}***

**Example:**

***public class IfElseExample {***

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

***int number=13;***

***if(number%2==0){***

***System.out.println("even number");***

***}else{***

***System.out.println("odd number");***

***}***

***}***

***}***

## Java IF-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

**Syntax:**

***if(condition1){***

***//code to be executed if condition1 is true***

***}else if(condition2){***

***//code to be executed if condition2 is true***

***}***

***else if(condition3){***

***//code to be executed if condition3 is true***

***}***

***...***

***else{***

***//code to be executed if all the conditions are false***

***}***

**Example**:

***public class IfElseIfExample {***

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

***int marks=65;***

***if(marks<50){***

***System.out.println("fail");***

***}***

***else if(marks>=50 && marks<60){***

***System.out.println("D grade");***

***}***

***else if(marks>=60 && marks<70){***

***System.out.println("C grade");***

***}***

***else if(marks>=70 && marks<80){***

***System.out.println("B grade");***

***}***

***else if(marks>=80 && marks<90){***

***System.out.println("A grade");***

***}else if(marks>=90 && marks<100){***

***System.out.println("A+ grade");***

***}else{***

***System.out.println("Invalid!");***

***}***

***}***

***}***

# Java Switch Statement

The Java switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement.

**Syntax:**

***switch(expression){***

***case value1:***

***//code to be executed;***

***break;  //optional***

***case value2:***

***//code to be executed;***

***break;  //optional***

***......***

***default:***

***code to be executed if all cases are not matched;***

***}***

**Example:**

***public class SwitchExample {***

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

***int number=20;***

***switch(number){***

***case 10: System.out.println("10");break;***

***case 20: System.out.println("20");break;***

***case 30: System.out.println("30");break;***

***default:System.out.println("Not in 10, 20 or 30");***

***}***

***}***

***}***

## Java Switch Statement is fall-through

The java switch statement is fall-through. It means it executes all statement after first match if break statement is not used with switch cases.

**Example:**

***public class SwitchExample2 {***

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

***int number=20;***

***switch(number){***

***case 10: System.out.println("10");***

***case 20: System.out.println("20");***

***case 30: System.out.println("30");***

***default:System.out.println("Not in 10, 20 or 30");***

***}***

***}***

***}***

# CHAPTER 5 - Java For Loop

The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

There are three types of for loop in java.

* Simple For Loop
* For-each or Enhanced For Loop
* Labeled For Loop

## Java Simple For Loop

The simple for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

Syntax:

***for(initialization;condition;incr/decr){***

***//code to be executed***

***}***

**Example:**

***public class ForExample {***

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

***for(int i=1;i<=10;i++){***

***System.out.println(i);***

***}***

***}***

***}***

## Java For-each Loop

The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on elements basis not index. It returns element one by one in the defined variable.

**Syntax:**

***for(Type var:array){***

***//code to be executed***

***}***

**Example:**

***public class ForEachExample {***

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

***int arr[]={12,23,44,56,78};***

***for(int i:arr){***

***System.out.println(i);***

***}***

***}***

***}***

## Java Labeled For Loop

We can have name of each for loop. To do so, we use label before the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop.

Normally, break and continue keywords breaks/continues the inner most for loop only.

**Syntax:**

***labelname:***

***for(initialization;condition;incr/decr){***

***//code to be executed***

***}***

**Example:**

***public class LabeledForExample {***

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

***aa:***

***for(int i=1;i<=3;i++){***

***bb:***

***for(int j=1;j<=3;j++){***

***if(i==2&&j==2){***

***break aa;***

***}***

***System.out.println(i+" "+j);***

***}***

***}***

***}***

***}***

## Java Infinitive For Loop

If you use two semicolons ;; in the for loop, it will be infinitive for loop.

**Syntax:**

***for(;;){***

***//code to be executed***

***}***

**Example:**

***public class ForExample {***

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

***for(;;){***

***System.out.println("infinitive loop");***

***}***

***}***

***}***

# Java While Loop

The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

**Syntax:**

***while(condition){***

***//code to be executed***

***}***

**Example:**

***public class WhileExample {***

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

***int i=1;***

***while(i<=10){***

***System.out.println(i);***

***i++;***

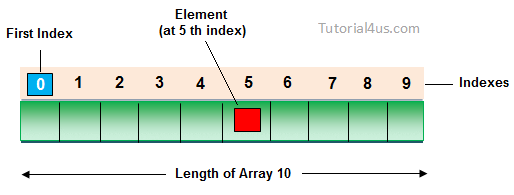
***}***

***}***

***}***

# CHAPTER - ARRAYS

**Array** is a collection of similar type of data. It is fixed in size means that you can't increase the size of array at run time. It is a collection of homogeneous data elements. It stores the value on the basis of the index value.



## Advantage of Array

**One variable can store multiple value:** The main advantage of the array is we can represent multiple value under the same name.

**Code Optimization:** No, need to declare a lot of variable of same type data, We can retrieve and sort data easily.

**Random access:** We can retrieve any data from array with the help of the index value.

## Disadvantage of Array

The main limitation of the array is **Size Limit** when once we declare array there is no chance to increase and decrease the size of an array according to our requirement, Hence memory point of view array concept is not recommended to use. To overcome this limitation in Java introduce the collection concept.

## Types of Array

There are two types of array in Java.

* Single Dimensional Array
* Multidimensional Array

**Array has 3 phases:**

* Declaration
* Construction
* Initialization

## Phase 1 - Array Declaration

**Single dimension array declaration.**

## Syntax

**1. int[] a; //recommended way - > more readable**

**2. int a[];**

**3. int []a;**

**Note:** At the time of array declaration we cannot specify the size of the array. For Example int[5] a; this is wrong.

## Phase 2 - Array Construction:

**Int[] arr=new int[5]; // Every array is an object**

**Or**

**Int[] arr = {}**

**Phase 3 – Initialization**

**arr[0] = 10;**

**arr[1]=20;**

**arr[3]=30;**

# Arrays can also be used to store the objects

Class Pen{}

Pen[] pn = new Pen[3];

pn[0] = new Pen();

pn[1] = new Pen();

Note : 0th index will have the memory address to reach out the

Index of an Array has memory address to reach out the value of array index. But in primitive data type, index store the binary format of that number/value

## Array Initializer:

Int[] marks={20,90,89,90};

# *2D Array Declaration.*

## Syntax

**1. int[][] a;**

**2. int a[][];**

**3. int [][]a;**

**4. int[] a[];**

**5. int[] []a;**

**6. int []a[];**

## Array creation

Every array in a Java is an object, Hence we can create array by using **new** keyword.

## Syntax

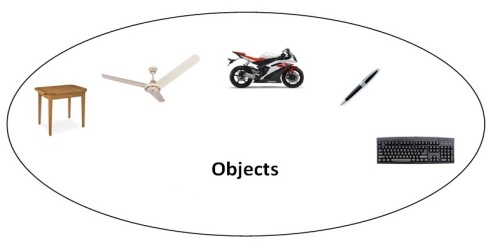
**int[] arr = new int[10]; // The size of array is 10.**

**or**

**int[] arr = {10,20,30,40,50};**

# CHAPTER - Java OOPs Concepts

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

* Object
* Class
* Inheritance
* Polymorphism
* Abstraction
* Encapsulation

## Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

## Class

**Collection of objects** is called class. It is a logical entity.

#### Inheritance

**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

#### Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

#### Abstraction

**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

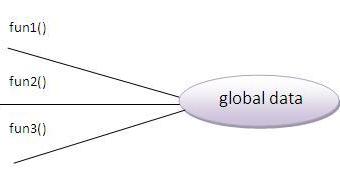
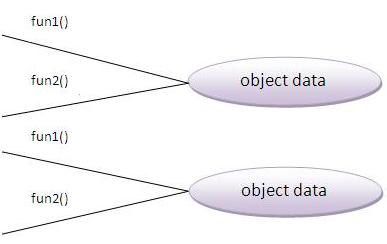
#### Encapsulation

**Binding (or wrapping) code and data together into a single unit is known as encapsulation**. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Advantage of OOPs over Procedure-oriented programming language

|  |
| --- |
| 1)OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows. |
| 2)OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere. |
| 3)OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language. |



## What is difference between object-oriented programming language and object-based programming language?

Object based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object based programming languages.

Java Naming conventions

Java **naming convention** is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.

But, it is not forced to follow. So, it is known as convention not rule.

All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

## Advantage of naming conventions in java

By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that **less time** is spent to figure out what the code does.

|  |  |
| --- | --- |
| **Name** | **Convention** |
| class name | should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc. |
| interface name | should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc. |
| method name | should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc. |
| variable name | should start with lowercase letter e.g. firstName, orderNumber etc. |
| package name | should be in lowercase letter e.g. java, lang, sql, util etc. |
| constants name | should be in uppercase letter. e.g. RED, YELLOW, MAX\_PRIORITY etc. |

## CamelCase in java naming conventions

Java follows camelcase syntax for naming the class, interface, method and variable.

If name is combined with two words, second word will start with uppercase letter always e.g. actionPerformed(), firstName, ActionEvent, ActionListener etc.

Object and Class in Java

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 (tangible and intangible). The example of intangible 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.

**Object Definitions:**

* Object is *a real world entity*.
* Object is *a run time entity*.
* Object is *an entity which has state and behavior*.
* Object is *an instance of a class*.

# Class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

* **fields**
* **methods**
* **constructors**
* **blocks**
* **nested class and interface**

**Syntax to declare a class:**

***class <class\_name>{***

***field;***

***method;***

***}***

Instance variable in Java

A variable which 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 run time 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 behavior of an object.

#### Advantage of Method

* Code Reusability
* Code Optimization

new keyword in Java

***The new keyword is used to allocate memory at run time. All objects get memory in Heap memory area.***

Object and Class Example: main within class

Here, we are creating main() method inside the class.

File: Student.java

***class Student{***

***int id;//field or data member or instance variable***

***String name;***

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

***Student s1=new Student();//creating an object of Student***

***System.out.println(s1.id);//accessing member through reference variable***

***System.out.println(s1.name);***

***}***

***}***

Object and Class Example: main outside class

In real time development, we create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class.

We can have multiple classes in different java files or single java file. If you define multiple classes in a single java source file, it is a good idea to save the file name with the class name which has main() method.

File: TestStudent1.java

***class Student{***

***int id;***

***String name;***

***}***

***class TestStudent1{***

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

***Student s1=new Student();***

***System.out.println(s1.id);***

***System.out.println(s1.name);***

***}***

***}***

3 Ways to initialize object

There are 3 ways to initialize object in java.

1. By reference variable
2. By method
3. By constructor (will discuss later)

object gets the memory in heap memory area. The reference variable refers to the object allocated in the heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.

# 1) Object and Class Example: Initialization through reference

Initializing object simply means storing data into object. Let's see a simple example where we are going to initialize object through reference variable.

# 2) Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

**Example:**

*class Student{*

*int id;*

*String name;*

*public void insertData(int sid, String sname){*

*id=sid;*

*name=sname;*

*}*

*public void showData(){*

*System.out.println("Student Id is:"+id);*

*System.out.println("Student Name is:"+name);*

*}*

*}*

*public class MainOutsideClass {*

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

*Student sobj = new Student();*

***//initialize object By reference variable***

*sobj.id=100;*

*sobj.name="Shaveta";*

*System.out.println(sobj.id);*

*System.out.println(sobj.name);*

*Student sobj2 = new Student();*

***//initialize object By method***

*sobj2.insertData(100,"shaveta");*

*sobj2.showData();*

*}*

*}*

What are the different ways to create an object in Java?

There are many ways to create an object in java. They are:

* By new keyword
* By newInstance() method
* By clone() method
* By deserialization
* By factory method etc.

We will learn these ways to create object later.

Anonymous object

Anonymous simply means nameless. An object which has no reference is known as anonymous object. It can be used at the time of object creation only.

If you have to use an object only once, anonymous object is a good approach. For example:

*new Calculation();//anonymous object*

Calling method through reference:

*Calculation c=new Calculation();*

*c.fact(5);*

Calling method through anonymous object:

**Example:**

***class*** *CalcFactorial{*

***int*** *number;*

***int*** *fact=1;*

***public******void*** *calFactorial(****int*** *num){*

*number=num;*

***for****(****int*** *i=1;i<=number;i++){*

*fact = fact \* i;*

*}*

*System.****out****.println(fact);*

*}*

*}*

***public******class*** *AnonymousObject {*

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

***new*** *CalcFactorial().calFactorial(5);*

*}*

*}*

Creating multiple objects by one type only

We can create multiple objects by one type only as we do in case of primitives.

Initialization of primitive variables:

***int a=10, b=20;***

Initialization of refernce variables:

***Rectangle r1=new Rectangle(), r2=new Rectangle();//creating two objects***

# Example:

***class Rectangle1{***

***int length;***

***int width;***

***void insert(int l,int w){***

***length=l;***

***width=w;***

***}***

***void calculateArea(){***

***System.out.println(length\*width);***

***}***

***}***

***public class MultipleObjectsByOneType {***

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

***Rectangle1 r1=new Rectangle1(),r2=new Rectangle1();//creating two objects***

***r1.insert(11,5);***

***r2.insert(3,15);***

***r1.calculateArea();***

***r2.calculateArea();***

***}***

***}***

# Real World Example:

*class AccountBalance{*

*int accountNumber;*

*String name;*

*float amount;*

*public void insert(int actnum,String pName,float amount){*

*accountNumber=actnum;*

*name=pName;*

*this.amount=amount;*

*}*

*public void deposit(float amt){*

*amount = amount+amt;*

*}*

*public void withDraw(int amt){*

*if(amount<amt){*

*System.out.println("Balance is less than wiothdrwlAmount");*

*}else{*

*amount = amount-amt;*

*}*

*System.out.println(amount);*

*}*

*}*

*public class RealWorldExampleAccount {*

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

*AccountBalance actBlncObj = new AccountBalance();*

*actBlncObj.insert(1234567890,"Shaveta",55000);*

*actBlncObj.deposit(20000);*

*actBlncObj.withDraw(40000);*

*}*

*}*

# Do You Know ?

* Can we overload main method ?
* Constructor returns a value but, what ?
* Can we create a program without main method ?
* What are the 6 ways to use this keyword ?
* Why multiple inheritance is not supported in java ?
* Why use aggregation ?
* Can we override the static method ?
* What is covariant return type ?
* What are the three usage of super keyword?
* Why use instance initializer block?
* What is the usage of blank final variable ?
* What is marker or tagged interface ?
* What is runtime polymorphism or dynamic method dispatch ?
* What is the difference between static and dynamic binding ?
* How downcasting is possible in java ?
* What is the purpose of private constructor?
* What is object cloning ?

# CHAPTER - Constructor in Java

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

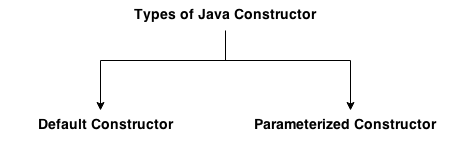
Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.

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**:

***class Bike1{***

***Bike1(){System.out.println("Bike is created");}***

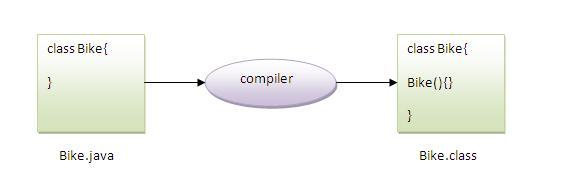
***public static void main(String args[]){***

***Bike1 b=new Bike1();***

***}***

***}***

#### Rule: If there is no constructor in a class, compiler automatically creates a default constructor.



# What is the purpose of default constructor?

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

***public class defaultConstructorWithDefaultValues {***

***int id;***

***String name;***

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

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

***defaultConstructorWithDefaultValues s1=new defaultConstructorWithDefaultValues();***

***defaultConstructorWithDefaultValues s2=new defaultConstructorWithDefaultValues();***

***s1.display();***

***s2.display();***

***}***

***}***

**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**:

***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();***

***}***

***}***

Constructor Overloading in Java

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

***public class ConsturctorOverloading {***

***int id;***

***String name;***

***int age;***

***ConsturctorOverloading(int i,String n){***

***id = i;***

***name = n;***

***}***

***ConsturctorOverloading(int i,String n,int a){***

***id = i;***

***name = n;***

***age=a;***

***}***

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

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

***ConsturctorOverloading s1 = new ConsturctorOverloading(111,"Karan");***

***ConsturctorOverloading s2 = new ConsturctorOverloading(222,"Aryan",25);***

***s1.display();***

***s2.display();***

***}***

***}***

Difference between constructor and method in java

There are many differences between constructors and methods. They are given below.

|  |  |
| --- | --- |
| **Java Constructor** | **Java Method** |
| Constructor is used to initialize the state of an object. | Method is used to expose behaviour of an object. |
| Constructor must not have return type. | Method must have return type. |
| Constructor is invoked implicitly. | Method is invoked explicitly. |
| The java compiler provides a default constructor if you don't have any constructor. | Method is not provided by compiler in any case. |
| Constructor name must be same as the class name. | Method name may or may not be same as class name. |

Java Copy Constructor

There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.

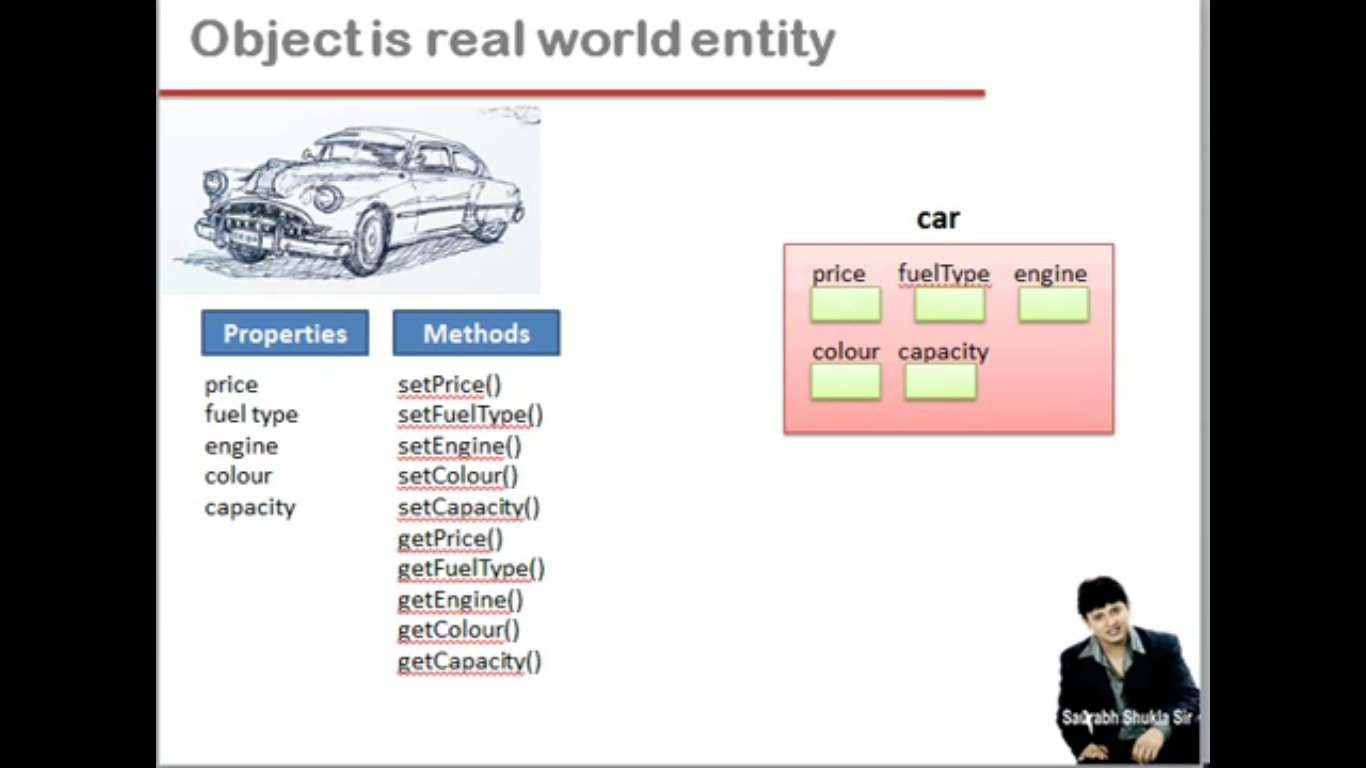
There are many ways to copy the values of one object into another in java. They are:

* By constructor
* By assigning the values of one object into another
* By clone() method of Object class

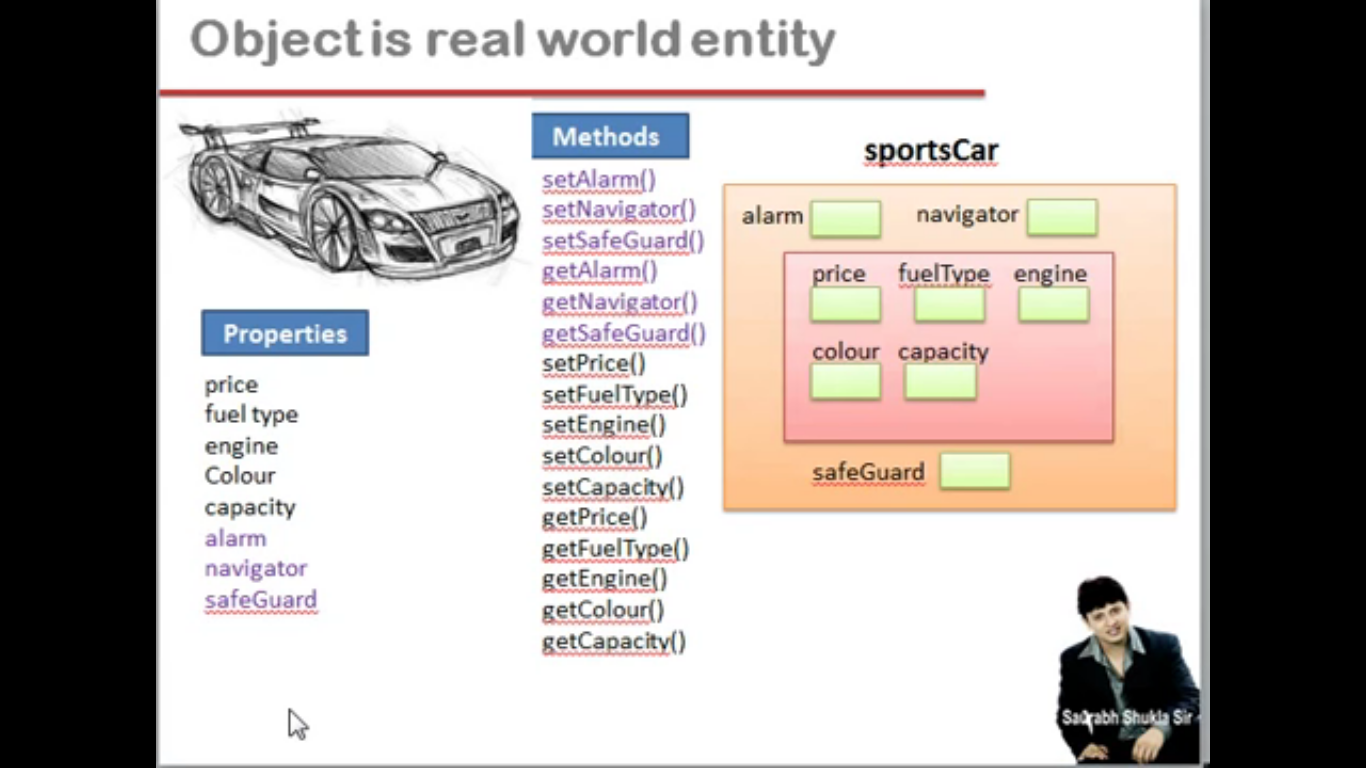
In this example, we are going to copy the values of one object into another using java constructor.

# CHAPTER - INHERITANCE

CAR



Sports CAR – having extra properties



# Inetrface Questions Exercises

## Questions

Question 1: What methods would a class that implements the java.lang.CharSequence interface have to implement?  
Answer 1: charAt, length, subSequence, and toString.

Question 2: What is wrong with the following interface?

public interface SomethingIsWrong {

void aMethod(int aValue) {

System.out.println("Hi Mom");

}

}

Answer 2: It has a method implementation in it. Only default and static methods have implementations.

Question 3: Fix the interface in Question 2.  
Answer 3:

public interface SomethingIsWrong {

void aMethod(int aValue);

}

Alternatively, you can define aMethod as a default method:

public interface SomethingIsWrong {

default void aMethod(int aValue) {

System.out.println("Hi Mom");

}

}

Question 4: Is the following interface valid?

public interface Marker {

}

Answer 4: Yes. Methods are not required. Empty interfaces can be used as types and to mark classes without requiring any particular method implementations. For an example of a useful empty interface, see java.io.Serializable.

## Exercises

**Exercise 1: Write a class that implements the CharSequence interface found in the java.lang package. Your implementation should return the string backwards. Select one of the sentences from this book to use as the data. Write a small main method to test your class; make sure to call all four methods.**

// CharSequenceDemo presents a String value -- backwards.

public class CharSequenceDemo implements CharSequence {

private String s;

public CharSequenceDemo(String s) {

//It would be much more efficient to just reverse the string

//in the constructor. But a lot less fun!

this.s = s;

}

//If the string is backwards, the end is the beginning!

private int fromEnd(int i) {

return s.length() - 1 - i;

}

public char charAt(int i) {

if ((i < 0) || (i >= s.length())) {

throw new StringIndexOutOfBoundsException(i);

}

return s.charAt(fromEnd(i));

}

public int length() {

return s.length();

}

public CharSequence subSequence(int start, int end) {

if (start < 0) {

throw new StringIndexOutOfBoundsException(start);

}

if (end > s.length()) {

throw new StringIndexOutOfBoundsException(end);

}

if (start > end) {

throw new StringIndexOutOfBoundsException(start - end);

}

StringBuilder sub =

new StringBuilder(s.subSequence(fromEnd(end), fromEnd(start)));

return sub.reverse();

}

public String toString() {

StringBuilder s = new StringBuilder(this.s);

return s.reverse().toString();

}

//Random int from 0 to max. As random() generates values between 0 and 0.9999

private static int random(int max) {

return (int) Math.round(Math.random() \* (max+1));

}

public static void main(String[] args) {

CharSequenceDemo s =

new CharSequenceDemo("Write a class that implements the CharSequence interface found in the java.lang package.");

//exercise charAt() and length()

for (int i = 0; i < s.length(); i++) {

System.out.print(s.charAt(i));

}

System.out.println("");

//exercise subSequence() and length();

int start = random(s.length() - 1);

int end = random(s.length() - 1 - start) + start;

System.out.println(s.subSequence(start, end));

//exercise toString();

System.out.println(s);

}

}

**Exercise 2: Suppose that you have written a time server, which periodically notifies its clients of the current date and time. Write an interface that the server could use to enforce a particular protocol on its clients.**  
import java.time.\*;

public interface TimeClient {

void setTime(int hour, int minute, int second);

void setDate(int day, int month, int year);

void setDateAndTime(int day, int month, int year,

int hour, int minute, int second);

LocalDateTime getLocalDateTime();

}

**INTERVIEW QUESTIONS**

### 1. Why a static block executes before the main method ?

A class has to be loaded in main memory before we start using it. Static block is executed during class loading. This is the reason why a static block executes before the main method.

### 2. Why use this keyword in java ?

The main purpose of **using this keyword** is to differentiate the formal parameter and data members of class, whenever the formal parameter and data members of the class are similar then jvm get ambiguity (no clarity between formal parameter and member of the class)

## Usage of this keyword ?

* sIt can be used to refer current class instance variable.
* this() can be used to invoke current class constructor.
* It can be used to invoke current class method (implicitly)
* It can be passed as an argument in the method call.
* It can be passed as argument in the constructor call.
* It can also be used to return the current class instance.