

Java Tutorial

Our core Java programming tutorial is designed for students and working professionals. Java is an object-oriented, class-based, concurrent, secured and general-purpose computer-programming language. It is a widely used robust technology.

What is Java?

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

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

Java Example

Let's have a quick look at Java programming example. A detailed description of Hello Java example is available in next page.

Simple.java

```
1. class Simple{
2.     public static void main(String args[]){
3.         System.out.println("Hello Java");
4.     }
5. }
```

Test it Now

Application

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 types of applications that can be created using Java programming:

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are 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 a dynamic page is called a web application. Currently, [Servlet](#), [JSP](#), [Struts](#), [Spring](#), [Hibernate](#), [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. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, [EJB](#) is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, [String](#), Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2) Java EE (Java Enterprise Edition)

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, [JPA](#), etc.

3) Java ME (Java Micro Edition)

It is a micro platform that is dedicated to mobile applications.

4) JavaFX

It is used to develop rich internet applications. It uses a lightweight user interface API.

Prerequisite

To learn Java, you must have the basic knowledge of C/C++ programming language.

Audience

Our Java programming tutorial is designed to help beginners and professionals.

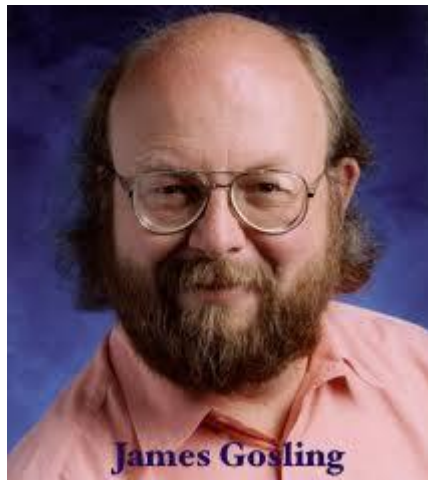
History of Java

1. [History of Java](#)
2. [Java Version History](#)

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of Java starts with the Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was best suited for internet programming. Later, Java technology was incorporated by Netscape.

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic". [Java](#) was developed by James

Gosling, who is known as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s.



Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. Following are given significant points that describe 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) Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.
- 3) Firstly, it was called "**Greentalk**" by James Gosling, and the file extension was .gt.
- 4) After that, it was called **Oak** and was developed as a part of the Green project.

Why Java was named as "Oak"?



- 5) **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like the 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 Programming named "Java"?

7) Why had they chose the name Java 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 than other names.

8) Java is an island in Indonesia where the first coffee was produced (called Java coffee). It is a kind of espresso bean. Java name was chosen by James Gosling while having a cup of coffee nearby his office.

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

10) Initially 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 was released on January 23, 1996. After the first release of Java, there have been many additional features added to the language. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds new features in Java.

Java Version History

Many java versions have been released till now. The current stable release of Java is Java SE 10.

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)

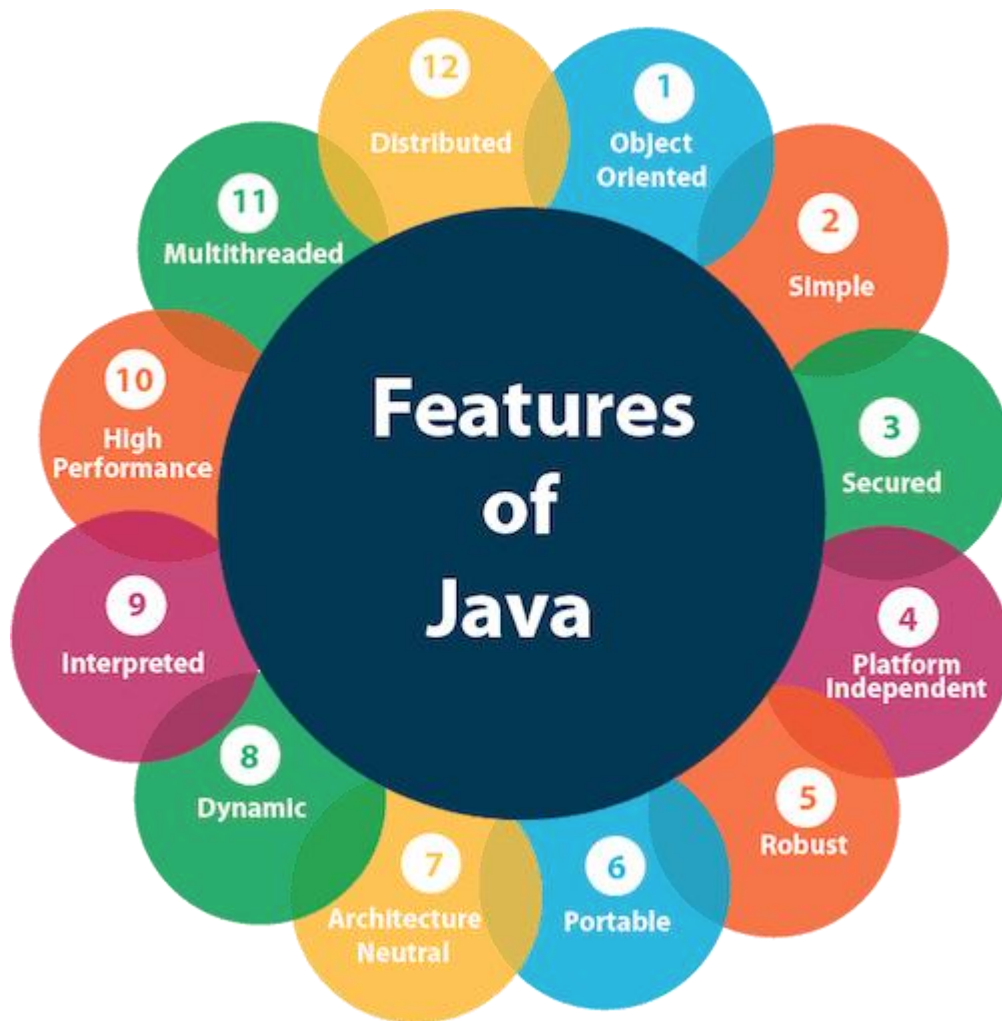
7. J2SE 5.0 (30th Sep 2004)
8. Java SE 6 (11th Dec 2006)
9. Java SE 7 (28th July 2011)
10. Java SE 8 (18th Mar 2014)
11. Java SE 9 (21st Sep 2017)
12. Java SE 10 (20th Mar 2018)
13. Java SE 11 (September 2018)
14. Java SE 12 (March 2019)
15. Java SE 13 (September 2019)
16. Java SE 14 (Mar 2020)
17. Java SE 15 (September 2020)
18. Java SE 16 (Mar 2021)
19. Java SE 17 (September 2021)
20. Java SE 18 (to be released by March 2022)

Since Java SE 8 release, the Oracle corporation follows a pattern in which every even version is release in March month and an odd version released in September month.

Features of Java

The primary objective of [Java programming](#) language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords.

A list of the most important features of the Java language is given below.



1. [Simple](#)
 2. [Object-Oriented](#)
 3. [Portable](#)
 4. [Platform independent](#)
 5. [Secured](#)
 6. [Robust](#)
 7. [Architecture neutral](#)
 8. [Interpreted](#)
 9. [High Performance](#)
 10. [Multithreaded](#)
 11. [Distributed](#)
 12. [Dynamic](#)
-

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystems, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

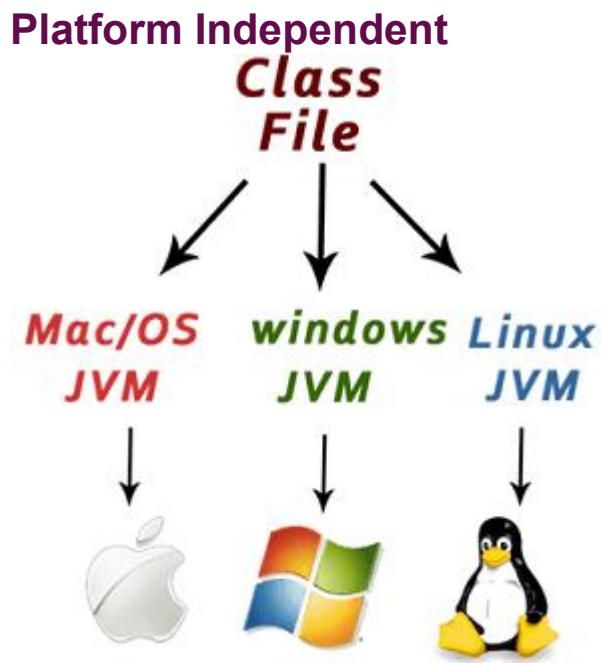
Object-oriented

Java is an [object-oriented](#) programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies 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](#)
-



Java is platform independent because it is different from other languages like [C](#), [C++](#), etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. 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 a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:

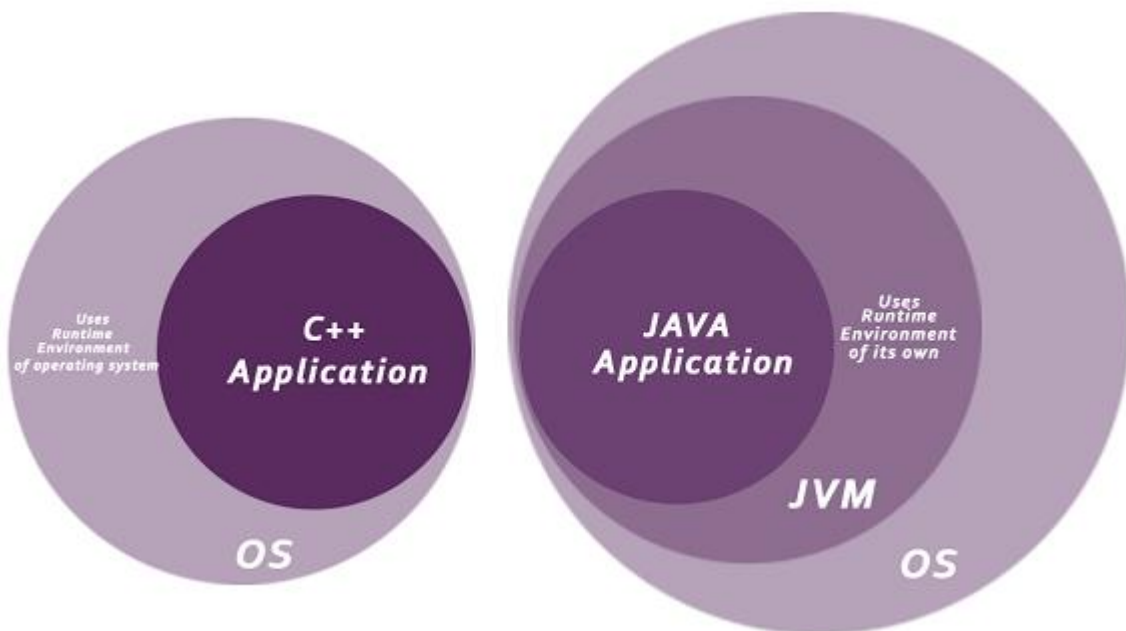
1. Runtime Environment
2. API(Application Programming Interface)

Java code can be executed on multiple platforms, for example, 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 best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- **No explicit pointer**
- **Java Programs run inside a virtual machine sandbox**



- **ClassLoader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access rights to objects.
- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

Robust

The English meaning of Robust is strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.

- Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Architecture-neutral

Java is architecture neutral because there are no implementation dependent features, for example, the 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. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Portable

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to 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.

Dynamic

Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

C++ vs Java

There are many differences and similarities between the [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 used in Windows-based, web-based, and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of the C programming language .	Java was designed and created as an intermediate language between systems and applications programming. It was designed to be easy to use and to have a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance. Multiple inheritance can be achieved by using interfaces in java .

Operator Overloading	C++ supports operator overloading .	Java doesn't support operator overloading.
Pointers	C++ supports pointers . You can write a pointer program in C++.	Java supports pointer internally. However, you cannot write a pointer program in java. It means java has no pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time and JVM executes this bytecode at runtime and platform independent. That is why it is platform-independent.
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 comments.	Java supports documentation comments for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not to override a function.	Java has no virtual keyword. We can override methods by default. In other words, non-virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator. For positive numbers, it works same like >> operator. For negative numbers, it shifts zero at the top for the negative numbers.
Inheritance Tree	C++ always creates a new inheritance tree.	Java always uses a single inheritance tree. All classes are the child of the Object class in Java. Object is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.

Object-oriented	C++ is an object-oriented language. However, in the C language, a single root hierarchy is not possible.	Java is also an object-oriented language. Everything (except fundamental types) is a single root hierarchy as everything inherits from java.lang.Object.
------------------------	--	--

Note

- Java doesn't support default arguments like C++.
- Java does not support header files like C++. Java uses the import keyword to include different classes and methods.

C++ Program Example

File: main.cpp

```

1. #include <iostream>
2. using namespace std;
3. int main() {
4.     cout << "Hello C++ Programming";
5.     return 0;
6. }
```

Output:

```
Hello C++ Programming
```

Java Program Example

File: Simple.java

```

1. class Simple{
2.     public static void main(String args[]){
3.         System.out.println("Hello Java");
4.     }
5. }
```

Output:

```
Hello Java
```

First Java Program | Hello World Example

1. [Software Requirements](#)
2. [Creating Hello Java Example](#)
3. [Resolving javac is not recognized](#)

In this section, 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 the main method. Let's understand the requirement first.

The requirement for Java Hello World Example

For executing any Java program, the following software or application must be properly installed.

- 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 World Example

Let's create the hello java program:

1. **class** Simple{
2. **public static void** main(String args[]){
3. System.out.println("Hello Java");
4. }
5. }

[Test it Now](#)

Save the above file as Simple.java.

To compile:

javac Simple.java

To execute:

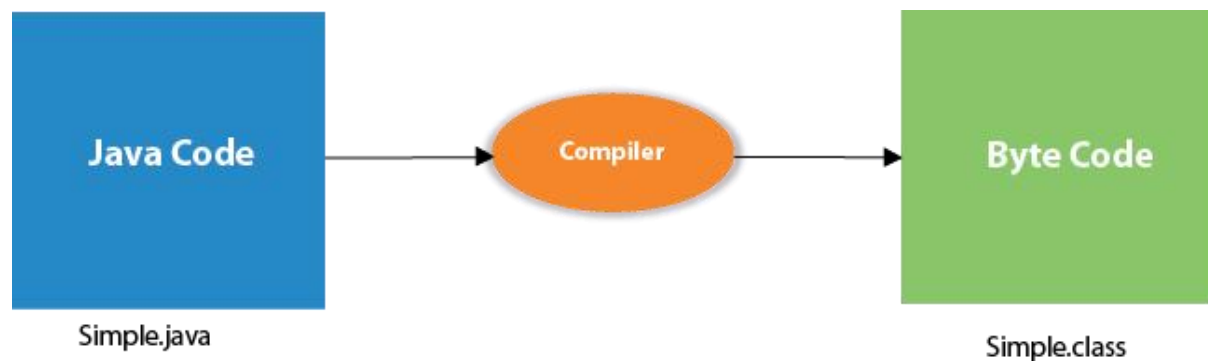
java Simple

Output:

```
Hello Java
```

Compilation Flow:

When we compile Java program using javac tool, the Java compiler converts the source code into byte code.

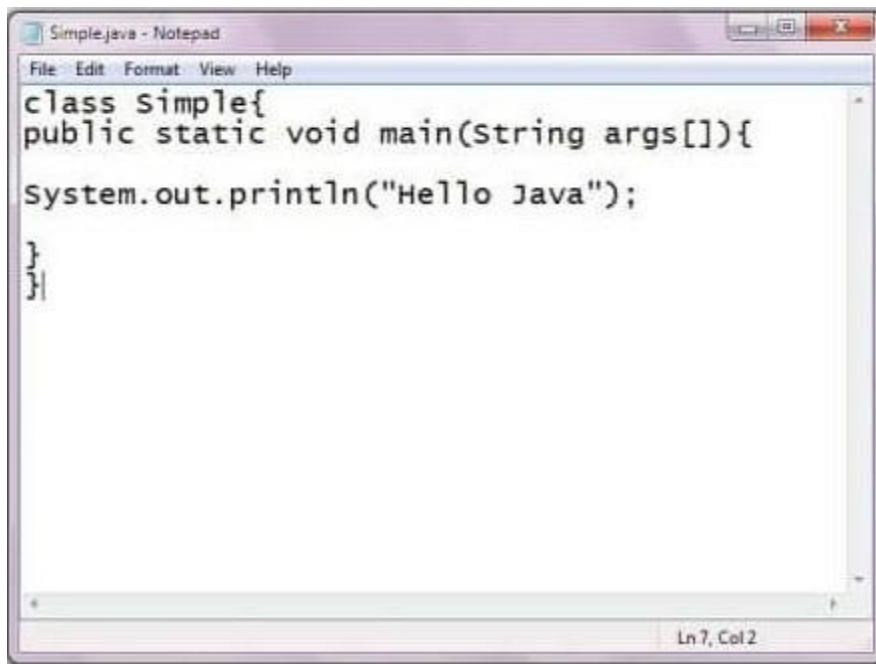


Parameters used in First Java Program

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

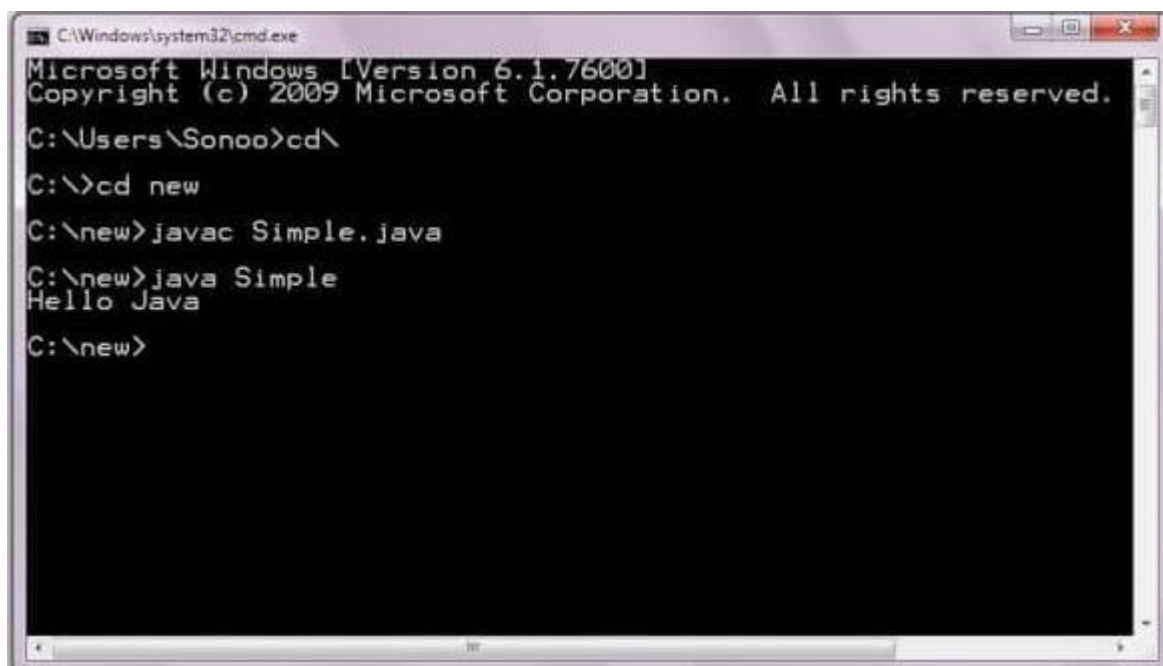
- **class** keyword is used to declare a class in Java.
 - **public** keyword is an access modifier that represents visibility. It means it is visible to all.
 - **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an 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 the starting point of the program.
 - **String[] args** or **String args[]** is used for [command line argument](#). We will discuss it in coming section.
 - **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of [System.out.println\(\)](#) statement in the coming section.
-

To write the simple program, you need to open notepad by **start menu -> All Programs -> Accessories -> Notepad** and write a simple program as we have shown below:



```
Simple.java - Notepad
File Edit Format View Help
class Simple{
public static void main(String args[]){
System.out.println("Hello Java");
}
}
Ln 7, Col 2
```

As displayed in the above diagram, write the simple program of Java in notepad and saved it as Simple.java. In order to compile and run the above program, you need to open the command prompt by **start menu -> All Programs -> Accessories -> command prompt**. When we have done with all the steps properly, it shows the following output:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>cd\
C:\>cd new
C:\new>javac Simple.java
C:\new>java Simple
Hello Java
C:\new>
```

To compile and run the above program, go to your current directory first; my current directory is c:\new. Write here:

To compile: `javac Simple.java`

To execute: `java Simple`

In how many ways we can 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 the sequence of the modifiers, method prototype is not changed in Java.

Let's see the simple code of the main method.

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

2) The subscript notation in the Java array can be used after type, before the variable or after the 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 the main() method by passing 3 ellipses (dots)

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

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

4) Having a semicolon at the end of class is optional in Java.

Let's see the simple code.

1. `class` A{
2. `static public void` main(String... args){

3. `System.out.println("hello java4");`
 4. `}`
 5. `};`
-

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. **public static void** main(String... args)
 5. **static public void** main(String[] args)
 6. **public static final void** main(String[] args)
 7. **final public static void** main(String[] args)
 8. **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)
-

Resolving an error "javac is not recognized as an internal or external command"?

If there occurs a problem like displayed in the below figure, you need to set a path. Since DOS doesn't recognize javac and java as internal or external command. To overcome this problem, we need to set a path. The path is not required in a case where you save your program inside the JDK/bin directory. However, it is an excellent approach to set the path. Click here for [How to set path in java](#).

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>cd\

C:\>cd new

C:\new>javac Simple.java
'javac' is not recognized as an internal or external command,
operable program or batch file.

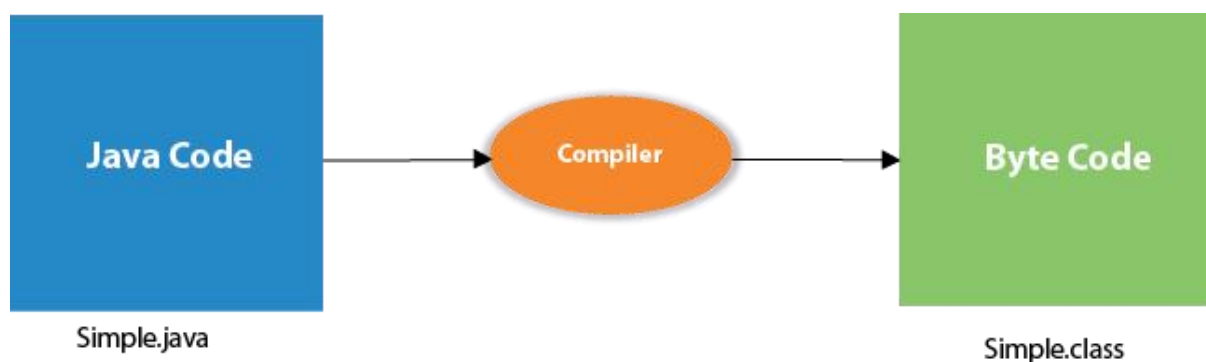
C:\new>■
```

Internal Details of Hello Java Program

In the previous section, we have created Java Hello World program and learn how to compile and run a Java program. In this section, we are going to learn, what happens while we compile and run the Java program. Moreover, we will see some questions based on the first program.

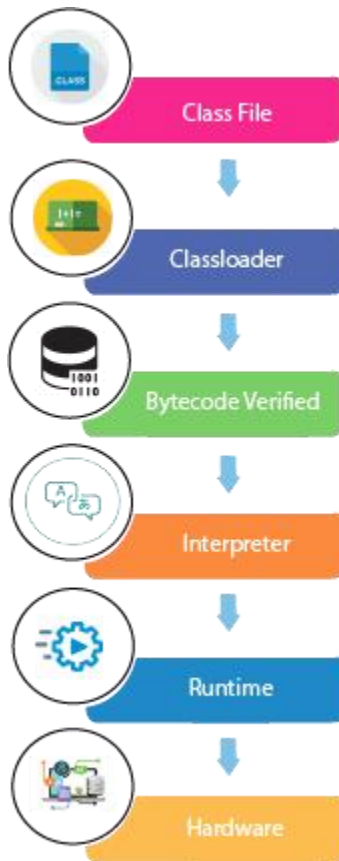
What happens at compile time?

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



What happens at runtime?

At runtime, the following steps are performed:



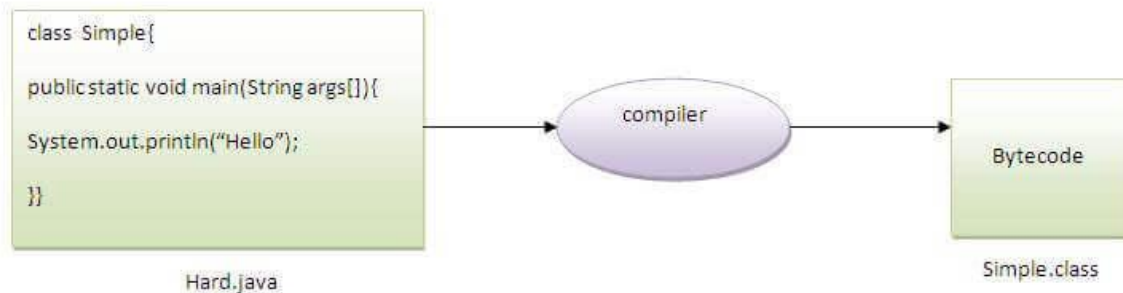
Classloader: It is the subsystem of JVM that is used to load class files.

Bytecode Verifier: Checks the code fragments for illegal code that can violate access rights to objects.

Interpreter: Read bytecode stream then execute the instructions.

Q) Can you save a Java source file by another 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

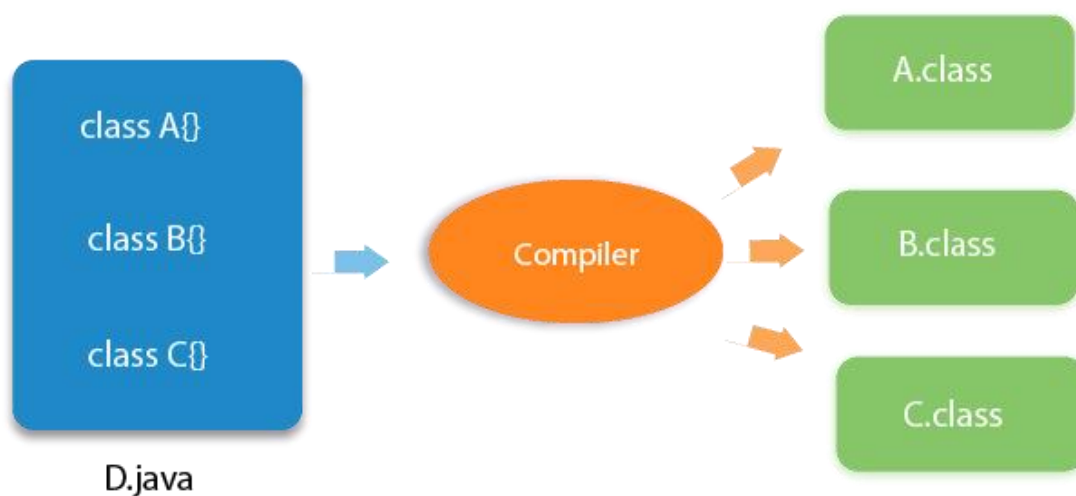
To execute:

java Simple

Observe that, we have compiled the code with file name but running the program with class name. Therefore, we can save a Java program other than class name.

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

Yes, like the figure given below illustrates:



How to set path in Java

1. [How to set the path of JDK in Windows OS](#)
 1. [Setting Temporary Path of JDK](#)
 2. [Setting Permanent Path of JDK](#)
2. [How to set the path of JDK in Linux OS](#)

The path is required to be set for using tools such as javac, java, etc.

If you are saving the Java source file inside the JDK/bin directory, the path is not required to be set because all the tools will be available in the current directory.

However, if you have your Java file outside the JDK/bin folder, it is necessary to set the path of JDK.

There are two ways to set the path in Java:

1. Temporary
2. Permanent

1) How to set the Temporary Path of JDK in Windows

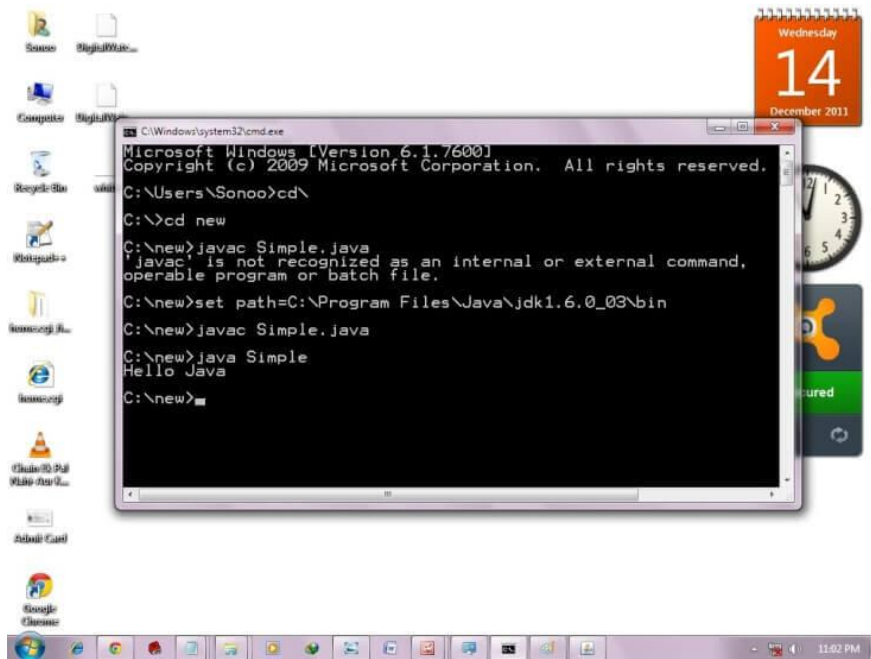
To set the temporary path of JDK, you need to follow the following steps:

- Open the command prompt
- Copy the path of the JDK/bin directory
- Write in command prompt: set path=copied_path

For Example:

```
set path=C:\Program Files\Java\jdk1.6.0_23\bin
```

Let's see it in the figure given below:



2) How to set Permanent Path of JDK in Windows

For setting the permanent path of JDK, you need to follow these steps:

- Go to MyComputer properties -> advanced tab -> environment variables -> new tab of user variable -> write path in variable name -> write path of bin folder in variable value -> ok -> ok -> ok

For Example:

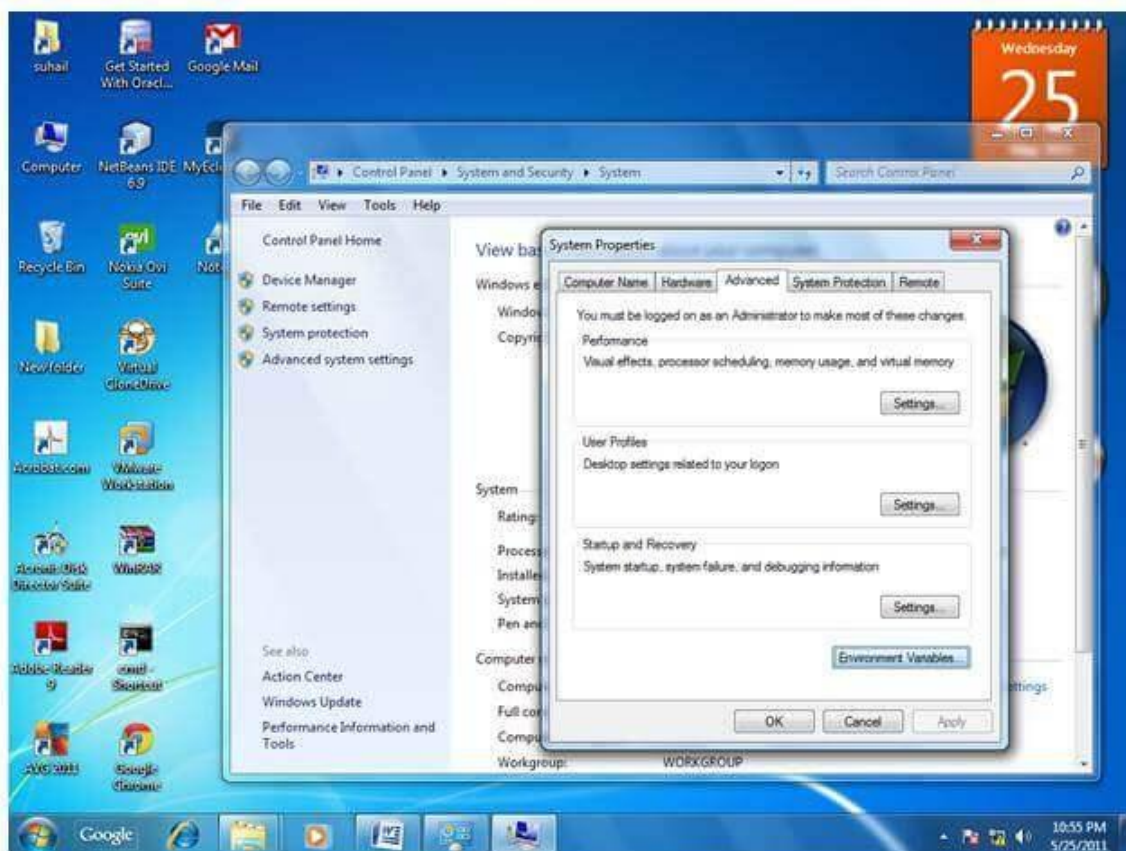
1) Go to MyComputer properties



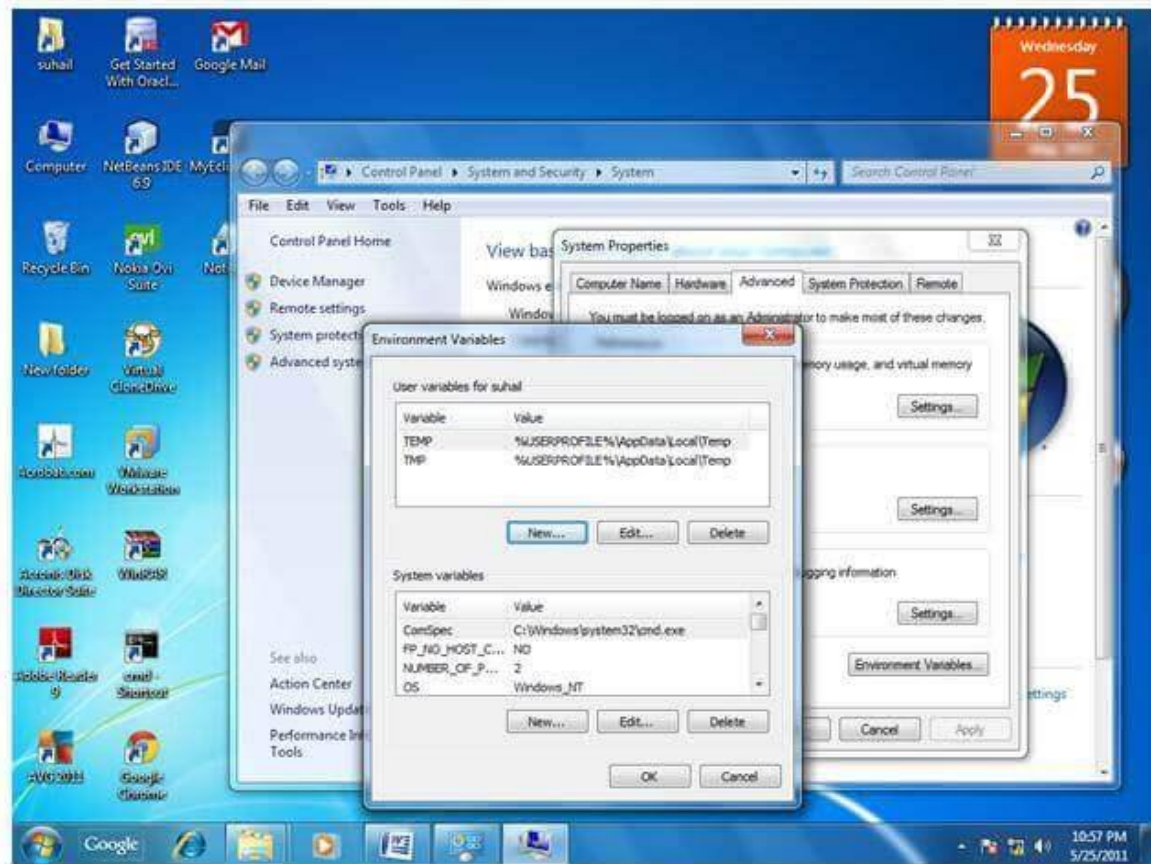
2) Click on the advanced tab



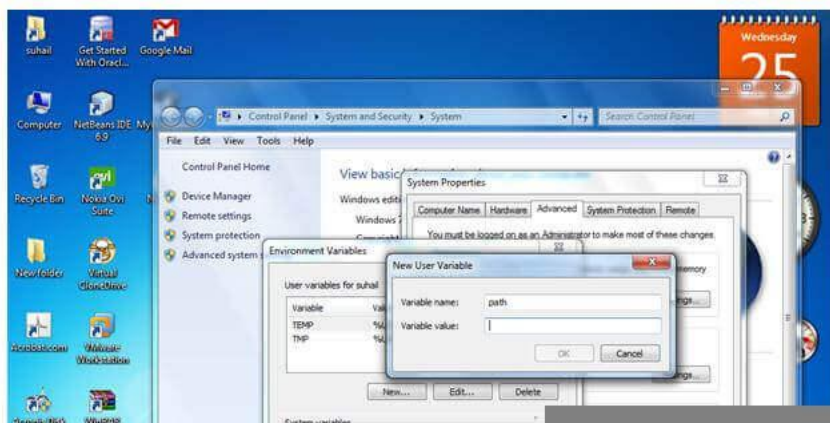
3) Click on environment variables



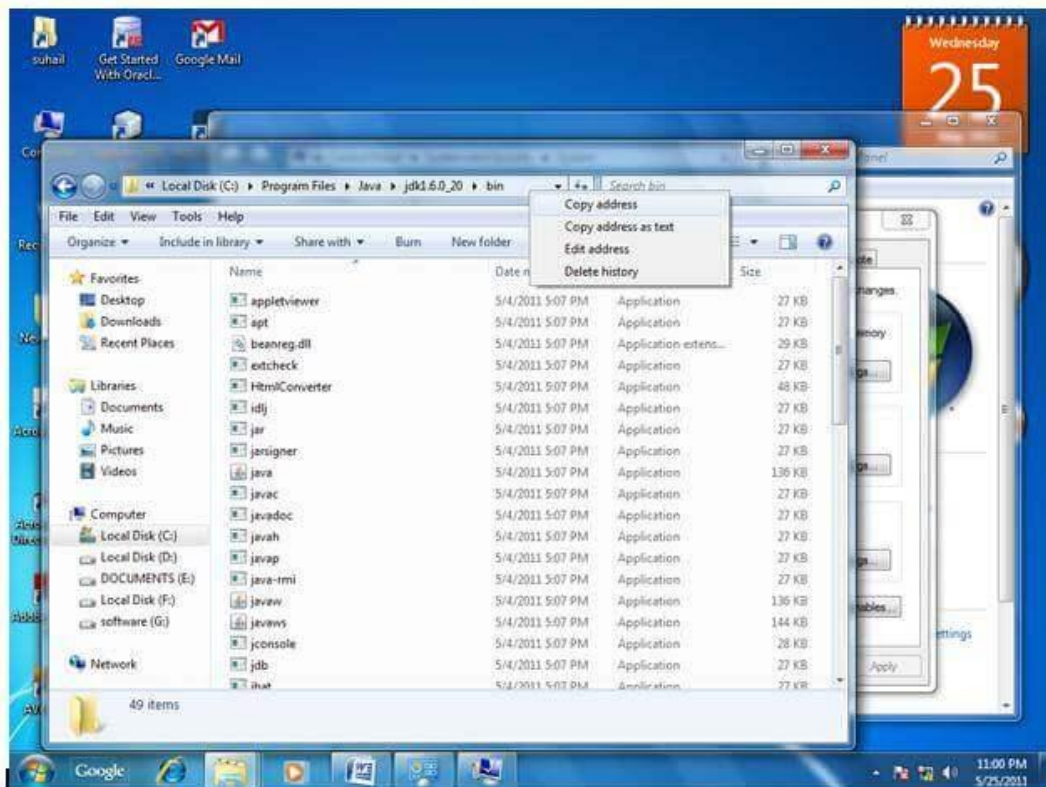
4) Click on the new tab of user variables



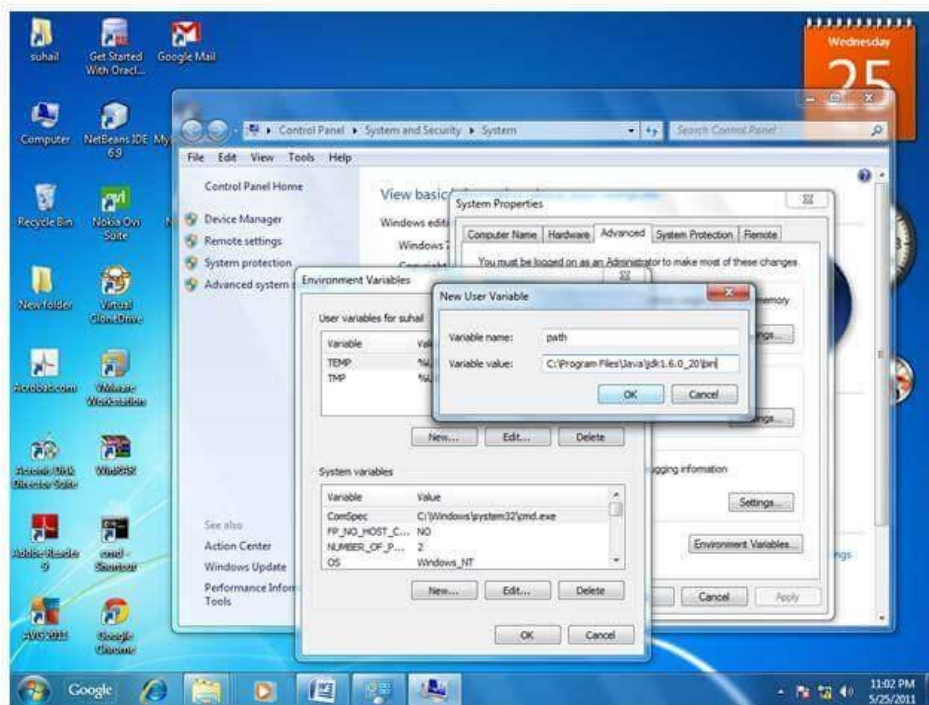
5) Write the path in the variable name



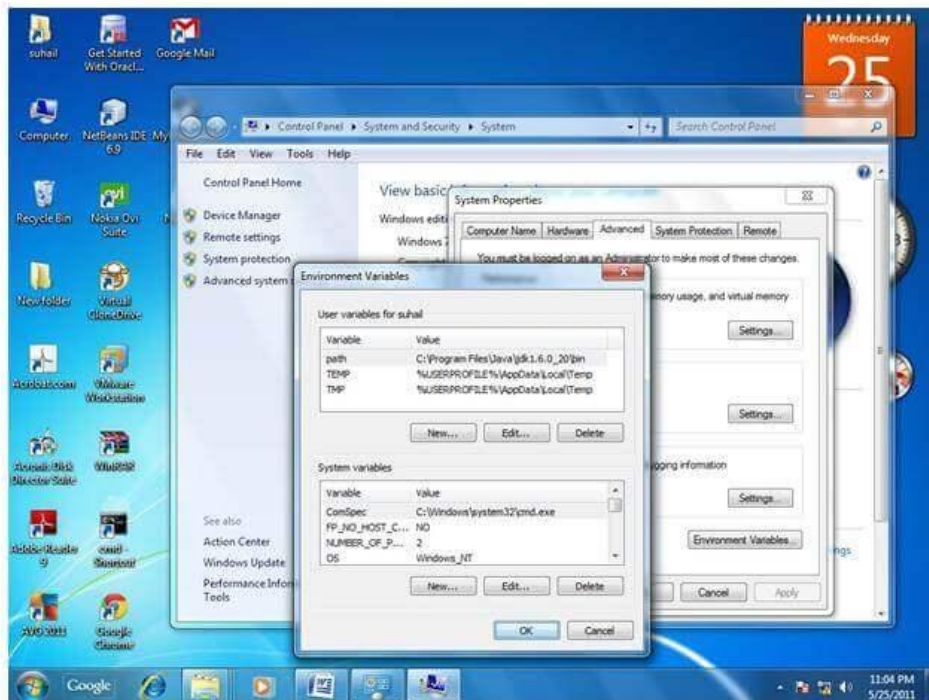
6) Copy the path of bin folder



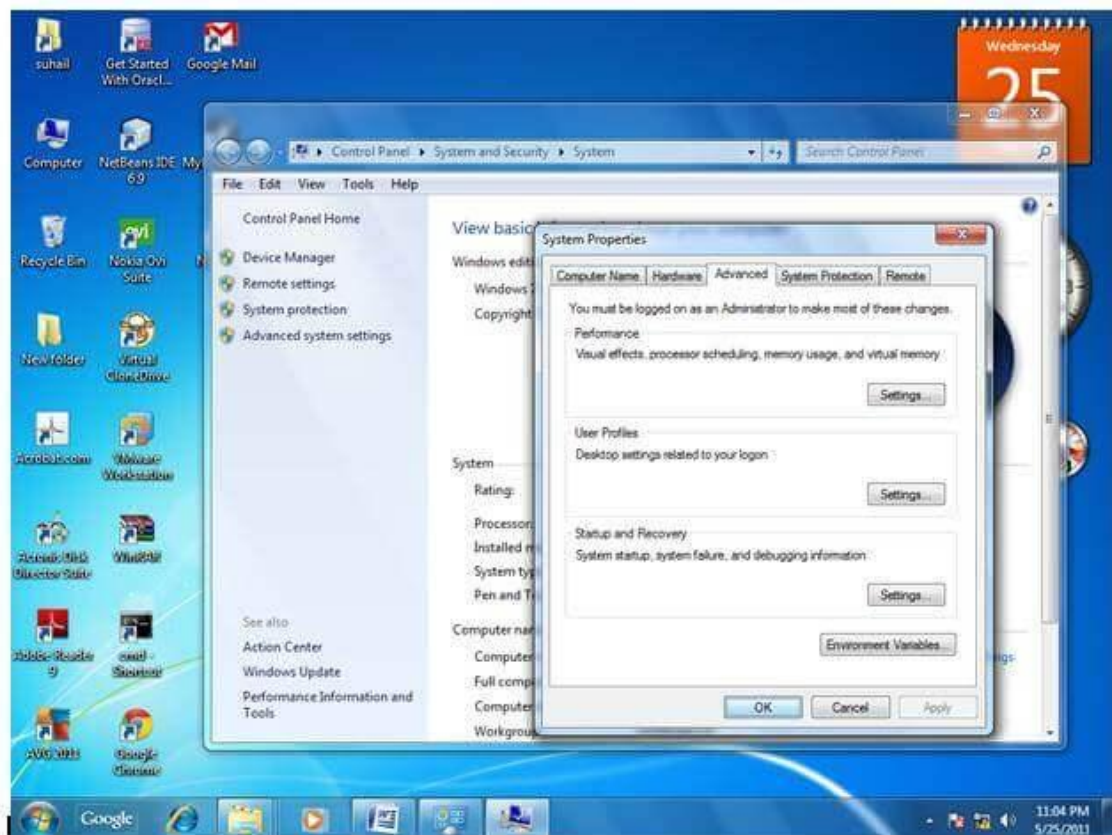
7) Paste path of bin folder in the variable value



8) Click on ok button



9) Click on ok button



Now your permanent path is set. You can now execute any program of java from any drive.

Setting Java Path in Linux OS

Setting path in Linux OS is the same as setting the path in the Windows OS. But, here we use the export tool rather than set. Let's see how to set path in Linux OS:

```
export PATH=$PATH:/home/jdk1.6.01/bin/
```

Difference between JDK, JRE, and JVM

1. [A summary of JVM](#)
2. [Java Runtime Environment \(JRE\)](#)
3. [Java Development Kit \(JDK\)](#)

We must understand the differences between JDK, JRE, and JVM before proceeding further to [Java](#). See the brief overview of JVM [here](#).

If you want to get the detailed knowledge of Java Virtual Machine, move to the next page. Firstly, let's see the differences between the JDK, JRE, and JVM.

JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each [OS](#) is different from each other. However, Java is platform independent. There are three notions of the JVM: *specification*, *implementation*, and *instance*.

The JVM performs the following main tasks:

- Loads code
- Verifies code

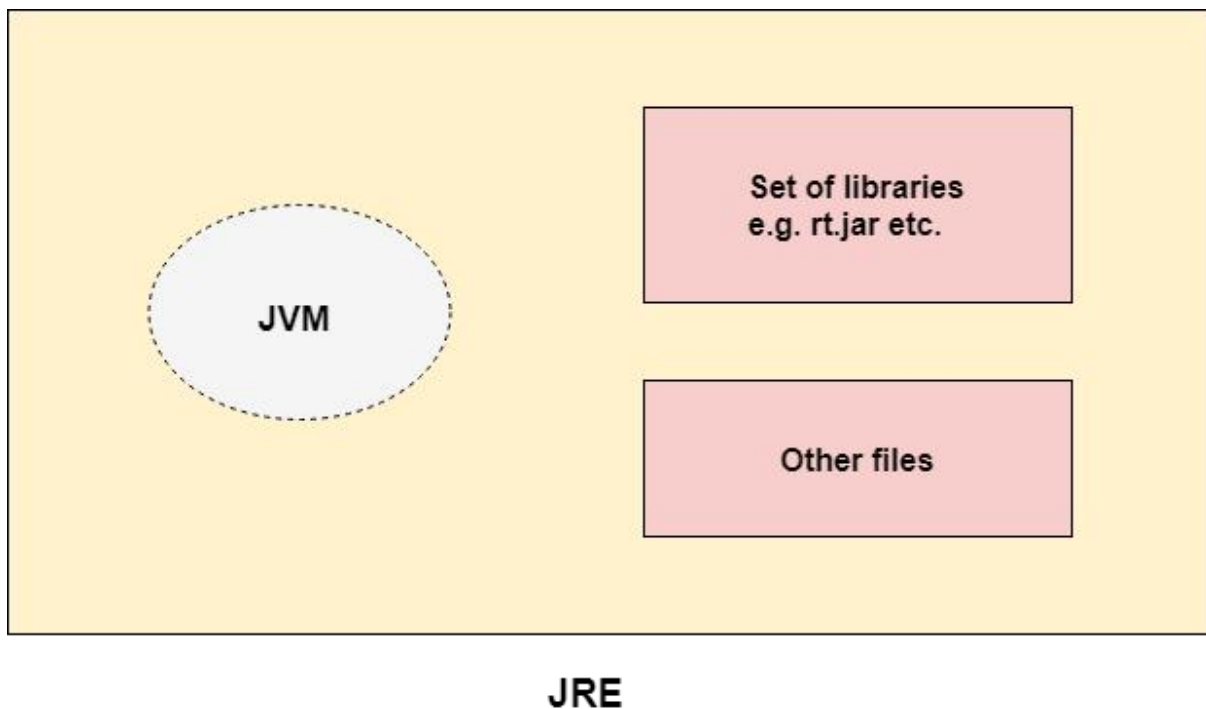
- Executes code
- Provides runtime environment

[More Details.](#)

JRE

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

The implementation of JVM is also actively released by other companies besides Sun Micro Systems.



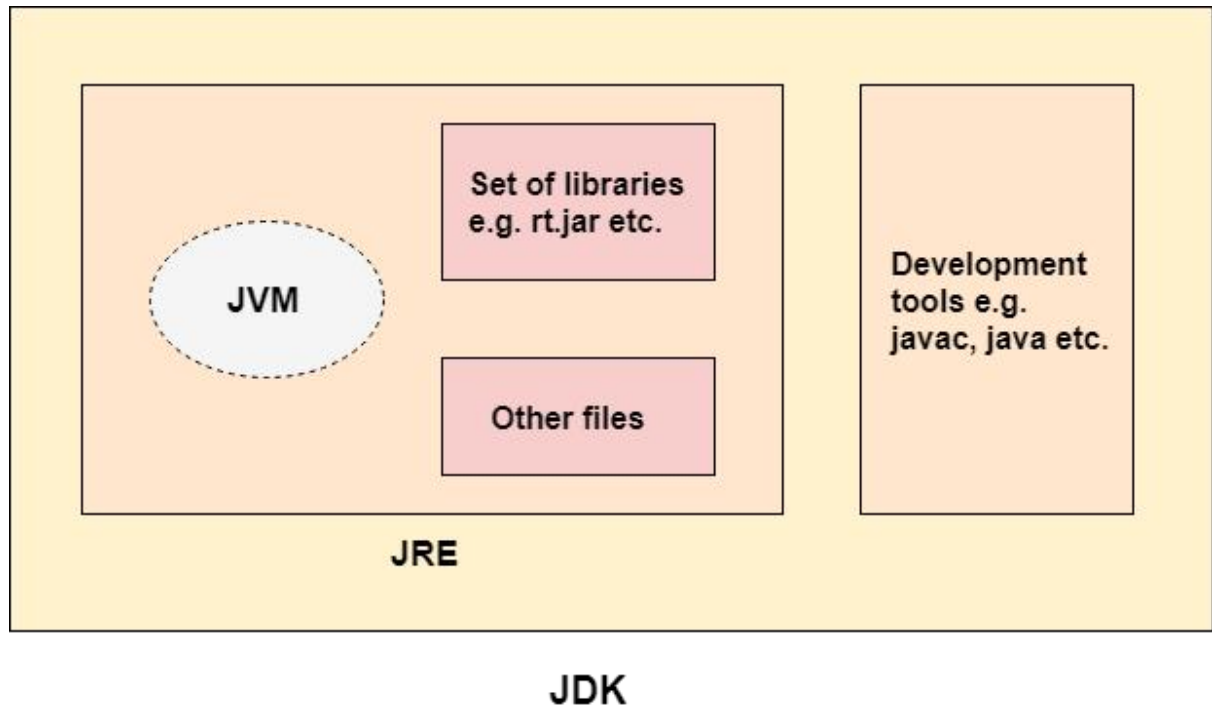
JDK

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and [applets](#). It physically exists. It contains JRE + development tools.

JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



JVM (Java Virtual Machine) Architecture

1. [Java Virtual Machine](#)
2. [Internal Architecture of 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 (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 Oracle 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:

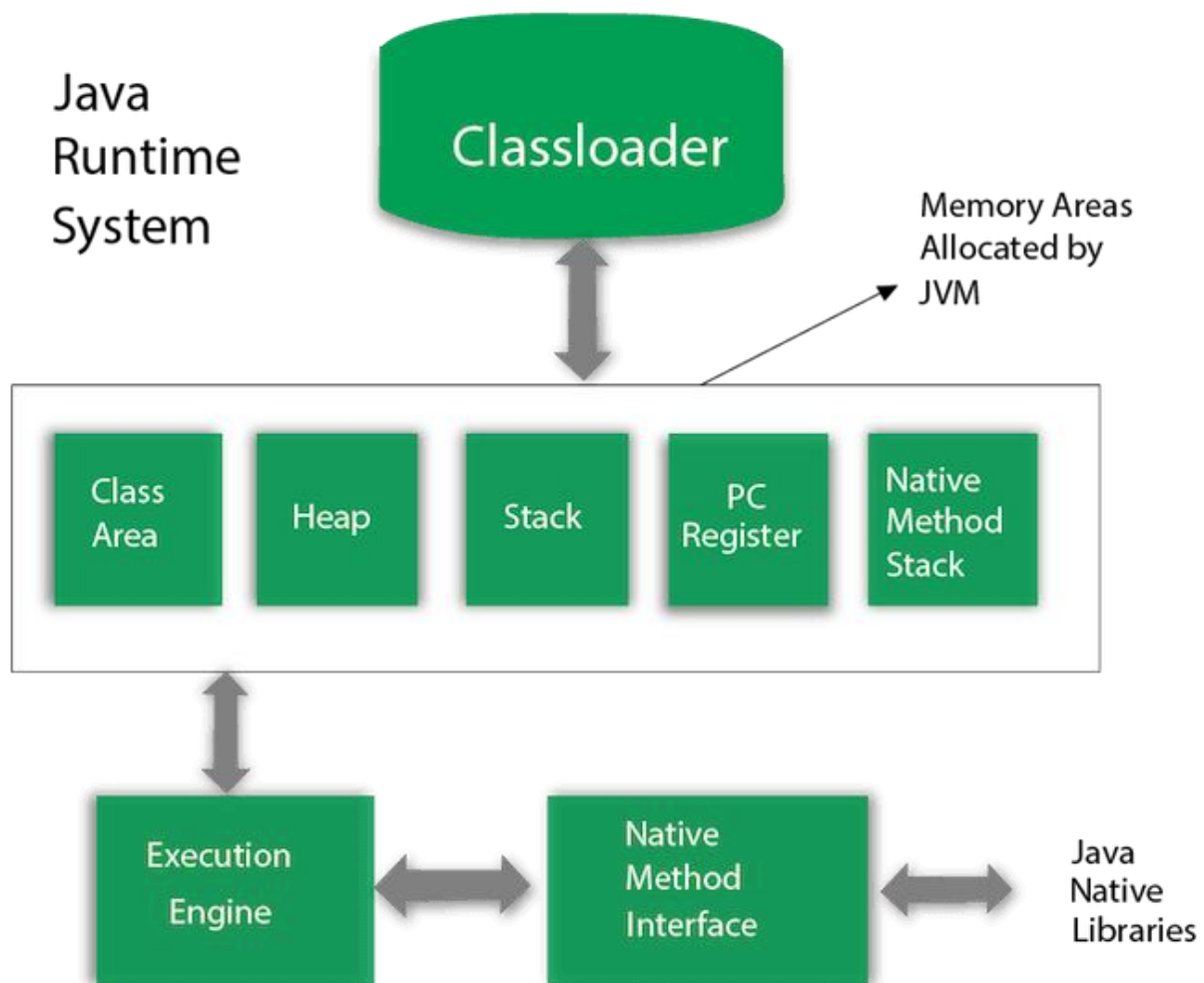
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

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

JVM Architecture

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



1) Classloader

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

1. **Bootstrap ClassLoader**: This is the first classloader which is the super class of Extension classloader. It loads the *rt.jar* file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.
2. **Extension ClassLoader**: This is the child classloader of Bootstrap and parent classloader of System classloader. It loads the jar files located inside `$JAVA_HOME/jre/lib/ext` directory.
3. **System/Application ClassLoader**: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current

directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

```
1. //Let's see an example to print the classloader name
2. public class ClassLoaderExample
3. {
4.     public static void main(String[] args)
5.     {
6.         // Let's print the classloader name of current class.
7.         //Application/System classloader will load this class
8.         Class c=ClassLoaderExample.class;
9.         System.out.println(c.getClassLoader());
10.        //If we print the classloader name of String, it will print null because it is an
11.                                                //in-
        built class which is found in rt.jar, so it is loaded by Bootstrap classloader
12.        System.out.println(String.class.getClassLoader());
13.    }
14. }
```

Test it Now

Output:

```
sun.misc.Launcher$AppClassLoader@4e0e2f2a
null
```

These are the internal classloaders provided by Java. If you want to create your own classloader, you need to extend the ClassLoader class.

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 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.

8) Java Native Interface

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

Java Variables

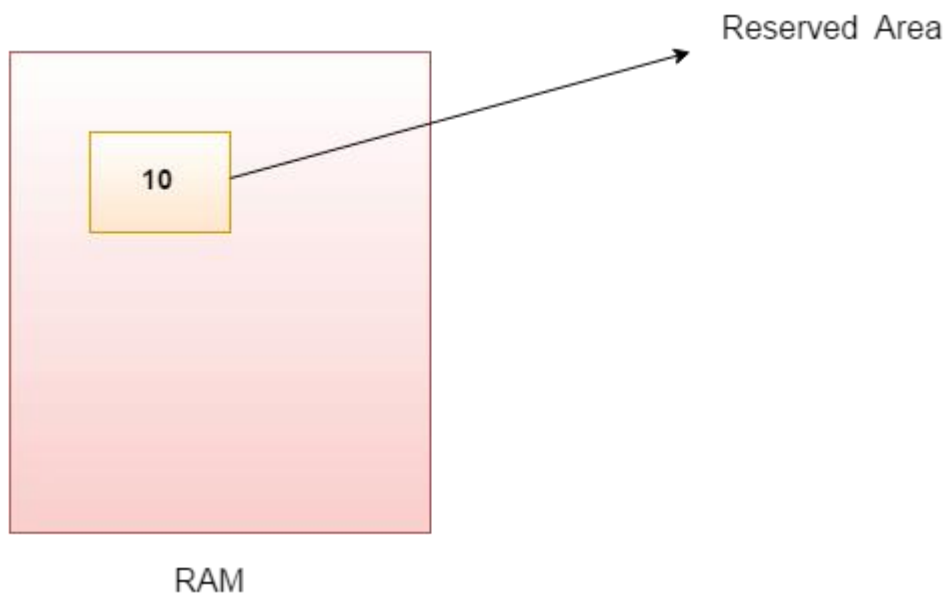
A variable is a container which holds the value while the [Java program](#) is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of [data types in Java](#): primitive and non-primitive.

Variable

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



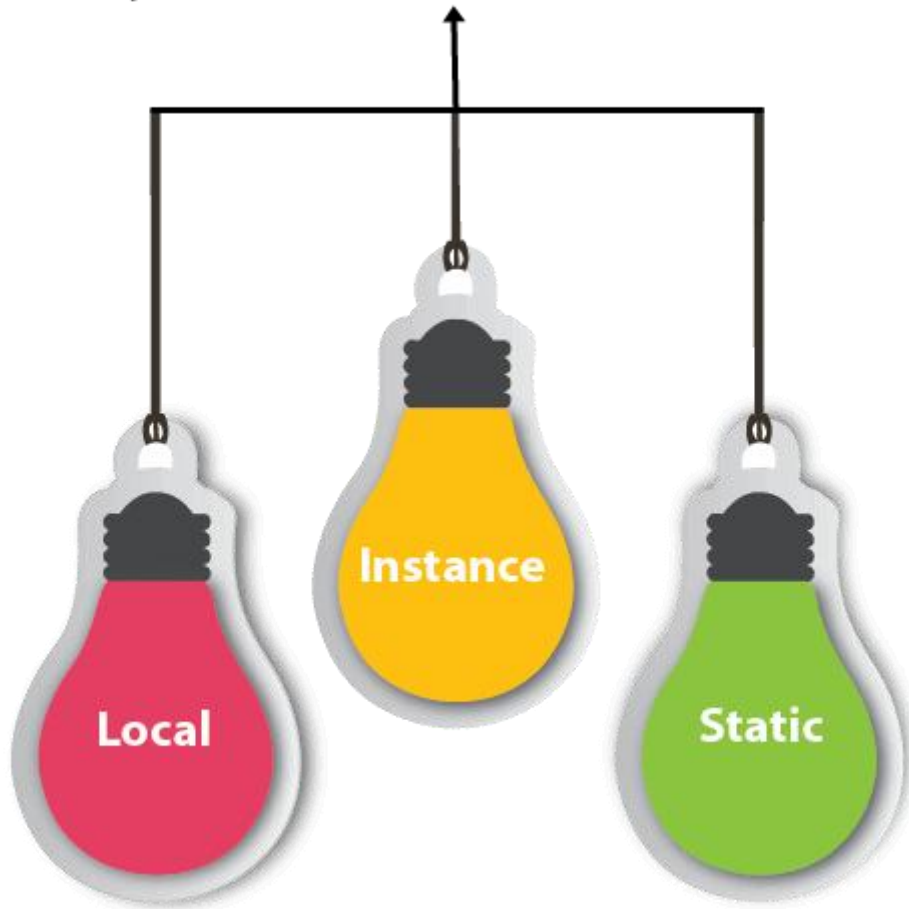
1. `int data=50;`//Here data is variable

Types of Variables

There are three types of variables in [Java](#):

- local variable
- instance variable
- static variable

Types of Variables



1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

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

It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Example to understand the types of variables in java

```
1. public class A
2. {
3.     static int m=100;//static variable
4.     void method()
5.     {
6.         int n=90;//local variable
7.     }
8.     public static void main(String args[])
9.     {
10.        int data=50;//instance variable
11.    }
12. }//end of class
```

Java Variable Example: Add Two Numbers

```
1. public class Simple{
2.     public static void main(String[] args){
3.         int a=10;
4.         int b=10;
5.         int c=a+b;
6.         System.out.println(c);
7.     }
8. }
```

Output:

20

Java Variable Example: Widening

```
1. public class Simple{
2.     public static void main(String[] args){
3.         int a=10;
4.         float f=a;
5.         System.out.println(a);
6.         System.out.println(f);
```

```
7.  }}
```

Output:

```
10
10.0
```

Java Variable Example: Narrowing (Typecasting)

```
1. public class Simple{
2.     public static void main(String[] args){
3.         float f=10.5f;
4.         //int a=f;//Compile time error
5.         int a=(int)f;
6.         System.out.println(f);
7.         System.out.println(a);
8.     }}
```

Output:

```
10.5
10
```

Java Variable Example: Overflow

```
1. class Simple{
2.     public static void main(String[] args){
3.         //Overflow
4.         int a=130;
5.         byte b=(byte)a;
6.         System.out.println(a);
7.         System.out.println(b);
8.     }}
```

Output:

```
130
-126
```

Java Variable Example: Adding Lower Type

```
1. class Simple{
2.     public static void main(String[] args){
3.         byte a=10;
4.         byte b=10;
5.         //byte c=a+b;//Compile Time Error: because a+b=20 will be int
```

6. `byte c=(byte)(a+b);`
7. `System.out.println(c);`
8. `}}`

Output:

20

Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include [Classes](#), [Interfaces](#), and [Arrays](#).

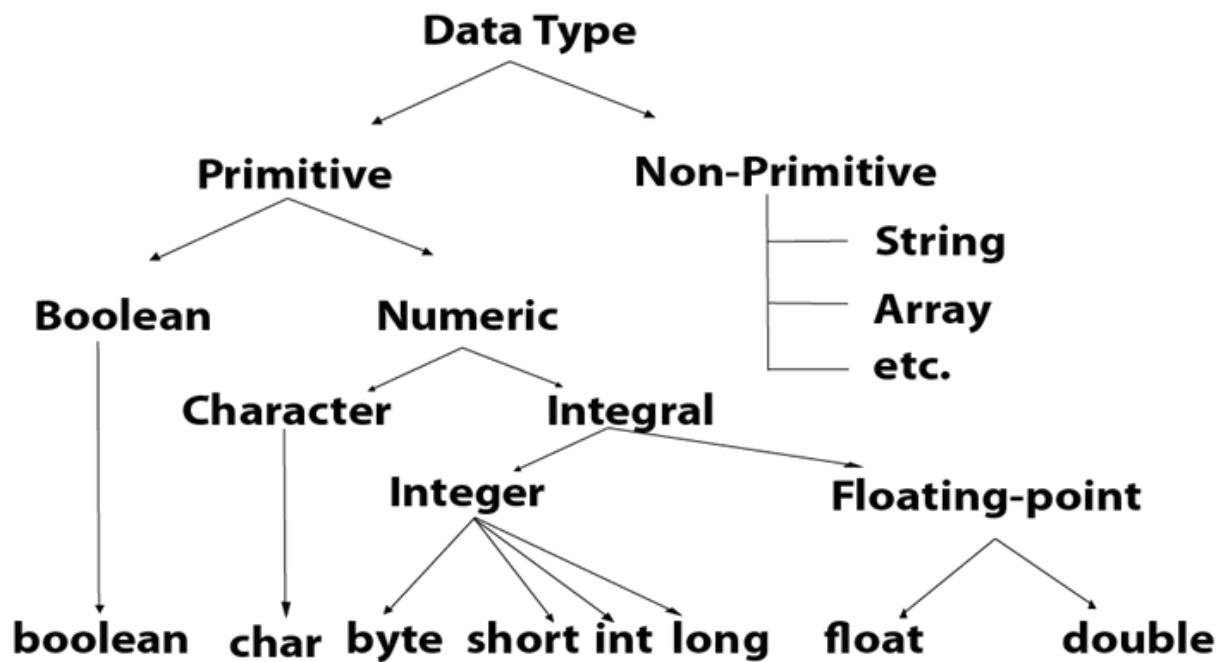
Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in [Java language](#).

Java is a statically-typed programming language. It means, all [variables](#) must be declared before its use. That is why we need to declare variable's type and name.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type



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

Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example:

1. Boolean one = **false**

Byte Data Type

The byte data type is an example of primitive data type. It is an 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example:

1. **byte** a = **10**, **byte** b = **-20**

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example:

1. **short** s = **10000**, **short** r = **-5000**

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^{31}) to 2,147,483,647 ($2^{31} - 1$) (inclusive). Its minimum value is - 2,147,483,648 and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example:

1. **int** a = 100000, **int** b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between $-9,223,372,036,854,775,808(-2^{63})$ to $9,223,372,036,854,775,807(2^{63} - 1)$ (inclusive). Its minimum value is $-9,223,372,036,854,775,808$ and maximum value is $9,223,372,036,854,775,807$. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example:

1. **long** a = 100000L, **long** b = -200000L

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point. Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example:

1. **float** f1 = 234.5f

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example:

1. **double** d1 = 12.3

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive). The char data type is used to store characters.

Example:

1. `char` letterA = 'A'

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

It is because java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system. To get detail explanation about Unicode visit next page.

Unicode System

Unicode is a universal international standard character encoding that is capable of representing most 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 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

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	<code>expr++ expr--</code>
	prefix	<code>++expr --expr +expr -expr ~ !</code>
Arithmetic	multiplicative	<code>* / %</code>
	additive	<code>+ -</code>
Shift	shift	<code><< >> >>></code>
Relational	comparison	<code>< > <= >= instanceof</code>
	equality	<code>== !=</code>
Bitwise	bitwise AND	<code>&</code>

	bitwise exclusive OR	^
	bitwise inclusive OR	
Logical	logical AND	& &
	logical OR	
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= = <<= >>= >>>=

Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Java Unary Operator Example: ++ and --

1. **public class** OperatorExample{
2. **public static void** main(String args[]){
3. **int** x=10;
4. System.out.println(x++);**//10 (11)**
5. System.out.println(++x);**//12**
6. System.out.println(x--);**//12 (11)**
7. System.out.println(--x);**//10**
8. }}

Output:

```
10
12
12
10
```

Java Unary Operator Example 2: ++ and --

1. **public class** OperatorExample{
2. **public static void** main(String args[]){

3. **int** a=10;
4. **int** b=10;
5. System.out.println(a++ + ++a);//10+12=22
6. System.out.println(b++ + b++);//10+11=21
- 7.
8. }}

Output:

```
22
21
```

Java Unary Operator Example: ~ and !

1. **public class** OperatorExample{
2. **public static void** main(String args[]){
3. **int** a=10;
4. **int** b=-10;
5. **boolean** c=true;
6. **boolean** d=false;
7. System.out.println(~a);//-
11 (minus of total positive value which starts from 0)
8. System.out.println(~b);//9 (positive of total minus, positive starts from 0)
9. System.out.println(!c);//false (opposite of boolean value)
10. System.out.println(!d);//true
11. }}

Output:

```
-11
9
false
true
```

Java Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

Java Arithmetic Operator Example

1. **public class** OperatorExample{
2. **public static void** main(String args[]){
3. **int** a=10;
4. **int** b=5;

5. `System.out.println(a+b);`//15
6. `System.out.println(a-b);`//5
7. `System.out.println(a*b);`//50
8. `System.out.println(a/b);`//2
9. `System.out.println(a%b);`//0
10. `}}`

Output:

```
15
5
50
2
0
```

Java Arithmetic Operator Example: Expression

1. `public class` OperatorExample{
2. `public static void` main(String args[]){
3. `System.out.println(10*10/5+3-1*4/2);`
4. `}}`

Output:

```
21
```

Java Left Shift Operator

The Java left shift operator `<<` is used to shift all of the bits in a value to the left side of a specified number of times.

Java Left Shift Operator Example

1. `public class` OperatorExample{
2. `public static void` main(String args[]){
3. `System.out.println(10<<2);`//10*2^2=10*4=40
4. `System.out.println(10<<3);`//10*2^3=10*8=80
5. `System.out.println(20<<2);`//20*2^2=20*4=80
6. `System.out.println(15<<4);`//15*2^4=15*16=240
7. `}}`

Output:

```
40
80
80
240
```


Java Right Shift Operator

The Java right shift operator `>>` is used to move the value of the left operand to right by the number of bits specified by the right operand.

Java Right Shift Operator Example

1. **public** OperatorExample{
2. **public static void** main(String args[]){
3. System.out.println(**10**`>>2`);*//10/2^2=10/4=2*
4. System.out.println(**20**`>>2`);*//20/2^2=20/4=5*
5. System.out.println(**20**`>>3`);*//20/2^3=20/8=2*
6. }}

Output:

```
2
5
2
```

Java Shift Operator Example: `>>` vs `>>>`

1. **public class** OperatorExample{
2. **public static void** main(String args[]){
3. *//For positive number, >> and >>> works same*
4. System.out.println(**20**`>>2`);
5. System.out.println(**20**`>>>2`);
6. *//For negative number, >>> changes parity bit (MSB) to 0*
7. System.out.println(**-20**`>>2`);
8. System.out.println(**-20**`>>>2`);
9. }}

Output:

```
5
5
-5
1073741819
```

Java AND Operator Example: Logical `&&` and Bitwise `&`

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

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

1. **public class** OperatorExample{
2. **public static void** main(String args[]){
3. **int** a=10;
4. **int** b=5;
5. **int** c=20;
6. System.out.println(a<b&&a<c);//false && true = false
7. System.out.println(a<b&a<c);//false & true = false
8. }}

Output:

```
false
false
```

Java AND Operator Example: Logical && vs Bitwise &

1. **public class** OperatorExample{
2. **public static void** main(String args[]){
3. **int** a=10;
4. **int** b=5;
5. **int** c=20;
6. System.out.println(a<b&&a++<c);//false && true = false
7. System.out.println(a);//10 because second condition is not checked
8. System.out.println(a<b&a++<c);//false && true = false
9. System.out.println(a);//11 because second condition is checked
10. }}

Output:

```
false
10
false
11
```

Java OR Operator Example: Logical || and Bitwise |

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

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

1. **public class** OperatorExample{
2. **public static void** main(String args[]){
3. **int** a=10;

4. **int** b=5;
5. **int** c=20;
6. System.out.println(a>b||a<c);//true || true = true
7. System.out.println(a>b|a<c);//true | true = true
8. **//|| vs |**
9. System.out.println(a>b||a++<c);//true || true = true
10. System.out.println(a);//10 because second condition is not checked
11. System.out.println(a>b|a++<c);//true | true = true
12. System.out.println(a);//11 because second condition is checked
13. }}

Output:

```
true
true
true
10
true
11
```

Java Ternary Operator

Java Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

Java Ternary Operator Example

1. **public class** OperatorExample{
2. **public static void** main(String args[]){
3. **int** a=2;
4. **int** b=5;
5. **int** min=(a<b)?a:b;
6. System.out.println(min);
7. }}

Output:

```
2
```

Another Example:

1. **public class** OperatorExample{
2. **public static void** main(String args[]){

3. **int** a=10;
4. **int** b=5;
5. **int** min=(a<b)?a:b;
6. System.out.println(min);
7. }}

Output:

5

Java Assignment Operator

Java assignment operator is one of the most common operators. It is used to assign the value on its right to the operand on its left.

Java Assignment Operator Example

1. **public class** OperatorExample{
2. **public static void** main(String args[]){
3. **int** a=10;
4. **int** b=20;
5. a+=4;//a=a+4 (a=10+4)
6. b-=4;//b=b-4 (b=20-4)
7. System.out.println(a);
8. System.out.println(b);
9. }}

Output:

14

16

Java Assignment Operator Example

1. **public class** OperatorExample{
2. **public static void** main(String[] args){
3. **int** a=10;
4. a+=3;//10+3
5. System.out.println(a);
6. a-=4;//13-4
7. System.out.println(a);
8. a*=2;//9*2
9. System.out.println(a);
10. a/=2;//18/2

```
11. System.out.println(a);
12. }}
```

Output:

```
13
9
18
9
```

Java Assignment Operator Example: Adding short

```
1. public class OperatorExample{
2.     public static void main(String args[]){
3.         short a=10;
4.         short b=10;
5.         //a+=b;//a=a+b internally so fine
6.         a=a+b;//Compile time error because 10+10=20 now int
7.         System.out.println(a);
8.     }}
```

Output:

```
Compile time error
```

After type cast:

```
1. public class OperatorExample{
2.     public static void main(String args[]){
3.         short a=10;
4.         short b=10;
5.         a=(short)(a+b);//20 which is int now converted to short
6.         System.out.println(a);
7.     }}
```

Output:

```
20
```

Java Keywords

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.

List of Java Keywords

A list of Java keywords or reserved words are given below:

1. **abstract**: Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.
2. **boolean**: Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
3. **break**: Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.
4. **byte**: Java byte keyword is used to declare a variable that can hold 8-bit data values.
5. **case**: Java case keyword is used with the switch statements to mark blocks of text.
6. **catch**: Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
7. **char**: Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
8. **class**: Java class keyword is used to declare a class.
9. **continue**: Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
10. **default**: Java default keyword is used to specify the default block of code in a switch statement.
11. **do**: Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.
12. **double**: Java double keyword is used to declare a variable that can hold 64-bit floating-point number.
13. **else**: Java else keyword is used to indicate the alternative branches in an if statement.
14. **enum**: Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
15. **extends**: Java extends keyword is used to indicate that a class is derived from another class or interface.

16. **final:** Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.
17. **finally:** Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.
18. **float:** Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
19. **for:** Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use for loop.
20. **if:** Java if keyword tests the condition. It executes the if block if the condition is true.
21. **implements:** Java implements keyword is used to implement an interface.
22. **import:** Java import keyword makes classes and interfaces available and accessible to the current source code.
23. **instanceof:** Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
24. **int:** Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
25. **interface:** Java interface keyword is used to declare an interface. It can have only abstract methods.
26. **long:** Java long keyword is used to declare a variable that can hold a 64-bit integer.
27. **native:** Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
28. **new:** Java new keyword is used to create new objects.
29. **null:** Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
30. **package:** Java package keyword is used to declare a Java package that includes the classes.
31. **private:** Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.

32. **protected**: Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.
33. **public**: Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
34. **return**: Java return keyword is used to return from a method when its execution is complete.
35. **short**: Java short keyword is used to declare a variable that can hold a 16-bit integer.
36. **static**: Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.
37. **strictfp**: Java strictfp is used to restrict the floating-point calculations to ensure portability.
38. **super**: Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.
39. **switch**: The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
40. **synchronized**: Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
41. **this**: Java this keyword can be used to refer the current object in a method or constructor.
42. **throw**: The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.
43. **throws**: The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.
44. **transient**: Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
45. **try**: Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.

46. **void:** Java void keyword is used to specify that a method does not have a return value.
47. **volatile:** Java volatile keyword is used to indicate that a variable may change asynchronously.
48. **while:** Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use the while loop.