Java Course…

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**

**class** Simple{

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

     System.out.println("Hello Java");

    }

}

## 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](https://www.javatpoint.com/servlet-tutorial), [JSP](https://www.javatpoint.com/jsp-tutorial), [Struts](https://www.javatpoint.com/struts-2-tutorial), [Spring](https://www.javatpoint.com/spring-tutorial), [Hibernate](https://www.javatpoint.com/hibernate-tutorial), [JSF](https://www.javatpoint.com/jsf-tutorial), 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](https://www.javatpoint.com/ejb-tutorial) 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](https://www.javatpoint.com/java-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](https://www.javatpoint.com/jpa-tutorial), etc.

#### **3) Java ME (Java Micro Edition)**

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

### **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.

Java SE 8 (March 18, 2014)

Its codename was **Spider**. Although, codenames have been discontinued, but the codename **Spider** is common among java developers.

It includes some features which were proposed for Java SE 7 but added in Java SE 8.

* Language-level support for Lambda expressions.
* Allowed developers to embed JavaScript code within applications.
* Annotation of Java Types.
* Provided Date and Time API.
* Repeating Annotations.
* Launching of JavaFX applications.
* Removal of permanent generation.

Java SE 8 is not supported in Windows XP but after JDK 8 update 25, we can install and run it under Windows XP.

Java 8 is set as a default version to download from java.com from October 2014.

Features of Java

The primary objective of [Java programming](https://www.javatpoint.com/java-tutorial) 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.



### **Simple**

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, 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](https://www.javatpoint.com/java-oops-concepts) 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](https://www.javatpoint.com/object-and-class-in-java)
2. [Class](https://www.javatpoint.com/object-and-class-in-java#class)
3. [Inheritance](https://www.javatpoint.com/inheritance-in-java)
4. [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
5. [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
6. [Encapsulation](https://www.javatpoint.com/encapsulation)

### **Portable**

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

### **Platform Independent**



Java is platform independent because it is different from other languages like [C](https://www.javatpoint.com/c-programming-language-tutorial), [C++](https://www.javatpoint.com/cpp-tutorial), 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 mining 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.

### **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.

### **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.

### **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).

### **Creating Hello World Example**

Let's create the hello java program:

**public class** Simple{

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

     System.out.println("Hello Java");

    }

}

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

* **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](https://www.javatpoint.com/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()](https://www.javatpoint.com/system-out-println-in-java) statement in the coming section.

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.

Difference between JDK, JRE, and JVM

1. [Java Virtual Machine (JVM)](https://www.javatpoint.com/difference-between-jdk-jre-and-jvm)
2. [Java Runtime Environment (JRE)](https://www.javatpoint.com/difference-between-jdk-jre-and-jvm#jre)
3. [Java Development Kit (JDK)](https://www.javatpoint.com/difference-between-jdk-jre-and-jvm#jdk)

We must understand the differences between JDK, JRE, and JVM before proceeding further to [Java](https://www.javatpoint.com/java-tutorial). 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](https://www.javatpoint.com/os-tutorial) 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
* Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.

JVM Architecture



### **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 loades 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.

### **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.

### **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](https://www.javatpoint.com/java-applet). 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.



# Java Variables

A variable is a container which holds the value while the [Java program](https://www.javatpoint.com/simple-program-of-java) 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](https://www.javatpoint.com/java-data-types): primitive and non-primitive.

**Types of Variables**

There are three types of variables in [Java](https://www.javatpoint.com/java-tutorial):

* local variable
* instance variable
* static variable

#### **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](https://www.javatpoint.com/static-keyword-in-java).

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.

**public** **class** A{

**static** **int** m=100; //static variable

**void** method(){

**int** n=90; //local variable

    }

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

**int** data=50; //instance variable

    }

} //end of class

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](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

## 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](https://www.javatpoint.com/java-tutorial).

* 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 |

Operators in Java

**Operator** in [Java](https://www.javatpoint.com/java-tutorial) is a symbol that is used to perform operations. For example: +, -, \*, / etc.

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 Scanner Class

Scanner class in Java is found in the java.util package. Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.

The Java Scanner class breaks the input into tokens using a delimiter which is whitespace by default. It provides many methods to read and parse various primitive values.

The Java Scanner class is widely used to parse text for strings and primitive types using a regular expression. It is the simplest way to get input in Java. By the help of Scanner in Java, we can get input from the user in primitive types such as int, long, double, byte, float, short, etc.

The Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

The Java Scanner class provides nextXXX() methods to return the type of value such as nextInt(), nextByte(), nextShort(), next(), nextLine(), nextDouble(), nextFloat(), nextBoolean(), etc. To get a single character from the scanner, you can call next().charAt(0) method which returns a single character.

## How to get Java Scanner

To get the instance of Java Scanner which reads input from the user, we need to pass the input stream (System.in) in the constructor of Scanner class. For Example:

Scanner in = **new** Scanner(System.in);

**import** java.util.Scanner;

**public** **class** ScannerExample {

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

          Scanner sc = **new** Scanner(System.in);

          System.out.print("Enter your name : ");

          String name = sc.nextLine();

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

          sc.close();

          }

}