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.

*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.

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 Microsystem, Java language is a simple programming language because:

* Java syntax is based on 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 behaviour. Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

**Platform Independent**: 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.

**Robust**: 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).

First Program in Java

*public* *class* helloworld {

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

System.out.println("Hello World!");

}

}

*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. 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.

*Valid Java main() method signatures*

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

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

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

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

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

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

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

*final* *strictfp* *public* *static* *void* main(*String*[] args)

*What is compile time in Java?*

After writing the program, one needs to compile that program. As soon as the someone starts compiling the program using javac compiler, the compile time gets started, and it ends when either a .class file is generated after successful compilation, or an error is thrown in compilation. In other way we can say, the process of compiling a program is referred as compile time.

For an example if you wrote a program and saved it as MyFirstProgram.java, Once you start compiling this program using javac command as javac MyFirstProgram.java, the compile time gets started and it ends when a .class file as MyFirstProgram.class is generated, or any error is thrown in compilation.

*What happens at compile time in Java?*

At compile time, the java compiler(javac) takes the source code(.java file) and checks if there is any syntax, type-checking, or any semantic errors inside the program. If there is no error, the compiler generates a .class(bytecode) file for that .java file. If there is any compilation error, java compiler displays that error in command window

*What is compile time error in Java?*

If a program element(class, method, variable, statements etc) is not written as per its syntax in java, the compiler throws error for that element while compiling the program. We call these errors as compile time errors as these errors are detected at compile time by the java compiler. The java compiler does not create a .class file if there is any compile time error detected while compiling the program.

*What is runtime in Java?*

As soon as one starts executing the program using java command, runtime gets started and it ends when execution of program ended either successfully or unsuccessfully. In other way the process of running a program is known as runtime.

For an example if you wrote a program and saved it as MyFirstProgram.java. After compilation when you execute the command java MyFirstProgram for running the program, runtime gets started and it ends when either the output of program is generated, or any runtime error is thrown.

*What is runtime error in Java?*

Errors which come during the execution(runtime) of a program are known as runtime errors. If a program contains runtime error, it won't run successfully, rather that error will be displayed in command window at the time of execution. E.g., division by zero.

NOTE: Do not use the extension .class in the command line when executing the program. Use java ClassName to run the program. If you use java ClassName.class in the command line, the system will attempt to fetch ClassName.class.class.

NOTE: If you execute a class file that does not exist, a NoClassDefFoundError will occur. If you execute a class file that does not have a main method or you mistype the main method (e.g., by typing Main instead of main), a NoSuchMethodError will occur.

NOTE: When executing a Java program, the JVM first loads the bytecode of the class to memory using a program called the class loader. If your program uses other classes, the class loader dynamically loads them just before they are needed. After a class is loaded, the JVM uses a program called the bytecode verifier to check the validity of the bytecode and to ensure that the bytecode does not violate Java’s security restrictions. Java enforces strict security to make sure that Java class files are not tampered with and do not harm your computer.

**Java Virtual Machine (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).

JVM is a:

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.

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*

**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.

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

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

**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.

**Program Counter Register:** PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

**Native Method Stack:** It contains all the native methods used in the application.

**Execution Engine:** An execution engine 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.

**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 Runtime Environment (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.

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Java Development Kit (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.

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Variables & Data Types in Java**

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.

*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.

*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.

*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.

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

* Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.
* 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.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Type | Default Value | Size | Declaration |
| boolean | false | 1 bit | boolean a = true; |
| char | ‘\u0000’ | 2 byte | char b = ‘a’; |
| byte | 0 | 1 byte | byte c = 24; |
| short | 0 | 2 byte | short d = 35; |
| int | 0 | 4 byte | int e = 244; |
| long | 0L | 8 byte | long f = 44569; |
| float | 0.0f | 4 byte | float g = 0.23f; |
| double | 0.0d | 8 byte | double h = 2.34; |

**CODE: Example for Variables in Java**

*public* *class* javavariables {

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

*int* a = 10;

*float* f = a; *// implicit type casting - Widening*

        System.out.println(a);

        System.out.println(f);

        System.out.println();

*float* f1 = 10.5f;

*int* a1 = (*int*)f1; *// explicit type casting - Narrowing*

*// int a = f; -> This will throw compile time error*

        System.out.println(f1);

        System.out.println(a1);

        System.out.println();

*int* c = 130;

*byte* b = (*byte*)a; *// explicit type casting - Overflow*

        System.out.println(c);

        System.out.println(b);

        System.out.println();

*byte* x = 10;

*byte* y = 20;

*// byte z = x + y; -> This will throw compile time error as the result of x + y is int*

*byte* z = (*byte*)(x + y);

        System.out.println(x);

        System.out.println(y);

        System.out.println(z);

    }

}

*Question: What is the Integral Promotion Rule in Java?*

Automatic Integral Promotion means that the variable is in a lower data type (e.g., int). Now, if this integer is called into a method which has input parameters defined in a higher type (e.g., float, double), then the integer is automatically promoted to floating point or a double data type. This is automatic integral promotion. This is implemented in the code below:

*public* *class* javavariables {

*public* *static* *void* method1(*double* d){

        System.out.println("The value of variable in the method: " + d);

    }

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

*int* a = 10;

        System.out.println("The value of variable in the main method: " + a);

        method1(a);

    }

}

*Question: Why is Java Platform Independent?*

The meaning of Platform independent means that the Java Byte code that is compiled can run on all Operating Systems. The program is written in a human-like language which is then converted into a machine level code which the machine can understand, and this is done by the Java compiler. Whenever a program is written in Java, it is compiled by the javac (Java Compiler) whose result is a .class file or we can say the bytecode. This generated bytecode is a non-executable file and needs to be executed on the machine. This interpreter is the Java Virtual Machine and thus the bytecode is executed by the JVM. Now, it is the trick of bytecode that makes Java Platform independent. This adds that Java has portability. Every system has its own JVM which gets installed automatically whenever JDK software is installed. For every OS, separate JVM is available that is capable to read the .class file or the compiled bytecode. It should be noted that Java is platform independent and not the JVM, different JVM exists for different OS and the byte code is the one that is able to run on each OS.