The History and Evolution of Java I

To fully understand Java, one must understand the reasons behind its creation, the forces that shaped it, and the legacy that it inherits.

Like the successful computer languages that came before, Java is a blend of the best elements of its rich heritage combined with the innovative concepts required by its unique mission.

The Creation of Java

Java was created by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991.

▶ It took 18 months to develop the first working version.

▶ This language was initially called “Oak”, but was renamed “Java” in 1995.

▶ Between the initial implementation of Oak in the fall of 1992 and the public announcement of Java in the spring of 1995, many more people contributed to the design and evolution of the language.

The History and Evolution of Java III

► Bill Joy, Arthur van Hoff, Jonathan Payne, Frank Yellin, and Tim Lindholm were key contributors to the maturing of the original prototype.

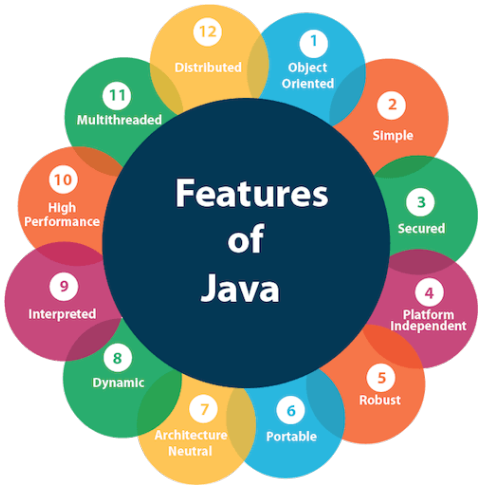
The Java Buzzwords I

No discussion of Java's history is complete without a look at the Java buzzwords.

Although the fundamental forces that necessitated the invention of Java are **portability and security**, other factors also played an important role in molding the final form of the language.

The key considerations were summed up by the Java team in the following list of **buzzwords**:

The Java Buzzwords II



The Java Buzzwords III

1. Object-oriented
2. Simple
3. Secure
4. Platform Independent
5. Robust
6. Portable
7. Architecture-neutral
8. Dynamic

The Java Buzzwords IV

9. Interpreted
10. High performance
11. Multithreaded
12. Distributed

The Java Buzzwords V

- **Simple** : Java was designed to be easy for the professional programmer to learn and use effectively.

  ✓Assuming that you have some programming experience, you will not find Java hard to master.

  ✓If you already understand the basic concepts of object-oriented programming, learning Java will be even easier.

  ✓Best of all, if you are an experienced C++ programmer, moving to Java will require very little effort.

The Java Buzzwords VI

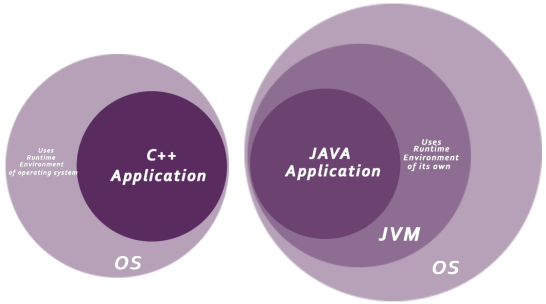
✓ Because Java inherits the C/C++ syntax and many of the object-oriented features of C++, most programmers have little trouble learning Java.

✓ 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.

The Java Buzzwords VII

- **Secure** :



The Java Buzzwords VIII

1. **Pointer Arithmetic**
   - **Java:** No explicit pointers. References are safe and cannot be manipulated to access arbitrary memory.
   - **C:** Explicit pointer arithmetic. A powerful feature but a major source of errors, allowing corruption of arbitrary memory locations.
2. **Array Bounds Checking**
   - **Java:** Always performed. Accessing an array outside its bounds throws an immediate `ArrayIndexOutOfBoundsException`.
   - **C:** Not performed. Out-of-bounds access leads to undefined behavior, often corrupting adjacent memory (a classic buffer overflow).

The Java Buzzwords IX

❸ Type Safety

- **Java:** Strongly typed. The language and JVM enforce strict type rules. Implicit casts are only allowed for safe widening conversions.
- **C:** Weakly typed. Allows implicit casts between types (e.g., `int` to `char*`), which can easily lead to logical errors and vulnerabilities.

❹ Buffer Overflow Vulnerabilities

- **Java:** Not possible due to array bounds checking and lack of pointer arithmetic.
- **C:** The most common and critical vulnerability. A primary attack vector for injecting malicious code.

The Java Buzzwords X

❺ String Handling

- **Java:** Immutable `String` class. Operations create new objects, preventing accidental modification of original data.
- **C:** Mutable null-terminated arrays of characters. Easy to miscalculate lengths, leading to overflows and corruption.

❻ Native Code Execution

- **Java:** Runs on the JVM. Code is isolated from the underlying OS and hardware by the virtual machine.
- **C:** Compiles directly to native machine code. Code has the same privileges as the host process, making any vulnerability far more dangerous.

The Java Buzzwords XI

❼ Bytecode Verification

- **Java:** Yes. The JVM verifies all bytecode before execution, ensuring it follows language rules and is safe to run.
- **C:** No. The compiler generates machine code, and the CPU executes it directly without any further checks.

❽ Security Manager / Sandbox

- **Java:** Yes. Provides a configurable sandbox to restrict actions based on a security policy.
- **C:** No. The program has the full permissions of the user who executes it.

❾ Exception Handling

- **Java:** Robust built-in mechanism. Errors are forced to be caught or declared, preventing crashes from going unhandled.
- **C:** Very minimal (`setjmp/longjmp`). Errors often lead to program termination (e.g., segmentation fault) or are ignored, creating unstable states.

⑩ Access Control

- **Java:** Strong (`public`, `private`, `protected`, package). Enforced at both compile time and by the JVM at runtime.
- **C:** Very weak (`static` functions/variables). Primarily a compile-time linker feature; memory can often be accessed directly.

⑪ Standard Library Safety

- **Java:** Designed with safety. Methods often include built-in bounds checks.
- **C:** Notoriously unsafe. Functions like `gets()`, `strcpy()`, and `scanf()` are infamous for causing buffer overflows.

⓬ Undefined Behavior

- **Java:** Minimal. The JVM spec tightly defines behavior for almost all operations, ensuring predictability across platforms.
- **C:** Pervasive. Many actions, like signed integer overflow or accessing invalid pointers, result in undefined behavior, making programs unpredictable and hard to secure.

⓭ Code Portability & Sandbox

- **Java:** The JVM is the environment. The same sandbox and security policies work across Windows, Linux, macOS, etc.
- **C:** No portable sandbox. Security mechanisms are OS-specific (e.g., SELinux, seccomp-bpf on Linux), making secure deployment complex.

⓮ Object Orientation & Encapsulation

- **Java:** Everything is an object (except primitives). Encapsulation helps protect an object's internal state.
- **C:** Procedural language. Data and functions are separate, making encapsulation harder to enforce.

⓯ Memory Corruption

- **Java:** Extremely difficult to achieve due to the layers of abstraction and checks.
- **C:** The defining vulnerability class. Entire categories of exploits (e.g., stack smashing, heap spraying) are based on it.

⑯ Learning Curve for Safety

- **Java:** Easier to write safe code. The language and runtime prevent many common mistakes by default.
- **C:** Hard to master safety. Writing secure C code requires expert-level knowledge and constant vigilance.

⑰ Performance Cost

- **Java:** Higher. Overhead from the JVM, garbage collection, and runtime checks (bounds, null, types) impacts performance.
- **C:** Lower. Compiles to efficient native code with minimal overhead, which is why it's used in performance-critical systems.

⑱ Vulnerability Surface

- **Java:** Smaller. Many classic vulnerability types are eliminated by the language design.
- **C:** Larger. The programmer is responsible for avoiding a vast landscape of potential pitfalls.

⑲ Design Philosophy

The Java Buzzwords XVII

- **Java:** "Safety by Design." Security is a primary goal, enforced by the language and runtime environment.
- **C:** "Freedom and Responsibility." Provides maximum power and flexibility, with the assumption that the programmer knows what they are doing.

Platform Independent :

What is Platform?

The Java Buzzwords XIX

Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own run-time environment Java Run-Time Environment (JRE) and Application Programming Interface (API), it is called platform.

✓ "Platform Independence" is one of the core feature of Java.

► **Bytecode** is an intermediate code between the **source code** and **machine code**. It is a **low-level code** that is the result of the compilation of a source code which is written in a high-level language. It is processed by a virtual machine like Java Virtual Machine (JVM).

► Bytecode is a non-runnable code after it is translated by an interpreter into machine code then it is understandable by the machine.

The Java Buzzwords XX

► It is compiled to run on JVM, any system having JVM can run it irrespective of their operating system. That's why Java is platform-independent. Bytecode is referred to as a Portable code.

```
public int add(int a, int b) {
    return a + b;
}
```

```
0: iload_1  // Load first integer parameter
1: iload_2  // Load second integer parameter
2: iadd     // Add the two integers
3: ireturn  // Return the integer result
```

The Java Buzzwords XXII

```
def add(a, b):
    return a + b
```

```
 0 LOAD_FAST                0 (a)
 2 LOAD_FAST                1 (b)
 4 BINARY_ADD
 6 RETURN_VALUE
```

The Java Buzzwords XXIII

```
for (int i = 2; i < 1000; i++) {
    for (int j = 2; j < i; j++) {
        if (i % j == 0)
            continue outer;
    }
    System.out.println (i);
}
```

```
 0: iconst_2           // Push int constant 2 (initialize i)
 1: istore_1           // Store in local var 1 (i = 2)

 2: iload_1            // Load i (outer loop start)
 3: sipush 1000        // Push 1000
 6: if_icmpge 44       // if i >= 1000, jump to instruction 44 (exit)

 9: iconst_2           // Push 2 (initialize j)
10: istore_2           // Store in local var 2 (j = 2)
```

```
11: iload_2             // Load j (inner loop start)
12: iload_1             // Load i
13: if_icmpge 32        // if j >= i, jump to instruction 32 (print)

16: iload_1             // Load i (if i % j == 0)
17: iload_2             // Load j
18: irem                // Compute i % j
19: ifne 26             // if remainder != 0, jump to instruction 26 (j++)
```

```
22: iinc 2, 1          // Increment j by 1 (skipped by continue)
23: goto 2             // Jump to outer loop (continue outer)

26: iinc 2, 1          // Increment j by 1 (inner loop iteration)
29: goto 11            // Jump to inner loop start

32: getstatic #2       // Get System.out (println)
35: iload_1            // Load i
36: invokevirtual #3   // Call println
39: iinc 1, 1          // Increment i by 1
```

```
42: goto 2              // Jump to outer loop start


44: return              // End of method
```

The Java Buzzwords XXVIII

Machine Code: Machine code is a set of instructions that is directly machine-understandable and it is processed by the Central Processing Unit (CPU).

► Machine code is in binary (0's and 1's) format which is completely different from the byte code and source code. It is regarded as the most lowest-level representation of the source code.

► Machine code is obtained after compilation or interpretation. It is also called machine language.

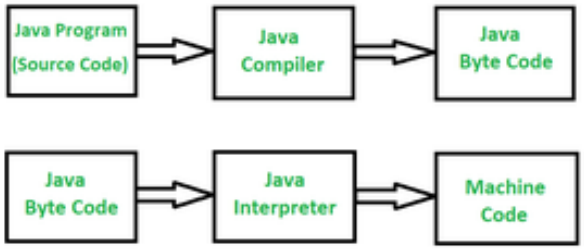


Figure 1: Java source code is converted to Bytecode and then to machine code

Difference between Byte Code and Machine Code:

	Bytecode	Machine Code
01.	Bytecode consisting of binary, hexadecimal, macro instructions like (new, add, swap, etc) and it is not directly understandable by the CPU. It is designed for efficient execution by software such as a virtual machine.intermediate-level	Machine code consisting of binary instructions that are directly understandable by the CPU.
02.	Bytecode is considered as the intermediate-level code.	Machine Code is considered as the low-level code.

The Java Buzzwords XXXI

	Byte Code	Machine Code
03.	Bytecode is a non-runnable code generated after compilation of source code and it relies on an interpreter to get executed.	Machine code is a set of instructions in machine language or in binary format and it is directly executed by CPU.
04.	Bytecode is executed by the virtual machine then the Central Processing Unit.	Machine code is not executed by a virtual machine it is directly executed by CPU.
05.	Bytecode is less specific towards machine than the machine code.	Machine code is more specific towards machine than the byte code.

The Java Buzzwords XXXII

	Byte Code	Machine Code
06.	It is platform-independent as it is dependent on the virtual machine and the system having a virtual machine can be executed irrespective of the platform.	It is not platform independent because the object code of one platform can not be run on the same Operating System. Object varies depending upon system architecture and native instructions associated with the machine.
07.	All the source code need not be converted into byte code for execution by CPU. Some source code written by any specific high-level language is converted into byte code then byte code to object code for execution by CPU.	All the source code must be converted into machine code before it is executed by the CPU.

The Java Buzzwords XXXIII

✓ Platform independent: Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into **platform independent bytecode**.

✓This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.

✓ Platform independence in Java means you can “write once, run anywhere” which notifies that you can run Java code in any operating system that supports Java without recompilation.

The Java Buzzwords XXXIV

✓ 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).

✓ The Java language typically compiles to a Virtual Machine a virtual CPU which runs all of the code that is written for the language.
✓ This enables the same executable binary to run on all systems that implement a Java Virtual machine (JVM).
✓ Java programs can be executed natively using a Java Processor. This isn't common and is mostly used for embedded systems.

The Java Buzzwords XXXV

✓Java code running in the JVM has access to OS-related services, like disk I/O and network access, if the appropriate privileges are granted. The JVM makes the system calls on behalf of the Java application.

✓ Currently, Java Standard Edition programs can run on Microsoft Windows , Mac OS X, several UNIX-Like operating systems, and several more non-UNIX-like operating systems like embedded systems. For mobile applications, browser plugins are used for Windows and Mac based devices, and Android has built-in support for Java.

Portable :

What do you mean by portability?

The Java Buzzwords XXXVII

✓Portability refers to the ability to run a program on different machines.

✓Running a given program on different machines can require different amounts of work (for example, no work whatsoever, recompiling, or making small changes to the source code).

✓ Java is object-compatible. You compile it on one platform and the resultant class files can run on any JVM.

The Java Buzzwords XXXVIII

Note: **C is source-portable. You can take your C source code and compile it on any ISO C compiler, provided you follow the rules - that means no using undefined or implementation defined behaviour or any non-standard features.**

► Java provides three distinct types of portability:

✓ **Source code portability**: A certain Java program must produce identical results, CPU, of the operating system or the underlying Java compiler.

The Java Buzzwords XXXIX

- ✓ **CPU architecture portability**: Current Java compilers produce object code (called byte-code) to a CPU that does not yet exist. For each real CPU in which the Java programs must run, a Java interpreter or virtual machine, executes the Java code. This nonexistent CPU allows the same object code to be executed in any CPU for which there is a Java interpreter.

The Java Buzzwords XL

✓ **OS / GUI**: Java solves this problem by providing a set of library functions (contained in libraries provided by Java, such as awt, util and lang) that converse with an imaginary operating system and an imaginary GUI. Just as the JVM presents a virtual CPU, the Java libraries have a virtual operating system / GUI.

- Each Java implementation provides libraries that implement this virtual OS / GUI.

The Java Buzzwords XLI

✓Portability is a major aspect of the Internet because there are many different types of computers and operating systems connected to it.

✓If a Java program were to be run on virtually any computer connected to the Internet, there needed to be some way to enable that program to execute on different systems.

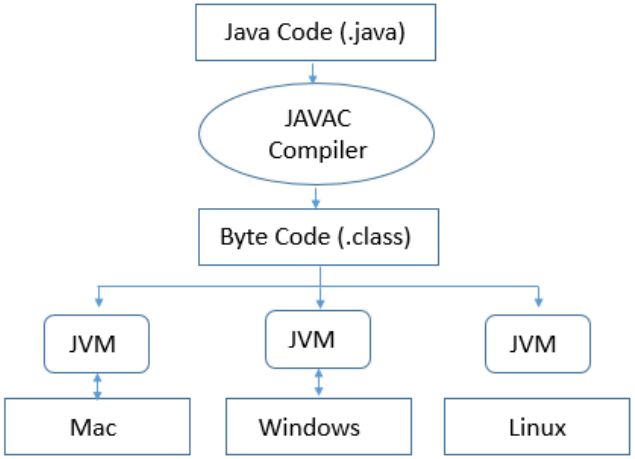
✓For example, in the case of an applet, the same applet must be able to be downloaded and executed by the wide variety of CPUs, operating systems, and browsers connected to the Internet.

✓It is not practical to have different versions of the applet for different computers.

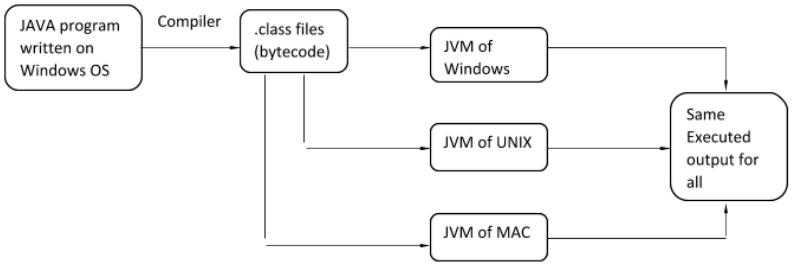
The Java Buzzwords XLII

✓ The same code must work on all computers. Therefore, some means of generating portable executable code was needed.

The Java Buzzwords XLIII



The Java Buzzwords XLIV



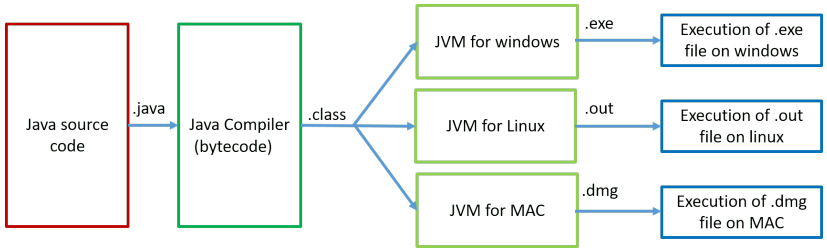


Figure 2: Java platform neutral

The Java Buzzwords XLVI

Object-Oriented :

✓ Although influenced by its predecessors, Java was not designed to be source-code compatible with any other language.

✓ This allowed the Java team the freedom to design with a blank slate.

✓One outcome of this was a clean, usable, pragmatic approach to objects.

✓Borrowing liberally from many seminal object-software environments of the last few decades, Java manages to strike a balance between the purist's "everything is an object" paradigm and the pragmatist's "stay out of my way" model.

The Java Buzzwords XLVII

✓The object model in Java is simple and easy to extend, while primitive types, such as integers, are kept as high-performance non-objects.

✓ Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.
✓ Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules. These rules are:

- Object
- Class

The Java Buzzwords XLVIII

- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

The Java Buzzwords XLIX

Robust :

✓ Java is Robust because it is highly supported language.

✓ It is portable across many Operating systems.

✓ Java also has feature of Automatic memory management and garbage collection.

✓ Strong type checking mechanism of on a program, because the program must execute reliably in a variety of systems. Thus, the ability to create robust programs was given a high priority in the design of Java.

✓ Robust programming is a style of programming that focuses on handling unexpected termination and unexpected actions. It requires code to handle these terminations and actions

gracefully by displaying accurate and unambiguous error messages. These error messages allow the user to more easily debug the program.

To better understand how Java is robust, consider two of the main reasons for program failure: memory management mistakes and mishandled exceptional conditions (that is, run-time errors).

✓ Memory management can be a difficult, tedious task in traditional programming environments.

✓ For example, in C/C++, the programmer will often manually allocate and free all dynamic memory.

✓ This sometimes leads to problems, because programmers will either forget to free memory that has been previously allocated or, worse, try to free some memory that another part of their code is still using.

The Java Buzzwords LI

✓ Java virtually eliminates these problems by managing memory allocation and deallocation for you. (In fact, deallocation is completely automatic, because Java provides garbage collection for unused objects.)

✓ Exceptional conditions in traditional environments often arise in situations such as division by zero or “file not found,” and they must be managed with clumsy and hard-to- read constructs.

Java helps in this area by providing object oriented exception handling. In a well-written Java program, all run-time errors can—and should—be managed by your program.

The Java Buzzwords LII

```
if(var == true) {
...
} else {
...
}
```

The Java Buzzwords LIII

```
if(var == true) {
...
} else if (var == false) {
...
}
```

The Java Buzzwords LIV

```
if(var == true) {
...
} else if (var == false) {
...
} else {
...
}
```

- It uses strong memory management.

The Java Buzzwords LV

- There is a lack of pointers that avoids security problems.
- There is automatic garbage collection in java 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.

The Java Buzzwords LVI

Multithreaded :

✓ Java was designed to meet the real-world requirement of creating interactive, networked programs.

✓ To accomplish this, Java supports multithreaded programming, which allows you to write programs that do many things simultaneously.

✓The Java run-time system comes with an elegant yet sophisticated solution for multiprocess synchronization that enables you to construct smoothly running interactive systems.

✓Java's easy-to-use approach to multithreading allows you to think about the specific behavior of your program, not the multitasking subsystem.

The Java Buzzwords LVII

Architecture-neutral :

A central issue for the Java designers was that of code longevity and portability.

✓ At the time of Java's creation, one of the main problems facing programmers was that no guarantee existed that if you wrote a program today, it would run tomorrow—even on the same machine.

✓ Operating system upgrades, processor upgrades, and changes in core system resources can all combine to make a program malfunction.

The Java Buzzwords LVIII

✓ The Java designers made several hard decisions in the Java language and the Java Virtual Machine in an attempt to alter this situation. Their goal was “write once; run anywhere, any time, forever.” To a great extent, this goal was accomplished.

✓ 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.

The Java Buzzwords LIX

Interpreted and High Performance:

✓ Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode.

✓This code can be executed on any system that implements the Java Virtual Machine. Most previous attempts at cross platform solutions have done so at the expense of performance.

✓The Java bytecode was carefully designed so that it would be easy to translate directly into native machine code for very high performance by using a just-in-time compiler.

✓Java run-time systems that provide this feature lose none of the benefits of the platform-independent code.

✓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.

The Java Buzzwords LXI

Distributed :

Java is designed for the distributed environment of the Internet because it handles TCP/IP protocols. In fact, accessing a resource using a URL is not much different from accessing a file.

►Java also supports Remote Method Invocation (RMI). This feature enables a program to invoke methods across a network.

The Java Buzzwords LXII

Dynamic: Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe and expedient manner. This is crucial to the robustness of the Java environment, in which small fragments of bytecode may be dynamically updated on a running system.

Java Virtual Machine

Difference between JDK, JRE, and JVM II

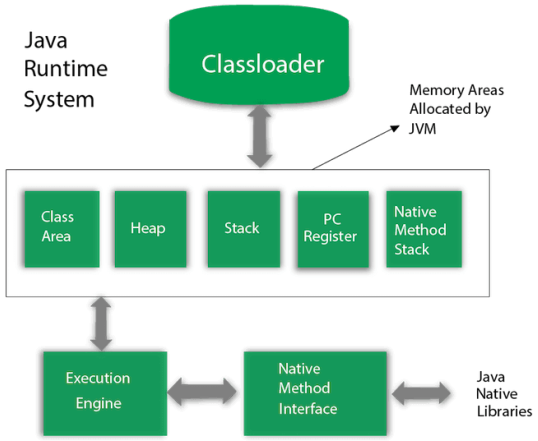


Figure 3: JVM architecture

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed with a goal of being easy to use and accessible to 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 through class. It can be achieved by interfaces in Java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in Java. It means Java has restricted pointer support in Java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses compiler and interpreter both. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted 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 comment.	Java supports documentation comment (/** ... */) to create documentation for Java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>>operator.	Java supports unsigned right shift >>>operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >>operator.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses a single inheritance tree always because all classes are the child of Object class in Java. The object class 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 C language, single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from Java.lang.Object.

Package I

► A package is a **namespace** that organizes a set of related classes and interfaces.

A namespace is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it. Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries.

► Conceptually you can think of packages as being similar to different folders on your computer.

Package II

► You might keep HTML pages in one folder, images in another, and scripts or applications in yet another. Because software written in the Java programming language can be composed of hundreds or thousands of individual classes, it makes sense to keep things organized by placing related classes and interfaces into packages.

►The Java platform provides an enormous class library (a set of packages) suitable for use in your own applications. This library is known as the ”Application Programming Interface”, or ”API” for short.

Package III

► Its packages represent the tasks most commonly associated with general-purpose programming. For example, a String object contains state and behavior for character strings; a File object allows a programmer to easily create, delete, inspect, compare, or modify a file on the filesystem; a Socket object allows for the creation and use of network sockets; various GUI objects control buttons and checkboxes and anything else related to graphical user interfaces.

►There are literally thousands of classes to choose from. This allows you, the programmer, to focus on the design of your particular application, rather than the infrastructure required to make it work.

Package IV

► The Java Platform API Specification contains the complete listing for all packages, interfaces, classes, fields, and methods supplied by the Java SE platform.

►As a programmer, it will become your single most important piece of reference documentation.