# The Differences Between JAVA And C/C++

## PreProcessor
All C/C++ compilers implement a stage of compilation known as the preprocessor. The C++ preprocessor basically performs an intelligent search and replace on identifiers that have been declared using the #define or #typedef directives. Most of the processor definitions in C++ are stored in header files, which complement the actual source code files. Java does not have a preprocessor. It provides similar functionality (#define, #typedef, and so on) to that provided by the C++ preprocessor, but with far more control. Constant data members are used in place of the #define directive, and class definitions are used in lieu of the #typedef directive. The result is that Java source code is much more consistent and easier to read than C++ source code. Additionally, **Java programs don't use header files**; the Java compiler builds class definitions directly from the source code files, which contain both class definitions and method implementations.

## Pointers
Most developers agree that the misuse of pointers causes the majority of bugs in C/C++ programming. Put simply, when you have pointers, you have the ability to trash memory. C++ programmers regularly use complex pointer arithmetic to create and maintain dynamic data structures. In return, C++ programmers spend a lot of time hunting down complex bugs caused by their complex pointer arithmetic. The Java language does not support pointers. Java provides similar functionality by making heavy use of references. Java passes all arrays and objects by reference. This approach prevents common errors due to pointer mismanagement. It also makes programming easier in a lot of ways simply because the correct usage of pointers is easily misunderstood by all but the most seasoned programmers.

**Structure and Unions** There are three types of complex data types in C++: classes, structures, and unions. **Java only implements** one of these data types: **classes.** Java forces programmers to use classes when the functionality of structures and unions is desired. Although this sounds like more work for the programmer, it actually ends up being more consistent, because classes can imitate structures and unions with ease.

## Multiple Inheritance
Multiple inheritance is a feature of C++ that allows you to derive a class from multiple parent classes. Although multiple inheritance is indeed powerful, it is complicated to use correctly and causes many problems otherwise. It is also very complicated to implement from the compiler perspective. Java takes the high road and **provides no direct support for multiple inheritance**. You can implement functionality similar to multiple inheritance by using interfaces in Java. Java interfaces provide object method descriptions but contain no implementations.

## String
C and C++ have no built-in support for text strings. The standard technique adopted among C and C++ programmers is that of using null-terminated arrays of characters to represent strings. **In Java, strings are implemented as first class objects** (String and StringBuffer), meaning that they are at the core of the Java language.

## Goto Statement

The dreaded goto statement is pretty much a relic these days even in C and C++, but it is technically a legal part of the languages. The goto statement has historically been cited as the cause for messy, impossible to understand, and sometimes even impossible to predict code known as "spaghetti code." The primary usage of the goto statement has merely been as a convenience to substitute not thinking through an alternative, more structured branching technique.
For all these reasons and more, Java does not provide a goto statement. The Java language specifies goto as a keyword, but its usage is not supported. I suppose the Java designers wanted to eliminate the possibility of even using goto as an identifier! Not including goto in the Java language simplifies the language and helps eliminate the option of writing messy code.

## Operator Overloading

Operator overloading, which is considered a prominent feature in C++, **is not supported in Java.** Although roughly classes in Java can implement the same functionality, the convenience of operator overloading is still missing. However, in defense of Java, operator overloading can sometimes get very tricky. No doubt the Java developers decided not to support operator overloading to keep the Java language as simple as possible.

## Automatic Coercions

Automatic coercion refers to the implicit casting of data types that sometimes occurs in C and C++. For example, in C++ you can assign a float value to an int variable, which can result in a loss of information. Java does not support C++ style automatic coercions. In Java, if coercion will result in a loss of data, you must always explicitly cast the data element to the new type.

## Variable Arguments

C and C++ let you declare functions, such as printf, that take a variable number of arguments. Although this is a convenient feature, it is impossible for the compiler to thoroughly type check the arguments, which means problems can arise at runtime without you knowing. Again Java takes the high road and doesn't support variable arguments at all.

## Command-line Arguments

The command-line arguments passed from the system into a Java program differ in a couple of ways from the command-line arguments passed into a C++ program. First, the number of parameters passed differs between the two languages.
In C and C++, the system passes two arguments to a program: argc and argv. argc specifies the number of arguments stored in argv. argv is a pointer to an array of characters containing the actual arguments. In Java, the system passes a single value to a program: args. ‘args’ is an array of Strings that contains the command-line arguments.