**<http://www.programmersheaven.com/2/FAQ-JAVA-Differences-Between-JAVA-And-C-CPP>**

**The Differences Between Java, C And C++**

This article aims to set out some of the differences between C, C++ and Java. What it does not aim to do is conclude that one language is always the best one to use. Language choice depends upon a range of factors including field of application (operating systems, desktop software, web applications etc), what programming paradigm suits the application (OOP, procedural, etc), the target platform and available programmer expertise. What follows should help you decide where it might be suitable to use C, C++ or Java.  
  
**Paradigm**  
  
C is geared towards procedural programming. That is, you write a number of procedures to do certain tasks and build up a program by calling those procedures as needed.   
  
Java, on the other hand, is geared towards OOP (object oriented programming). With OOP, you define classes which represent an entity (for example, a window, a button, a string of text, a file). From one class many objects may be created, with every object of a certain class having the fields (places to store data) and methods (named blocks of code associated with the object) as defined by the class.  
  
It is possible to write in an object oriented style in C and in a procedural style in Java, but in each case the language will somewhat get in your way. C++ is designed to support both paradigms.  
  
**Preprocessor**  
  
All C and C++ compilers implement a stage of compilation known as the preprocessor. The preprocessor basically performs an intelligent search and replace on identifiers that have been declared using the #define or #typedef directives. #define can also be used to declare macros. For example, a macro MAX(x,y) could be defined to return whichever of x or y holds the greatest value. This is not like calling a function as the substitution is done before the code is compiled. Most of the preprocessor definitions in C and C++ are stored in header files, which complement the actual source [code files](http://www.programmersheaven.com/2/FAQ-JAVA-Differences-Between-JAVA-And-C-CPP).   
  
Java does not have a preprocessor. Constant data members are used in place of the #define directive and class definitions are used in lieu of the #typedef directive, however there is no substitute for macros, which can be useful. The Java approach to defining constants and naming types of data structures is probably conceptually simpler for the programmer. Additionally, [Java programs](http://www.programmersheaven.com/2/FAQ-JAVA-Differences-Between-JAVA-And-C-CPP) don't use header files; the [Java compiler](http://www.programmersheaven.com/2/FAQ-JAVA-Differences-Between-JAVA-And-C-CPP) builds class definitions directly from the source code files, which contain both class definitions and method implementations.  
  
**Memory Management**  
  
In C and C++, any memory that is allocated **on the heap** (e.g. using malloc or new) must be explicitly freed by the programmer (e.g. using free or delete). Forgetting to free memory leads to memory leaks, and in long-running programs can lead to the memory usage of the program growing very large.

C provides three distinct ways to allocate memory for objects:[[26]](http://en.wikipedia.org/wiki/C_(programming_language)" \l "cite_note-bk21st-28)

* [Static memory allocation](http://en.wikipedia.org/wiki/Static_memory_allocation): space for the object is provided in the binary at compile-time; these objects have an [extent](http://en.wikipedia.org/wiki/Variable_(programming)#Scope_and_extent) (or lifetime) as long as the binary which contains them is loaded into memory.
* [Automatic memory allocation](http://en.wikipedia.org/wiki/Automatic_memory_allocation): temporary objects can be stored on the [stack](http://en.wikipedia.org/wiki/Call_stack), and this space is automatically freed and reusable after the block in which they are declared is exited.
* [Dynamic memory allocation](http://en.wikipedia.org/wiki/Dynamic_memory_allocation): blocks of memory of arbitrary size can be requested at run-time using library functions such as [malloc](http://en.wikipedia.org/wiki/Malloc" \o "Malloc) from a region of memory called the [heap](http://en.wikipedia.org/wiki/Dynamic_memory_allocation); these blocks persist until subsequently freed for reuse by calling the library function [realloc](http://en.wikipedia.org/wiki/Malloc" \o "Malloc) or [free](http://en.wikipedia.org/wiki/Malloc)

Java provides garbage collection, meaning that memory is freed automatically when it is no longer reachable by any references. This prevents memory leaks, but can lead to pauses in execution while the garbage collector runs. Also, there is no promise of timely destruction in Java.  
  
**Pointers**  
  
Most developers agree that the misuse of pointers causes the majority of bugs in C and C++ programs. Put simply, **when you have pointers, you have the ability to attempt to access memory that isn't yours** and modify memory relating to a different data structure than the one you intended by accident. C/C++ programmers regularly use complex pointer arithmetic to create and maintain dynamic data structures. It's powerful, but can lead to a lot of time spent hunting down complex and often subtle bugs that arise as a result of having unguarded memory access.  
  
The Java language does not support pointers. Instead, it provides similar functionality by making heavy use of references. A reference can be thought of as a "safe pointer" - the programmer can not directly manipulate the memory address. 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 inexperienced programmers.  
  
C++ does provide references too. It considers them as aliases to another variable or object. They are safer than pointers where they can be used.   
  
**Bounds Checking**  
  
An array in C or C++ is not bounds checked, so attempts to access the sixth element of a 5-element array will appear to work - that is, no runtime error will occur. This means the programmer needs to code very carefully, especially considering the potential for buffer overflow attacks.  
  
Java will bounds check arrays to prevent this from happening, of course with a little extra runtime cost.  
  
**Portability And Performance**  
  
C and C++ both compile to native machine code. This means that, with a good compiler, programs written in these languages will perform very well. However, it also restricts them to running on the platform they were compiled to run on.  
  
Java generally compiles to Java bytecode, which then runs on top of a virtual machine (the JVM). The JVM has to turn instructions in the bytecode into instructions that are understood by the machine that the bytecode is running on. This gives a runtime performance penalty (although this is getting less significant as the JVM improves and computers get faster). However, now only the virtual machine (and standard library) have to be ported to different platforms, then the bytecode for many Java programs can be executed on that platform. So bytecode is portable accross different operating systems and processors.  
  
**Complex Data Types**  
  
There are two types of complex data types in C: structures and unions. C++ adds classes to this list. Java only implements one of these data types: classes.  
  
A structure can be emulated by a class - simply write a class without any methods and make all the fields public. However, emulating a union is not always possible in Java, and the memory saving advantages unions hold in C may not carry accross. Java presents a simpler model but at the cost of not being able to save a little memory. For many applications this will be a non-issue.  
  
**Strings**  
  
C has no built-in string data type. The standard technique adopted among C programmers is that of using null-terminated arrays of characters to represent strings. This practice if often seen in C++ programs too.  
  
Neither C++ or Java have string as a primitive type, but they do both have string objects that are a standard part of the language. In Java this type is called String, and in C++ it is called CString.  
  
**Multiple Inheritance**  
  
Multiple inheritance is a feature of some object oriented languages that allows you to **derive a class from multiple parent classes**. Although multiple inheritance is indeed powerful (and sometimes the logical way to define a class hierachy), it is complicated to use correctly and can create situations where it's uncertain which method will be executed. For example, if each of the parent classes provide a method X and the [derived class](http://www.programmersheaven.com/2/FAQ-JAVA-Differences-Between-JAVA-And-C-CPP) does not, it is unclear which X should be invoked. It is also complicated to implement from the compiler perspective.  
  
C++ supports multiple inheritance. Java provides no direct support for multiple inheritance, but you can implement functionality similar to multiple inheritance by using interfaces in Java. Java interfaces provide method descriptions but contain no implementations. Therefore implementations can only be inherited from one class, so there is no ambiguity over which method to invoke.  
  
**Operator Overloading**  
  
**Operator overloading enables a class to define special behaviour for built-in operators when they are applied to objects of that class**. For example, if **the \* (multiply) operator was to be used on two objects of type Matrix**, then matrix multiplication could be implemented. This allows object types to feel much more tightly integrated into the language and can deliver much clearer code. However, sometimes it is not clear what a particular operator would sensibly do for a particular type, whereas a well-named method call would be clear.   
  
Operatoroverloading is considered a prominent feature in C++. It is not supported in Java, probably in an effort to keep the language as simple as possible and help ensure it is obvious what code does, even though it may take longer to type and read.  
  
**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, although a compiler warning will be given about this**. 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**.  
  
**Goto Statement**  
  
The goto statement is rarely used these days in C and C++, but it is a standard part of the language. The goto statement has historically been cited as the cause for messy, difficult to understand, and sometimes near impossible to predict code known as "spaghetti code." The primary bad usage of the goto statement has merely been as a convenience to substitute not thinking through an alternative, more structured branching technique. Very occasionally, it can lead to clearer code.  
  
To avoid the potential for "spaghetti code", Java does not provide a goto statement. The Java language specifies goto as a keyword, but its usage is not supported. This is consistent with Java's desire to make programmers write clear, non-messy code.  
  
**Variadic 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. Java doesn't support variable arguments at all, though if it did it would likely be able to handle subsequent runtime problems better than C or C++.  
  
**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.   
  
**Table Comparing C, C++ and Java**  
  
This table is a summary of the differences found in the article.

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | C | C++ | Java |
| Paradigms | Procedural | Procedural, OOP, Generic Programming | OOP, Generic Programming (from Java 5) |
| Form of Compiled Source Code | Executable Native Code | Executable Native Code | Java bytecode |
| Memory management | Manual | Manual | Managed, using a garbage collector |
| Pointers | Yes, very commonly used. | Yes, very commonly used, but some form of references available too. | No pointers; references are used instead. |
| Preprocessor | Yes | Yes | No |
| String Type | Character arrays | Character arrays, objects | Objects |
| Complex Data Types | Structures, unions | Structures, unions, classes | Classes |
| Inheritance | N/A | Multiple class inheritance | Single class inheritance, multiple interface implementation |
| Operator Overloading | N/A | Yes | No |
| Automatic coercions | Yes, with warnings if loss could occur | Yes, with warnings if loss could occur | Not at all if loss could occur; must cast explicitly |
| Variadic Parameters | Yes | Yes | No |
| Goto Statement | Yes | Yes | No |

<http://stackoverflow.com/questions/9192309/the-main-difference-between-java-c>

1. C++ supports pointers whereas Java does not pointers. But when many programmers questioned how you can work without pointers, the promoters began saying "Restricted pointers.” So we can say java supports Restricted pointers.
2. At compilation time Java Source code converts into byte code .The interpreter execute this byte code at run time and gives output .Java is interpreted for the most part and hence platform independent. C++ run and compile using compiler which converts source code into machine level languages so c++ is plate from dependents
3. Java is platform independent language but c++ is depends upon operating system machine etc. C++ source can be platform independent (and can work on a lot more, especially embedeed, platforms), although the generated objects are generally platofrom dependent but there is clang for llvmwhich doesn't have this restriction.
4. Java uses compiler and interpreter both and in c++ there is only compiler
5. C++ supports operator overloading, multiple inheritance but java does not.
6. C++ is more nearer to hardware then Java
7. Everything (except fundamental types) is an object in Java (Single root hierarchy as everything gets derived from java.lang.Object).
8. Java does is a similar to C++ but not have all the complicated aspects of C++ (ex: Pointers, templates, unions, operator overloading, structures etc..) Java does not support conditional compile (#ifdef/#ifndef type).
9. Thread support is built-in Java but not in C++. C++11, the most recent iteration of the C++ programming language does have Thread support though.
10. Internet support is built-in Java but not in C++. However c++ has support for socket programming which can be used.
11. Java does not support header file, include library files just like C++ .Java use import to include different Classes and methods.
12. Java does not support default arguments like C++.
13. There is no scope resolution operator :: in Java. It has . using which we can qualify classes with the namespace they came from.
14. There is no goto statement in Java.
15. Exception and Auto Garbage Collector handling in Java is different because there are no destructors into Java.
16. Java has method overloading, but no operator overloading just like c++.
17. The String class does use the + and += operators to concatenate strings and String expressions use automatic type conversion,
18. [Java is pass-by-value](http://javadude.com/articles/passbyvalue.htm).
19. Java does not support unsigned integer.