|  |  |
| --- | --- |
| **Java** | **C++** |
| Java does not support pointers, templates, unions, operator overloading, structures etc.  The Java language promoters initially said "No pointers!", but when many programmers questioned how you can work without pointers, the promoters began saying "Restricted pointers." Java supports what it calls "references". References act a lot like pointers in C++ languages but you cannot perform arithmetic on pointers in Java. References have types, and they're type-safe. These references cannot be interpreted as raw address and unsafe conversion is not allowed. | C++ supports structures, unions, templates, operator overloading, pointers and pointer arithmetic. |
| Java support automatic garbage collection. It does not support destructors as C++ does. | C++ support destructors, which is automatically invoked when the object is destroyed. |
| Java does not support conditional compilation and inclusion. | Conditional inclusion (#ifdef #ifndef type) is one of the main features of C++. |
| Java has built in support for threads. In Java, there is a Thread class that you inherit to create a new thread and override the [RUN[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://cs-fundamentals.com/tech-interview/java/differences-between-java-and-cpp.php#21663725)](http://cs-fundamentals.com/tech-interview/java/differences-between-java-and-cpp.php#21663725)) method. | C++ has no built in support for threads. C++ relies on non-standard third-party libraries for thread support. |
| Java does not support default arguments. There is no scope resolution operator (::) in Java. The method definitions must always occur within a class, so there is no need for scope resolution there either. | C++ supports default arguments. C++ has scope resolution operator (::) which is used to to define a method outside a class and to access a global variable within from the scope where a local variable also exists with the same name. |
| There is no *goto* statement in Java. The keywords const and goto are reserved, even though they are not used. | C++ has *goto* statement. However, it is not considered good practice to use of*goto* statement. |
| Java doesn't provide multiple inheritance, at least not in the same sense that C++ does. | C++ does support multiple inheritance. The keyword virtual is used to resolve ambiguities during multiple inheritance if there is any. |
| Exception handling in Java is different because there are no destructors. Also, in Java, try/catch must be defined if the function declares that it may throw an exception. | While in C++, you may not include the try/catch even if the function throws an exception. |
| Java has method overloading, but no operator overloading. The String class does use the + and += operators to concatenate strings and Stringexpressions use automatic type conversion, but that's a special built-in case. | C++ supports both method overloading and operator overloading. |
| Java has built-in support for documentation comments (/\*\* ... \*/); therefore, Java source files can contain their own documentation, which is read by a separate tool usually javadoc and reformatted into HTML. This helps keeping documentation maintained in easy way. | C++ does not support documentation comments. |
| Java is interpreted for the most part and hence platform independent. | C++ generates object code and the same code may not [RUN[http://cdncache-a.akamaihd.net/items/it/img/arrow-10x10.png](http://cs-fundamentals.com/tech-interview/java/differences-between-java-and-cpp.php#498057)](http://cs-fundamentals.com/tech-interview/java/differences-between-java-and-cpp.php#498057) on different platforms. |

* **C++** was designed for systems and applications programming, extending the [C programming language](http://en.wikipedia.org/wiki/C_(programming_language)). To this [procedural programming](http://en.wikipedia.org/wiki/Procedural_programming) language designed for efficient execution, C++ has added support for [statically typed](http://en.wikipedia.org/wiki/Static_typing) [object-oriented programming](http://en.wikipedia.org/wiki/Object-oriented_programming), [exception handling](http://en.wikipedia.org/wiki/Exception_handling),[scoped resource management](http://en.wikipedia.org/wiki/RAII), and [generic programming](http://en.wikipedia.org/wiki/Generic_programming), in particular. It also added a [standard library](http://en.wikipedia.org/wiki/C%2B%2B_Standard_Library) which includes generic containers and algorithms.
* **Java**was created initially as an interpreter for printing systems but grew to support[network computing](http://en.wikipedia.org/wiki/Network_computing). It was once used as the base for the "HotJava" thin client system. It relies on a [virtual machine](http://en.wikipedia.org/wiki/Virtual_machine) to be [secure](http://en.wikipedia.org/wiki/Computer_security) and highly [portable](http://en.wikipedia.org/wiki/Porting). It is bundled with an extensive library designed to provide a complete abstraction of the underlying platform. Java is a statically typed object-oriented language that uses similar (but incompatible) syntax to C++. It was designed from scratch with the goal of being easy to use and accessible to a wider audience. It includes an extensive documentation called [Javadoc](http://en.wikipedia.org/wiki/Javadoc" \t "_blank).
* Then:

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++their 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.

|  |  |
| --- | --- |
| **C++** | **Java** |
| Extends [C](https://en.wikipedia.org/wiki/C) with [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) and [generic programming](https://en.wikipedia.org/wiki/Generic_programming). C code can most properly be used. | Strongly influenced by C++/C syntax. |
| Compatible with [C](https://en.wikipedia.org/wiki/C_(programming_language)) source code, except for a few [corner cases](https://en.wikipedia.org/wiki/Corner_case). | Provides the [Java Native Interface](https://en.wikipedia.org/wiki/Java_Native_Interface) and recently [Java Native Access](https://en.wikipedia.org/wiki/Java_Native_Access) as a way to directly call C/C++ code. |
| [Write once, compile anywhere](https://en.wikipedia.org/wiki/Write_once,_compile_anywhere) (WOCA). | [Write once, run anywhere](https://en.wikipedia.org/wiki/Write_once,_run_anywhere)/everywhere (WORA/WORE). |
| Allows [procedural programming](https://en.wikipedia.org/wiki/Procedural_programming), [functional programming](https://en.wikipedia.org/wiki/Functional_programming), [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming), [generic programming](https://en.wikipedia.org/wiki/Generic_programming), and [template metaprogramming](https://en.wikipedia.org/wiki/Template_metaprogramming). Favors a mix of paradigms. | Allows [procedural programming](https://en.wikipedia.org/wiki/Procedural_programming), [functional programming](https://en.wikipedia.org/wiki/Functional_programming) (since Java 8) and[generic programming](https://en.wikipedia.org/wiki/Generic_programming) (since Java 5), but strongly encourages the [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm). Includes support for creating [scripting languages](https://en.wikipedia.org/wiki/Scripting_language). |
| Runs as native executable machine code for the target [instruction set](https://en.wikipedia.org/wiki/Instruction_set)(s). | Runs on a [virtual machine](https://en.wikipedia.org/wiki/Java_virtual_machine). |
| Provides object types and type names. Allows reflection via [run-time type information](https://en.wikipedia.org/wiki/Run-time_type_information) (RTTI). | Is [reflective](https://en.wikipedia.org/wiki/Reflection_(computer_programming)), allowing metaprogramming and dynamic code generation at runtime. |
| Has multiple binary compatibility standards (commonly Microsoft (for MSVC compiler) and Itanium/GNU (for almost all other compilers)). | Has one binary compatibility standard, [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) for OS and compiler. |
| Optional automated [bounds checking](https://en.wikipedia.org/wiki/Bounds_checking) (e.g., the at() method in vector andstring containers). | All operations are required to be bound-checked by all compliant distributions of Java. [HotSpot](https://en.wikipedia.org/wiki/HotSpot" \o "HotSpot) can remove bounds checking. |
| Native [unsigned arithmetic](https://en.wikipedia.org/wiki/Unsigned_(arithmetic)) support. | Native unsigned arithmetic unsupported. Java 8 changes some of this, but aspects are unclear.[[1]](https://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B#cite_note-1) |
| Standardized minimum limits for all numerical types, but the actual sizes are implementation-defined. Standardized types are available via the standard library <cstdint>. | Standardized limits and sizes of all primitive types on all platforms. |
| Pointers, references, and pass-by-value are supported for all types (primitive or user-defined). | All types (primitive types and reference types) are always passed by value.[[2]](https://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B#cite_note-2) |
| [Memory management](https://en.wikipedia.org/wiki/Memory_management) can be done [manually](https://en.wikipedia.org/wiki/Manual_memory_management) via new / delete, automatically by scope, or by smart pointers. Supports deterministic destruction of objects.[Garbage collection](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) ABI standardized in C++11, though compilers are not required to implement garbage collection. | Automatic [garbage collection](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)). Supports a non-deterministic finalize() method which use is not recommended.[[3]](https://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B#cite_note-3) |
| [Resource management](https://en.wikipedia.org/wiki/Resource_management_(computing)) can be done manually or by automatic lifetime-based resource management ([RAII](https://en.wikipedia.org/wiki/RAII)). | Resource management must be done manually, or automatically via finalizers, though this is generally discouraged. Has try-with-resources for automatic scope-based resource management (version 7 onwards). |
| Supports classes, structs ([passive data structure](https://en.wikipedia.org/wiki/Passive_data_structure) (PDS) types), and unions, and can allocate them on the [heap](https://en.wikipedia.org/wiki/Dynamic_memory_allocation) or the [stack](https://en.wikipedia.org/wiki/Stack-based_memory_allocation). | Classes are allocated on the [heap](https://en.wikipedia.org/wiki/Dynamic_memory_allocation). [Java SE 6](https://en.wikipedia.org/wiki/Java_version_history#Java_SE_6_Update_14) optimizes with [escape analysis](https://en.wikipedia.org/wiki/Escape_analysis) to allocate some objects on the [stack](https://en.wikipedia.org/wiki/Stack-based_memory_allocation). |
| Allows explicitly overriding types, and some implicit narrowing conversions (for compatibility with C). | Rigid [type safety](https://en.wikipedia.org/wiki/Type_safety) except for widening conversions. |
| The [C++ Standard Library](https://en.wikipedia.org/wiki/C%2B%2B_Standard_Library) was designed to have a limited scope and functions, but includes language support, diagnostics, general utilities, strings, locales, containers, algorithms, [iterators](https://en.wikipedia.org/wiki/Iterator" \l "C.2B.2B" \o "Iterator), numerics, input/output, random number generators, regular expression parsing, threading facilities, type traits (for static type introspection) and Standard C Library. The [Boost library](https://en.wikipedia.org/wiki/Boost_(C%2B%2B_libraries)) offers more functions including network I/O.  A rich amount of third-party libraries exist for GUI and other functions like:[Adaptive Communication Environment](https://en.wikipedia.org/wiki/Adaptive_Communication_Environment) (ACE), [Crypto++](https://en.wikipedia.org/wiki/Crypto%2B%2B), various [XMPP](https://en.wikipedia.org/wiki/XMPP) [Instant Messaging](https://en.wikipedia.org/wiki/Instant_Messaging) (IM) libraries,[[4]](https://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B#cite_note-XMPP_Software_.C2.BB_Libraries-4) [OpenLDAP](https://en.wikipedia.org/wiki/OpenLDAP" \o "OpenLDAP), [Qt](https://en.wikipedia.org/wiki/Qt_(software)), [gtkmm](https://en.wikipedia.org/wiki/Gtkmm" \o "Gtkmm). | The standard library has grown with each release. By version 1.6, the library included support for locales, logging, containers and iterators, algorithms, GUI programming (but not using the system GUI), graphics, multi-threading, networking, platform security, introspection, dynamic class loading, blocking and non-blocking I/O. It provided interfaces or support classes for [XML](https://en.wikipedia.org/wiki/XML), [XSLT](https://en.wikipedia.org/wiki/XSLT), [MIDI](https://en.wikipedia.org/wiki/MIDI), database connectivity, naming services (e.g. [LDAP](https://en.wikipedia.org/wiki/LDAP)), cryptography, security services (e.g. [Kerberos](https://en.wikipedia.org/wiki/Kerberos_(protocol))), print services, and web services. SWT offers an abstraction for platform-specific GUIs. |
| [Operator overloading](https://en.wikipedia.org/wiki/Operator_overloading) for most operators. Preserving meaning (semantics) is highly recommended. | Operators are not overridable. The language overrides + and += for the String class. |
| Single and [Multiple inheritance](https://en.wikipedia.org/wiki/Multiple_inheritance) of classes, including virtual inheritance. | Single inheritance of classes. Supports multiple inheritance via the [Interfaces](https://en.wikipedia.org/wiki/Interface_(Java)" \o "Interface (Java))construct, which is equivalent to a C++ class composed of abstract methods. |
| Compile-time templates. Allows for [Turing complete](https://en.wikipedia.org/wiki/Turing_complete) meta-programming. | [Generics](https://en.wikipedia.org/wiki/Generics_in_Java) are used to achieve basic type-parametrization, but they do not translate from source code to byte code due to the use of [type erasure](https://en.wikipedia.org/wiki/Type_erasure) by the compiler. |
| Function pointers, function objects, lambdas (in [C++11](https://en.wikipedia.org/wiki/C%2B%2B11)), and interfaces. | References to functions achieved via the [reflection](https://en.wikipedia.org/wiki/Reflection_(computer_science)) API. OOP idioms using Interfaces, such as Adapter, Observer, and Listener are generally preferred over direct references to methods. |
| No standard inline documentation mechanism. Third-party software (e.g.[Doxygen](https://en.wikipedia.org/wiki/Doxygen" \o "Doxygen)) exists. | Extensive [Javadoc](https://en.wikipedia.org/wiki/Javadoc" \o "Javadoc) documentation standard on all system classes and methods. |
| const keyword for defining immutable variables and member functions that do not change the object. Const-ness is propagated as a means to enforce, at compile-time, correctness of the code with respect to mutability of objects (see[const-correctness](https://en.wikipedia.org/wiki/Const-correctness)). | final provides a version of const, equivalent to type\* const pointers for objects and const for primitive types. Immutability of object members achieved via read-only interfaces and object encapsulation. |
| Supports the [goto](https://en.wikipedia.org/wiki/Goto" \o "Goto) statement. | Supports labels with loops and statement blocks. |
| Source code can be written to be [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) (can be compiled for [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows" \o "Microsoft Windows),[BSD](https://en.wikipedia.org/wiki/BSD), [Linux](https://en.wikipedia.org/wiki/Linux), [OS X](https://en.wikipedia.org/wiki/OS_X), [Solaris](https://en.wikipedia.org/wiki/Solaris_(operating_system)), etc., without modification) and written to use platform-specific features. Typically compiled into native machine code, must be recompiled for each target platform. | Compiled into byte code for the [JVM](https://en.wikipedia.org/wiki/JVM). Byte code is dependent on the Java platform, but is typically independent of [operating system](https://en.wikipedia.org/wiki/Operating_system) specific features. |