**Differences between Java and others (C and C++):**

1. **Java is a dynamic programming language, but, C and C++ are static programming language:**

If any programming language allows memory allocation at the compilation time, then that programming language is called as Static programming language.

**Example:** C and C++

If any programming language allows memory allocation at run-time, then that programming language is called as Dynamic programming language.

**Example:** Java

In Java programming languages, objects are created at run-time.

1. **# include Vs import:**

|  |  |
| --- | --- |
| ***# include <stdio.h>***  #include directive makes the compiler go to the C/C++ standard library and copy the code from the header files into the program. As a result, the program size increases, thus wasting memory and processor’s time. | ***Import Java.io.\****  import statement makes the JVM go to the Java standard library, execute the code there, and substitute the result into the program. Here, no code is copied and hence no waste of memory or processor’s time. Hence import is an efficient mechanism than #include. |

**#include:**

In C, C++ applications, #include statement is used to include the predefined library available in the form of header files.

**Example:**

#include <stdio.h>  
#include <math.h>  
…

In C and C++ applications, as part of compilation preprocessor will recognize #include statement, when preprocessor encounters #include statement, preprocessor will take the specified header file and search for it. In the C and C++ library, if it is available the processor will load the respective executable code to the memory.

Loading executable code is predefined library at the time of compilation is called as Static Loading.

If we want to include more than one header files in C and C++ applications, then we have to write more than one #include statements.  
Whenever compiler encounters #include, it goes to C/C++ library copies the code from the library into our program where #include is encountered. The problem with this approach is our program size increases and processor time also gets increased.

**Import:**

In Java applications, import statement can be used to include the predefined library available in the form of packages.

**Example:**

import Java.io.\*;

import Java.util.Arraylist;

In Java applications, as part of compilation compiler will recognize all the import statements. When compiler encounter import statements then compiler will take the specified package name and search for it in Java predefined library. If the specified package is not available in predefined library, then compiler will raise an “**error**” like “**package Java.io doesn’t exist**”. If the specified package is available then compiler will not raise any error and compiler will load its bytecode.

At runtime of Java applications, when JVM encounter any .class file from the specified package then only JVM will load the respective class byte code to the memory loading predefined library at the runtime is called “**dynamic loading**”.

In Java applications, if we want to include more than one class of same package then it is not required to specify more than one time import statement, it is sufficient to use ‘\*’ notation.  
Whenever an import statement is encountered jvm /Java compiler goes to corresponding library executes the code there and substitutes the result into our program. As a result, there won’t be any increase in size of program and no wastage of memory will be there.

1. **C and C++ are platform dependent programming languages but Java is a platform independent programming language:**

Execution (Windows OS only)

Compilation  
(Windows OS)

Obj.exe

app.c

If any programming language allows its applications to compile and execute on the same operating system only, that programming language is called as “**Platform dependent programming language**”.

**Example**: C and C++.

If we compile the “C” application on windows operating system, then computer will generate windows specific .exe file, then we must provide the same windows operating system for execution process.

Execution  
(Linux)

jvm

Execution  
(Solaris)

Execution  
(Unix)

jvm

jvm

App.java

O/P

.class

Compilation  
(Windows OS)

If any programming language allows its applications to compile on one operating system and to execute on another operating system then that programming language is called as “Platform independent programming language”.

**Example:** Java

If we compile Java file on windows operating system then compiler will generate .class files with neutral byte code. Here, the generated .class file with neutral byte code. Here, the generated byte code is not directly executable code. It is not responding any operating system representations including windows and it is an intermediate code.

To execute the above neutral byte code is nay operating system, we required the conversion mechanism.

To convert neutral byte code to the respective operating system representations and to provide the required execution, SUN MC systems has provided a separate tool called JVM.

**Note:**Java is a platform independent due to the availability of JVM’s, but JVM is platform dependent.

|  |  |
| --- | --- |
| **.exe file** | **.class file** |
| Available in C and C++ | Available in Java. |
| Platform dependent file. | Platform independent file. |
| Includes executable code. | Includes only intermediate files |

**Note:**In Java applications, we are about to see only Java files and .class files. We are unable to see executable files.

**Note:**If we want to provide native methods in Java applications and if we use ‘C’ and ‘C++’ as non-Java language to implement native methods, then we have to provide the .exe files in Java applications explicitly. In Java applications, .Java or .class files can never be converted to .exe files.

**Note:**Native method is a method declared in Java but implemented in non-Java programming languages like C, C+….

1. **Multiple-inheritance is not possible in Java:**

Multiple Inheritance

Single Inheritance

**CLASS C:**i=i+10; j=j+10;  
m1(); m2();

extends

**CLASS B:**  
int j=20;  
void m1(){  
---  
x implementation;  
---}

**CLASS A:**  
int i=10;  
void m1(){  
---  
x implementation;  
---}

**CLASS B:**  
i=i+10;  
m1();

**CLASS A:**  
int i=10;  
void m1(){  
---  
}

**Inheritance :**The process of getting variables and methods from one class to another class is called as inheritance.  
**There are two types of inheritance at basic level:**

**1)** Single Inheritance.

2) Multiple Inheritances.

**Single-Inheritance:**

The process of getting variables and methods from only one super class to one or more number of subclass is called as **Single Inheritance**.

**Multiple-Inheritance:**

The process of getting variables and methods from more than one super-class to one or more number of sub-classes is called **Multiple Inheritance**.

If we declare some variables with different values and some method with different implementation at both the super class is multiple inheritance, then we will get confusion while accessing that variable method at a subclass. To eliminate this confusion, Java technology has eliminated multiple-inheritance.

**Note:** To eliminate multiple inheritance, Java technology has prepared “extends” keyword to allow only superclass name, not to allow more than one super class name.

**5. Pointers are not possible in Java:**

**Pointer is a variable; it’s able to refer a block of memory by storing its starting address location.**

Pointer variables will be recognized and initialized at the time of compilation.

Pointers are not possible in Java, because pointer variables should require static memory allocation but Java follows dynamic memory allocation.

Pointer variables are supported by platform dependent programming languages but Java is platform independent programming language.

Pointers will reduce security for the application data but Java is more secure programming language, it has to provide very good security for the application data.

Multi-level pointers will increase confusion to the developer. But, Java is a simple programming language it shouldn’t provide confusion to the developer.

**Why pointers are eliminated from Java?**

1. Pointers lead to confusion for a programmer.
2. Pointers may crash a program easily, for example, when we add two pointers, the program crashes immediately. The same thing could also happen when we forgot to free the memory allotted to a variable and reallot it to some other variable.
3. Pointers break security. Using pointers, harmful programs like virus and other hacking programs can be developed. Because of the above reasons, pointers have been eliminated from Java.

**What is the difference between pointer variable and reference variable ?**

|  |  |
| --- | --- |
| **Pointer Variable** | **Reference Variable** |
| Available in C and C++ | Available in Java. |
| Able to refer memory on the basis of address locations. | Able to refer object memory on the basis of reference values. |
| Should require static memory allocation. | Should require dynamic memory allocation. |

**Note:** Object reference value is the hexa-decimal format of object hash code. Object hash code is an integer value provided by heap manager at the time of creating object.

**Note:** Even though native methods are used in Java,even though C programming language is used as native language still pointers are used in “C” implementation only not in Java implementation.

1. **“Call by Reference” is not possible in Java :**

In any programming language, if we pass address location as parameters to the methods then the parameter passing mechanism is “Call By Reference”.

In C and C++, if we pass pointer variables as parameters to methods, then the internal parameters passing mechanism is “call by reference”, because pointer variables will store address locations.

In Java, if we pass reference variables as parameters to methods, still the parameter passing mechanism is “Call by value” only, not call by reference, because in Java reference variable will not store address locations. Reference variables will store identifies in the form of reference values.

1. **Destructors are not possible in Java**

In any object oriented programming language, it is common to perform the operations like creating object, with the data and destroying the objects.

In all the OOP language like C++, we will use constructors to create objects and destructors to destroy that objects.

In case of Java applications, developers has to take explicit responsibility only to create objects not to destroy objects, because in Java applications, there is an automatic object destructor in the form of a tool called as “Garbage Collector”.

In Java applications, Garbage Collector will destroy all the objects which are created by developers, it is not required to use destructors. So, that Java has not provided destructor feature.

1. Preprocessor is available in C and C++ applications but it is not available in Java.
2. **Operator overloading is not possible in Java:**

*class A  
< void add(int I, int j)  
 <  
 …  
 >  
 void add(float f1, float f2)  
 <  
 …  
 >*

Static Polymorphism

Operator Overloading

Method overloading

Dynamic Polymorphism

Overloading

Overriding

Polymorphism

Declaring single operator with more than one operation or functionality, then that operator is called Operator Overloading.

**Operator overloading is not possible in Java, because:**

a) It is rare requirement in Java applications development.  
b) It will increase confusion to the developers.

**Note:**

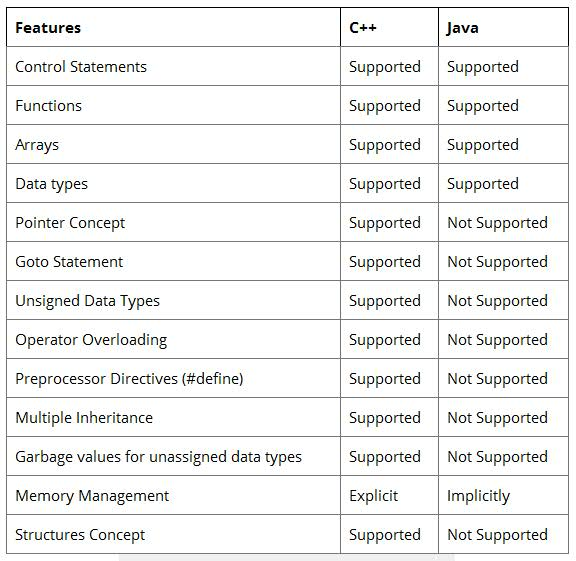
Java technology has defined some fixed number of operators as overloaded operator as per its requirement, but it is not providing any environment to perform operator overloading explicitly at developers level.

**Example:**  
int a=10;  
int b=20;

int c=a+b; 🡪+ **Operator performs arithmetic addition**.  
String s1=”abc”;  
String s2=”def”;  
String s3=s1+s2; 🡪 + **Operator performs string concatenation**.  
  
**C++ vs Java:**

|  |  |
| --- | --- |
| **C++** | **Java** |
| C++ is not a purely object-oriented programming language, since it is possible to write C++ programs without using a class or an object. | Java is purely an object-oriented programming language, since it is not possible to write Java program without using at least one class. |
| Pointers are available in C++. | We cannot create and use pointers in Java. |
| Allocation and deallocation memory is the responsibility of the programmer. | Allocation and deallocation of memory will be taken care by JVM. |
| C++ has goto statement. | Java doesn’t have goto statement. |
| Automatic casting is available in C++. | In some cases, implicit casting is available. But it is advisable that the programmer should use casting whenever required. |
| There are 3 access specifiers in C++: public, private, and protected. | Java supports 4 access specifiers: private, public, protected and default. |
| There are constructors and destructors in C++. | Only constructors are there in Java. No destructors are available in this language. |

**Java features Vs C++ Features:**



**How Java Related to C# ?**

1. After the creation of Java, **Microsoft developed the C# language** and C# is closely related to Java.
2. Many of C#’s features directly parallel Java. Both Java and C# share the same general C++-style syntax, support distributed programming, and utilize the same object model.
3. Though there are some differences between Java and C#, but the overall feel of these languages is very similar.
4. If you already know C#, then learning Java will be easy and vice versa
5. Java and C# are optimized for two different types of computing environments.
6. C# and Java Both Languages are drew from C++.
7. Both Languages are capable of creating cross platform portable program code.

## Java Vs C Sharp

Here are some differences between Java and C Sharp.

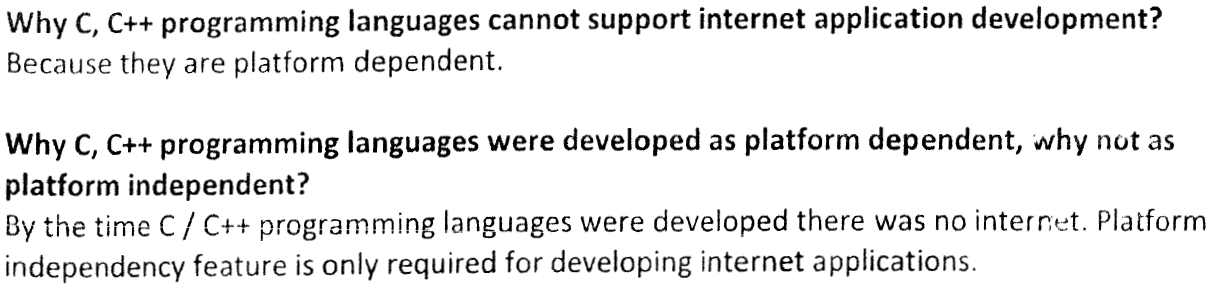
| **Point** | **Java** | **C#** |
| --- | --- | --- |
| Development | Sun Microsystem | Microsoft |
| Development Year | 1995 | 2000 |
| Data Types | Less Primitive DT | More Primitive DT |
| Structs Concept | Not Supported | Supported |
| Switch Case | String in Switch Not Allowed | String in Switch Allowed |
| Delegates | Absent | Supported |

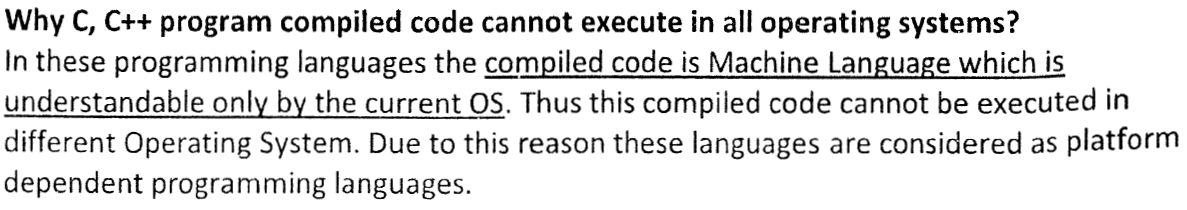
**Features of C# absent in Java**

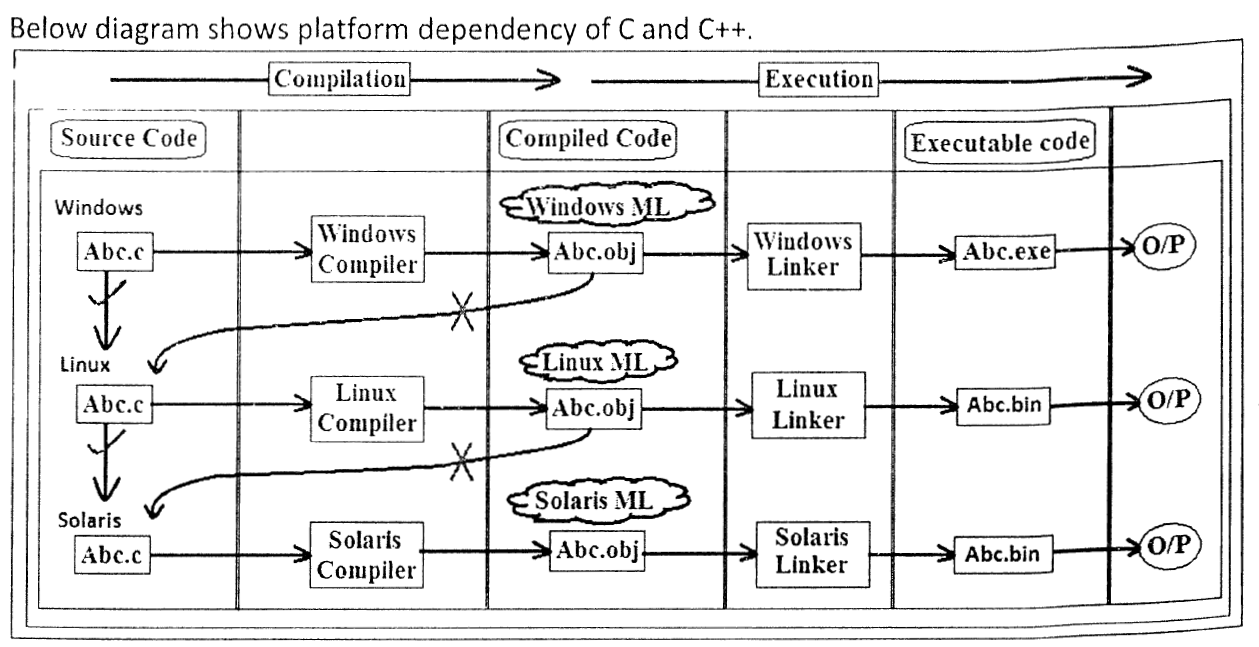
* C# includes more primitive types and the functionality to catch arithmetic exceptions.
* Includes a large number of notational conveniences over Java, many of which, such as operator overloading and user-defined casts, are already familiar to the large community of C++ programmers.
* Event handling is a "first class citizen"—it is part of the language itself.
* Allows the definition of "structs", which are similar to classes but may be allocated on the stack (unlike instances of classes in C# and Java).
* C# implements properties as part of the language syntax.
* C# allows switch statements to operate on strings.
* C# allows anonymous methods providing closure functionality.
* C# allows iterator that employs co-routines via a functional-style yield keyword.
* C# has support for output parameters, aiding in the return of multiple values, a feature shared by C++ and SQL.
* C# has the ability to alias namespaces.
* C# has "Explicit Member Implementation" which allows a class to specifically implement methods of an interface, separate from its own class methods. This allows it also to implement two different interfaces which happen to have a method of the same name. The methods of an interface do not need to be public; they can be made to be accessible only via that interface.
* C# provides integration with COM.
* Following the example of C and C++, C# allows call by reference for primitive and reference types.

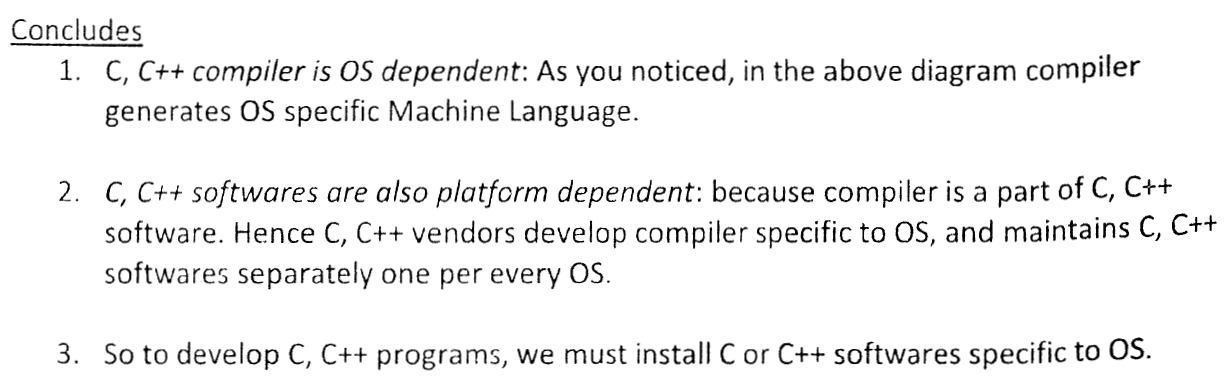
**Features of Java absent in C#**

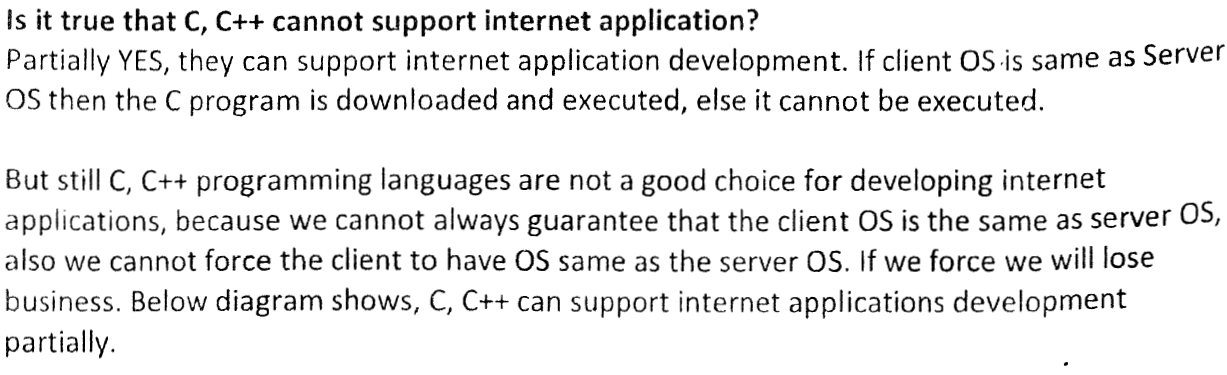
* Java's strictfp keyword guarantees that the result of floating point operations remain the same across platforms.
* Java supports checked exceptions for better enforcement of error trapping and handling.

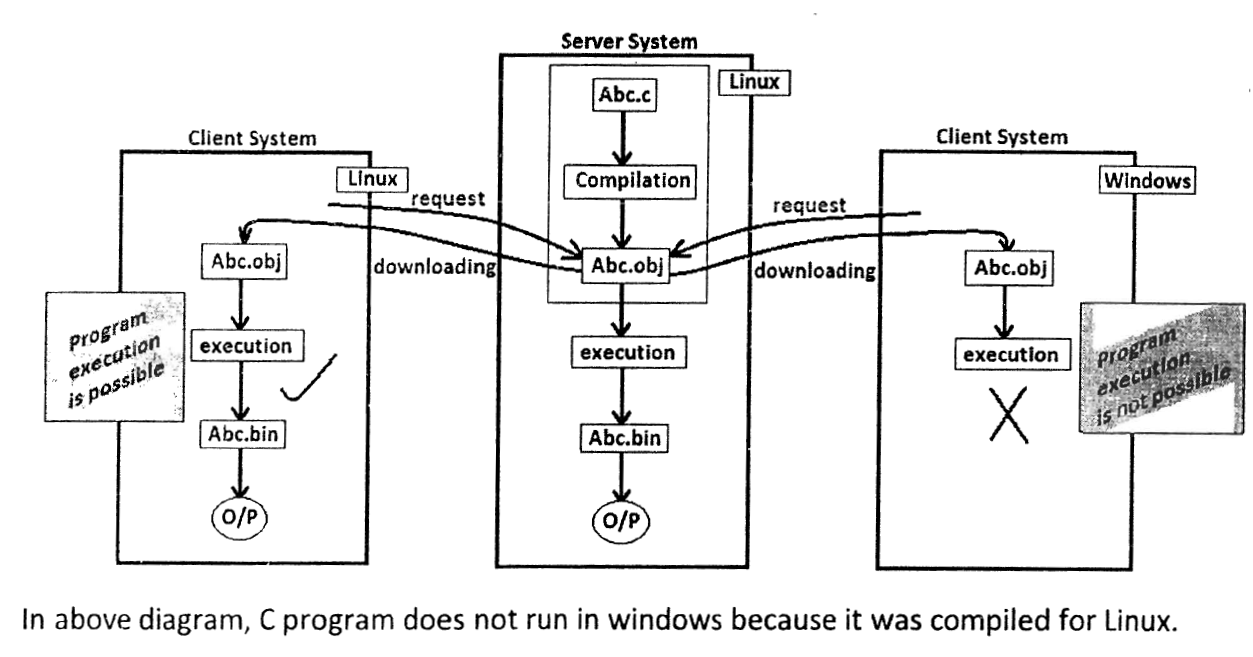












**C Programming vs. Java Programming**

|  |  |  |
| --- | --- | --- |
| Thing | C | Java |
| **type of language** | function oriented | object oriented |
| **basic programming unit** | function | class = ADT |
| **portability of source code** | possible with discipline | yes |
| **portability of compiled code** | no, recompile for each architecture | yes, bytecode is "write once, run anywhere" |
| **security** | limited | built-in to language |
| **compilation** | gcc hello.c creates machine language code | javac Hello.java creates Java virtual machine language bytecode |
| **linking in the Math library** | gcc -lm calculate.c | no special flags needed |
| **joint compilation** | gcc main.c helper1.c helper2.c | javac Main.java - any dependent files are automatically re-compiled if needed |
| **execution** | a.out loads and executes program | java Hello interprets byte code |
| **hello, world** | #include<stdio.h> int main(void) {    printf("Hello\n");    return 0; } | public class HelloWorld {    public static void main(String[] args) {         System.out.println("Hello");    } } |
| **integer types** | int usually 32 bit 2's complement; long usually 32 bit 2's complement | int is 32 bit 2's complement; long is 64 bit 2's complement |
| **floating point types** | float usually 32 bit; double usually 64 bit | float is 32 bit IEEE 754 binary floating point; double is 64 bit IEEE 754 |
| **boolean type** | use int: 0 for false, nonzero for true | boolean is its own type - stores value true or false |
| **character type** | char is usually 8 bit ASCII | char is 16 bit UNICODE |
| **for loops** | for (i = 0; i < N; i++) | for (int i = 0; i < N; i++) |
| **array declarations** | int \*a = malloc(N \* sizeof(\*a)); | int[] a = new int[N]; |
| **array size** | arrays don't know their own size | a.length |
| **strings** | '\0'-terminated character array | built-in immutable String data type |
| **accessing a library** | #include <stdio.h> | import java.io.File; |
| **accessing a library function** | #include "math.h" x = sqrt(2.2);  all function and variables names are global | x = Math.sqrt(2.2);  functions have different namespaces |
| **printing to standard output** | printf("sum = %d", x); | System.out.println("sum = " + x); |
| **formatted printing** | printf("avg = %3.2f", avg); | System.out.printf("avg = %3.2f", avg) |
| **reading from stdin** | scanf("%d", &x); | Java library support, but easier to use our library int x = StdIn.readInt(); |
| **memory address** | pointer | reference |
| **manipulating pointers** | \*, &, + | no direct manipulation permitted |
| **functions** | int max(int a, int b) | public static int max(int a, int b) |
| **pass-by-value** | primitive data types, structs, and pointers are passed by value; array decays to pointer | all primitive data types and references (which includes arrays), are passed by value |
| **defining a data structure** | struct | class - key difference is language support for defining methods to manipulate data |
| **accessing a data structure** | a.numerator for elements | a.numerator for instance variables, c = a.plus(b) for methods |
| **pointer chasing** | x->left->right | x.left.right |
| **allocating memory** | malloc | new |
| **de-allocating memory** | free | automatic garbage collection |
| **memory allocation of data structures and arrays** | heap, stack, data, or bss | heap |
| **buffer overflow** | segmentation fault, core dump, unpredicatable program | checked run-time error exception |
| **declaring constants** | const and #define | final |
| **variable auto-initialization** | not guaranteed | instance variables (and array elements) initialized to 0, null, or false, compile-time error to access uninitialized variables |
| **data hiding** | opaque pointers and static | private |
| **interface method** | non-static function | public method |
| **data type for generic item** | void \* | Object |
| **casting** | anything goes | checked exception at run-time or compile-time |
| **demotions** | automatic, but might lose precision | must explicitly cast, e.g., to convert from long to int |
| **polymorphism** | union | inheritence |
| **overloading** | no | yes for methods, no for operators |
| **graphics** | use external libraries | Java library support, use our standard drawing library |
| **null** | NULL | null |
| **enumeration** | enum | typesafe enum |
| **preprocessor** | yes | no |
| **variable declaration** | at beginning of a block | before you use it |
| **variable naming conventions** | sum\_of\_squares | sumOfSquares |
| **commenting** | /\* \*/ | /\* \*/ or // |
| **file naming conventions** | stack.c, stack.h | Stack.java - file name matches name of class |
| **callbacks** | pointers to global functions | use interfaces for [commmand dispatching](http://developer.java.sun.com/developer/TechTips/1997/tt1021.html#tip2) |
| **variable number of arguments** | varargs | String ... |
| **assertions** | assert | assert |
| **exit and return value to OS** | exit(1) | System.exit(1) |

**Here are the major differences between C and JAVA**.

**1. JAVA is Object-Oriented while C is procedural. Different Paradigms, that is.**

Most differences between the features of the two languages arise due to the use of different [programming paradigms](http://durofy.com/programming/c-as-a-multi-paradigm-programming-language/). C breaks down to functions while JAVA breaks down to Objects. C is more procedure-oriented while JAVA is data-oriented.

**2. Java is an Interpreted language while C is a compiled language.**

We all know what a compiler does. It takes your code & translates it into something the machine can understand-that is to say-0 & 1’s-the machine-level code. That’s exactly what happens with our C code-it gets ‘compiled’. While with JAVA, the code is first transformed to what is called the bytecode. This bytecode is then executed by the JVM (Java Virtual Machine). For the same reason, JAVA code is more portable.

**3. C is a low-level language while JAVA is a high-level language.**

C is a low-level language (difficult interpretation for the user, closer significance to the machine-level code) while JAVA is a high-level language (abstracted from the machine-level details, closer significance to the program itself).

**4. C uses the top-down {sharp & smooth} approach while JAVA uses the bottom-up {on the rocks} approach.**

In C, formulating the program begins by defining the whole and then splitting them into smaller elements. JAVA(and C++ and other OOP languages) follows the bottom-up approach where the smaller elements combine together to form the whole.

**5. Pointer go *backstage* in JAVA while C requires explicit handling of pointers.**

When it comes to JAVA, we don’t need the \*’s & &’s to deal with pointers & their addressing. More formally, there is no pointer syntax required in JAVA. It does what it needs to do. While in JAVA, we do create references for objects.

**6. The Behind-the-scenes Memory Management with JAVA & The User-Based Memory Management in C.**

Remember ‘malloc’ & ‘free’? Those are the library calls used in C to allocate & free chunks of memory for specific data (specified using the keyword ‘sizeof’). Hence in C, the memory is managed by the user while JAVA uses a garbage collector that deletes the objects that no longer have any references to them.

**7. JAVA supports Method Overloading while C does not support overloading at all.**

JAVA supports function or method overloading-that is we can have two or more functions with the same name (with certain varying parameters like return types to allow the machine to differentiate between them). That is to say, we can overload methods with the *same name having different method signatures*. JAVA(unlike C++), does not support Operator Overloading while C does not allow overloading at all.

**8. Unlike C, JAVA does not support Preprocessors, & does not really them.**

The preprocessor directives like #include & #define, etc are considered one of the most essential elements of C programming. However, there are no preprocessors in JAVA. JAVA uses other alternatives for the preprocessors. For instance, *public static final* is used instead of the #define preprocessor. Java maps class names to a directory and file structure instead of the #include used to include files in C.

**9. The standard Input & Output Functions.**

Although this difference might not hold any conceptual (intuitive) significance, but it’s maybe just the tradition. C uses the printf & scanf functions as its standard input & output while JAVA uses the System.out.print & System.in.read functions.

**10. Exception Handling in JAVA and the errors & crashes in C.**

When an error occurs in a Java program it results in an exception being thrown. It can then be handled using various exception handling techniques. While in C, if there’s an error, there IS an error.