Table of Contents

[Differences between Java, C and C++ 2](#_Toc64603645)

[1. Static and dynamic programming language 2](#_Toc64603646)

[2. Pre-Processor is required in C and C++, but not required in Java 2](#_Toc64603647)

[3. Platform Dependent VS Platform Independent 4](#_Toc64603648)

[4. Pointers 6](#_Toc64603649)

[5. Multiple inheritance 8](#_Toc64603650)

[6. Operator Overloading 10](#_Toc64603651)

[7. Destructors 12](#_Toc64603652)

[8. Call By Value Vs CallBy Reference 13](#_Toc64603653)

[9.Memory Allocation 13](#_Toc64603654)

C++ vs Java

There are many differences and similarities between the C++ programming language and Java. A list of top differences between C++ and Java are given below:

|  |  |  |
| --- | --- | --- |
| **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. |

Note

* Java doesn't support default arguments like C++.
* Java does not support header files like C++. Java uses the import keyword to include different classes and methods.

# Differences between Java, C and C++

## Static and dynamic programming language

As we know that there are two types of programming language:

Dynamically typed programming language

Statically typed Programming Language

C and C++ are static programming languages, but Java is a dynamic programming language.

**Static programming language**

If any programming language allows memory allocation for primitive data types at compilation time then that programming language is called as static programming language (examples C and C++).

In C and C++ applications memory will be allocated for primitive data types at compilation time and not at runtime (off course In C and C++, calloc () and malloc () functions allows dynamic memory allocation, but in general C and C++ applications are static programming language).

**Dynamic programming language**

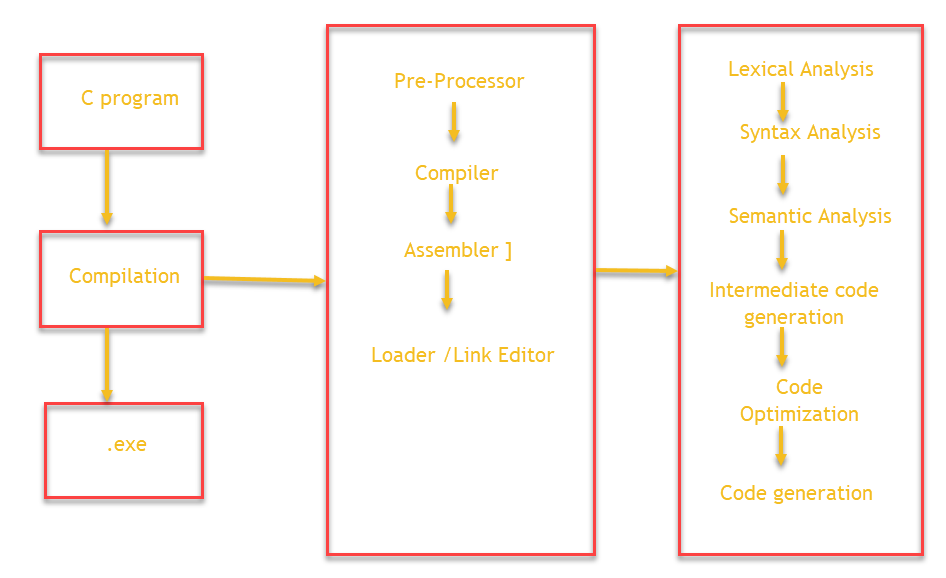
If any programming language allows memory allocation for primitive data types at run time then that programming language is called as dynamic programming language (example Java).

In Java applications, memory will be allocated for primitive data types at runtime only.

In Java memory is allocated for objects at runtime only, which means that memory is allocated for primitive types at run time only and not compilation time

## Pre-Processor is required in C and C++, but not required in Java

What is preprocessor and where it is located.



Job of preprocessor:

In C, we use header files in application

#include <stdio.h>

#include <math.h>

Preprocessor will recognize all #include statements in the C++ application.

It will check take all header file names from #include

It will search for specified / required header files in C and C++ software’s,

if the specified header files are not present then error will be generated

if the specified header files exists then preprocessor will load all the required libraries.

Preprocessor will load all the specified header files to the memory.

Loading pre-defined libraries at compilation time is called static loading.

Conclusion: In C and C++, preprocessor is required to recognize #include statements in order to load header file content to the memory.

Why preprocessor is not required in Java.

Compiler responsibilities

In java classes and interfaces are in the form on packages (like java.lang.\*, java.sql.\*, …)

We use import statements in Java like import java.lang.\*

When we compile java program, the job of the compiler is

Compiler will recognize all import statements

Compiler will take the specified packages from import statements

Compiler will go to Java software, where compiler will check whether these packages exists or not.

If packages do not exist then the compiler will raise an error

If packages exists then the compiler will not load the package content to memory.

When we execute Java application, job of JVM is

When the predefined classes and interfaces are identified by JVM, then JVM will load respective classes and interfaces into the memory.

In java applications, the predefined library is loaded by JVM as per requirement. This type of loading pre-defined libraries is called dynamic loading.

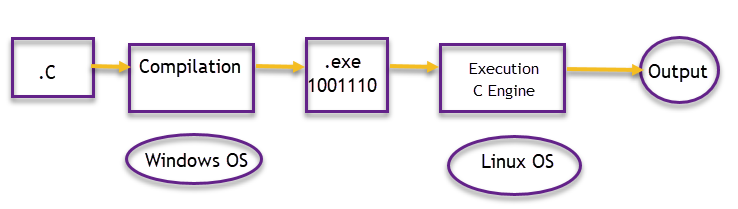
In java preprocessor is not required. In place of preprocessor, we have compiler and JVM.

<https://www.tutorialspoint.com/compiler_design/index.htm>

## Platform Dependent VS Platform Independent

C and C++ are platform dependent programming languages, but java is platform independent programming language

If any programming language allows its applications to perform compilation and execution on same operating systems, then that program is called is platform dependent language (example are Cand C++).



Program compiled in Windows OS: It generated .exe file (which contains sequence of 0’s and 1’s).

.Exe contains directly executable code and it has windows representation.

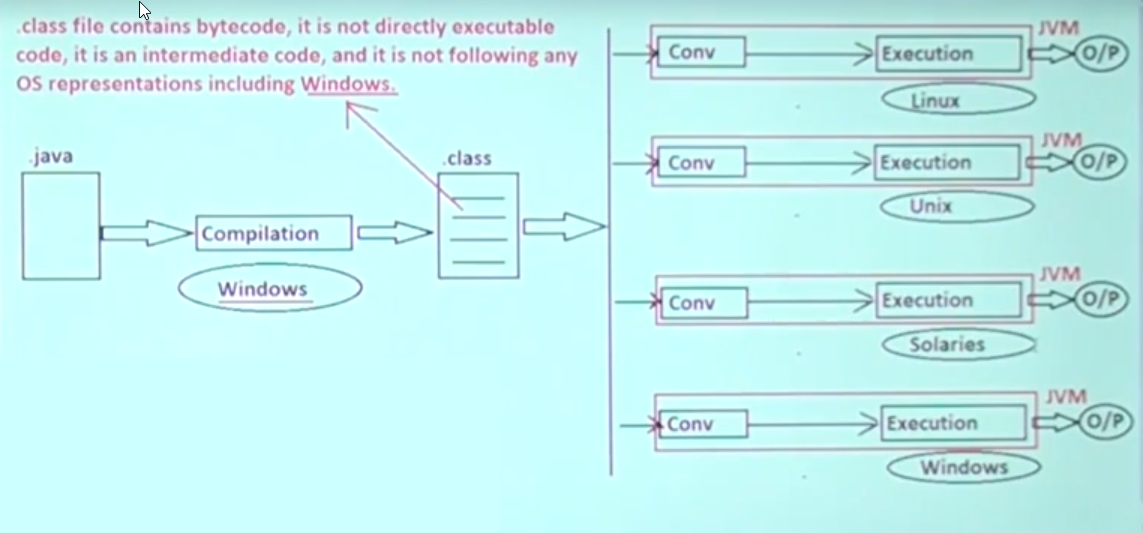
In above diagram c engine requires Linux representation code for execution and so we can’t execute windows representation.

So it becomes mandatory to compile and execute the code in same operating system in case of C and C++.

So how java becomes platform independent programming language?

If any programming language allows its applications to perform compilation and execution on other or same operating systems, then that program is called is platform dependent language (example Java).

The job of convertors and executors is to take intermediate (neutral) byte code and convert to OS executable code.



JVM (Java virtual machine) contains both conversion and execution mechanism. We have different JVM (JVM is platform dependent). Java software is platform dependent.

**Jvm is platform dependent** because we have different **JVM** for different operating system. **JVM** is one kind of interface or middleware between **OS**(Operating Systems) and java language. **JVM** provides the environment to execute the java file(. ... So **JVM is platform dependent.**

A class file contains the byte code generated from your original java source.

Byte code can only be interpreted by a JVM. An exe doesn’t need to be

interpreted since it's in machine code and the computer can read it directly.

|  |  |
| --- | --- |
| .exe | .class |
| It is in C and C++ | It is in Java |
| Contains executable code | Contains byte code (intermediate code). Intermediate code is executed by converters in JVM |
| Platform dependent | Platform independent |
| Less security (most viruses exist in form for .exe) | It is secured, as it doesn’t contain executable code |

## Pointers

Pointers are in C and C++, but not in Java.

Pointer is a variable; it is able to store address locations of the data structures. A data structure may be an array, a variable, a struct, another pointer variable ..etc.

In general pointer variables are recognized and initialized at the time of compilation

|  |  |  |
| --- | --- | --- |
| Int a=10; | a=10 | 2 bytes of memory will be allocated to a |
| Int \*p; | A is stored in some memory address. Let’s assume 1010. | p value will referring a address location (mean 1010) |
| p=&a; |  |  |

In above p is a pointer.

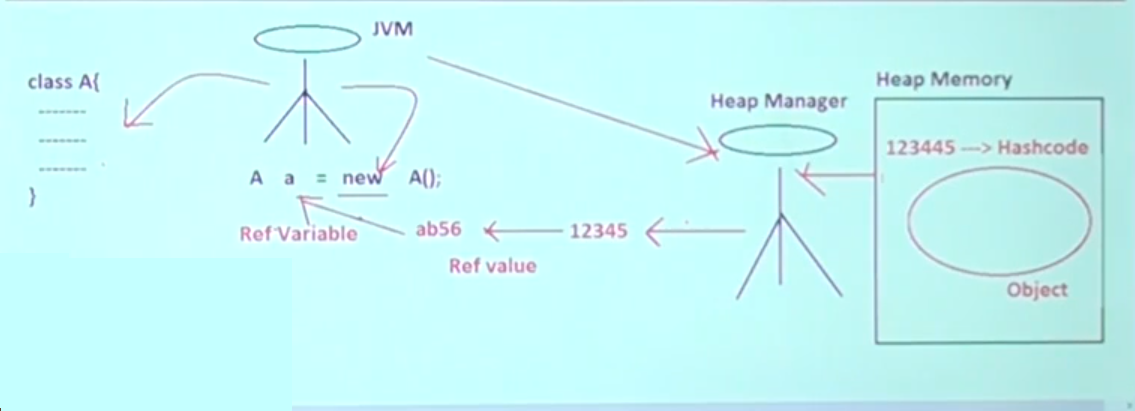
Pointer variable will be recognized and initialized.

Why pointer variables are not in Java?

1. Pointer variables required memory allocation at compilation time, that is pointer variables require static memory allocation, but Java is following dynamic memory allocation.
2. Pointers are supported by static programming languages only, but Java is a dynamic programming language. C and C++ are static programming language, but Java is dynamic programming language.
3. Pointer variables are suitable in platform dependent programming languages, but Java is program independent programming language.
4. Pointer variables provide less security for application data, but Java is secured programming language and it has to provide good security for application data.
5. Pointers concept is a bit confusing feature, but Java is simple programming language. In pointers, one pointer may refer another pointer, if 10 pointers refer each other, then it confuses a developer (which pointer is referring)

Q) in C and C++ applications a variable is referring a block of memory, so that a variable is pointer variable, similarly in Java applications, when we create an object for a particular variable, there also a variable is referring a block of memory (Object). Then why don’t we call that variable as pointer variable, and how we can say pointers are not existed in Java.

Ans)



JVM convers the hash code value into hexadecimal value. The reference variable is that hexadecimal value. reference variable is not address location.

Q) What is the difference between pointer variable and reference variable?

Pointer variable are the variables that refer a block of memory by storing address locations.

Reference variable are the variables that refer a block of memory (Object) by storing object reference value, where object reference value is a hexadecimal form of hash code. Hash code is a unique identity provided by heap manager to recognize the object individually.

Pointer variable refer static memory allocation, reference variable refers dynamic memory allocation

Pointer variables are in C and C++, but reference variables referred in Java.

## 5. Multiple inheritance

Multiple inheritance in Java is achieved through interfaces and not through classes

**public** **class** ClassA {

}

**public** **class** ClassB {

}

Below is not allowed (multiple inheritance in classes)

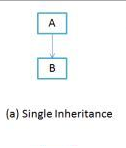
**public** **class** ChildClass **extends** ClassA,ClassB {

}

inheritance Types

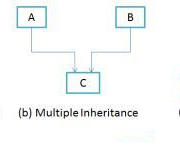
1. Single inheritance

**public** **class** ClassA **implements** IOne {



1. Multiple Inheritance

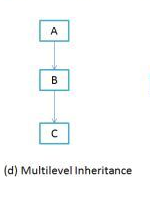
**public** **class** ClassA **implements** IOne,ITwo {



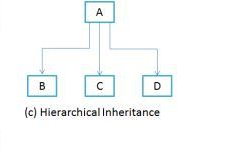
1. Multilevel inheritance

**public** **interface** IOne **extends** SuperInterface {

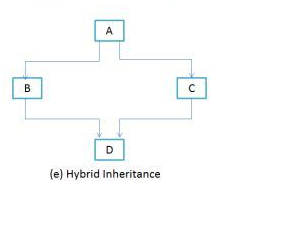
**public** **class** ClassA **implements** IOne {



1. Hierarchical inheritance



1. Hybrid inheritance



===Diamond Problem for Hybrid inheritance=========

**public** **interface** SuperInterface {

**void** m2();

}

**public** **interface** IOne **extends** SuperInterface {

**default** **void** m1() {

System.***out***.println("IOne.m1()");

}

}

**public** **interface** ITwo **extends** SuperInterface {

**default** **void** m1() {

System.***out***.println("IOne.m1()");

}

}

**public** **class** Impl **implements** IOne, ITwo {

/\*

\* Must override the default method (as it exists in both interfaces). Java

\* solves diamond problem by overriding and referring default method interface

\* name

\*/

@Override

**public** **void** m1() {

IOne.**super**.m1();

}

@Override

**public** **void** m2() {

System.***out***.println("Impl.m2");

}

}

## 6. Operator Overloading

Operator overloading: If we declare any operator with more than one functionality then it is called operator overloading.

**public** **class** Example {

**public** **static** **void** main(String[] args) {

**int** i = 10;

**int** j = 20;

String s1 = "Hello";

String s2 = " World!";

**int** k = i + j; // + is used for adding numbers

String output = s1 + s2; // + is used for concatenating strings

/\*

\* Here + is used as operator overloading (by Java internals.

\* A developer can't overload an operator

\*/

System.***out***.println(k);

System.***out***.println(output);

}

}

Why operator overloading is not supported in Java?

1. Simplicity and Cleanliness

The simple and clear design was one of the goals of Java designers. They, just don't want to replicate the language, but wanted to have a clear, truly object-oriented language. Adding Operator overloading would have definitely made the design more complex than without it, and it might have led to the more complex compiler or slows the JVM because it needs to do extra work to identify the actual meaning of operators and reduce the opportunity to optimize the language by guarantee behavior of operators in Java.

1. Avoid Programming Errors

Java doesn't allow user-defined operator overloading, because if you allow a programmer to do operator overloading, they will come up with multiple meanings for same operator, which will make the learning curve of any developer hard and things more confusing and messier.

It’s been observed that there is an increase in programming errors when language supports operator overloading, which in turn increases e development and delivery time.

Since Java and JVM have taken most of the developer's responsibility, in memory management by providing garbage collector, it doesn't really make sense to left this feature to pollute the code, and as a loop hole for programming errors.

1. JVM Complexity

From the JVM perspective, supporting operator overloading is more difficult, and if the same thing can be achieved, by using method overloading in a more intuitive and clean way, it does make sense to not support. A complex JVM, may result in slower JVM, than a relatively simpler JVM, and reduce the opportunity of optimization by taking out guaranteed behavior of operators in Java.

1. Easy Development of Tools

This is an additional benefit of not supporting operator overloading in Java. The omission of operator overloading has kept the language easier to handle and process, which in turn makes it easier to develop the tools, that process the language e.g., IDE or re-factoring tool. Re-factoring tools in Java are far better than C++.

Note: In Java, as per Java internal requirements, Java made some of the operators lie +, \*, % … are declared as overloaded operators with fixed functionalities (not with variable functionalities).

Java has not provided any environment to perform operator overloading explicitly at developer level.

Method overloading

**public** **class** MethodOverloading {

**public** **static** **void** main(String[] args) {

*add*(1, 2);

*add*(1.0, 2.0);

**float** i = 2;

**float** j = 2;

*add*(i, j);

}

**public** **static** **void** add(**int** i, **int** j) {

System.***out***.println("int add " + (i + j));

}

**public** **static** **void** add(**float** i, **float** j) {

System.***out***.println("float add " + (i + j));

}

**public** **static** **void** add(**double** i, **double** j) {

System.***out***.println("double add " + (i + j));

}

}

## 7. Destructors

C++ and Java are object-oriented programming languages, but not C.

A destructor is a special member function that works just opposite to constructor, unlike constructors that are used for initializing an object, destructors destroy (or delete) the object.

Java: In java as part of JVM, we have garbage collector, which internally destroys unused objects. Makes developers like easy.

C++: There is no garbage collector, so developers must take explicit responsibility to destroy the objects.

Java

A sample c++ program with destructor

**#include** <iostream>

**struct** A {

**int** i;

**A**(**int** i) :

i(i) {

}

**~A**() {

std::cout << "~a" << i << std::endl;

}

};

**int** **main**() {

A a1(1);

A \*p;

{ // nested scope

A a2(2);

p = **new** A(3);

} // a2 out of scope

**delete** p; // calls the destructor of a3

}

## 8. Call By Value Vs CallBy Reference

C and C++ use Call by Value and Call by reference, but Java uses Call by value only.

In any programming language, if we pass primitive data (byte, char, short, int, long, float, double, boolean) as parameters to the method, then the parameter passing mechanism is called as “Call by Value”.

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

In C and C++, if we pass pointer variables as parameters to the methods, the parameter mechanism is called as “Call by Reference”. Pointer variables store address location.

In case of Java, even if we pass object reference variables as parameter to a method, then the parameter passing mechanism is still called “Call by Value” only, because object reference variable does not store address location. Object reference variable stores object reference value, where object reference value is hexadecimal form of hash code. Hash code is an unique identity provided by heap manager in the form of integer value.

## 9.Memory Allocation

Memory allocation for primitive datatypes is different compared between C, C++ and Java.

In C and C++, integers will take 2 byes of memory and characters will take 1 byte of memory, but in Java integers will take 4 bytes of memory and characters will take 2 bytes of memory.

In C and C++, memory allocation for primitive data types is not fixed, it is variable. It depends on operation system we use.

In Java, memory allocation for the primitive data types is fixed, irrespective of operating system we use.

Q) In C and C++, characters take 1 byte of memory, the why java takes 2 bytes for characters?

In C, C++ characters are stored in ASCII format (which takes 1 byte), but in Java all characters are stored in form of UNICODE (which takes 2 bytes)

Q) What is UNICODE and what is its requirement in Java?

A) INICODE is a character representation. Represents all the alphabets from all the natural language like English, Spanish, Hindi …et. It provides good international support (I 18N) in Java support

