1. **JAVA HISTORY**

1991 -> SUN Microsystems -> PL -> S/W for simple electronics consumer devices like cable TV switch boxes, remote controllers etc

30 member team -> James Gousling and Patrck Naughtan

Project: Green Project

PL -> 1. Simple PL

2. Tight Coded PL

3. Arch Neutral PL

1. Simple PL ->

* Less Execution Time: More Performance
* Less Memory Consumption: Less Cost for the product
* Less power consumption: Less Maintenance cost

2. Tight Coded PL ->

In C PL:

Stack:

1. PUSH -> 20 LOC
2. POP -> 20 LOC
3. PEEK-> 20 LOC

In Java PL:

Stack:

Stack s = new Stack();

s.push(‘AAA’);

s.pop();

s.peek();

A PL takes less no of intersections (LOC) for implementing a requirement, then it is called tight coded PL

1. Arch Neutral PL ->

PL -> Execute functionalities in all the H/W Arch in the market

1991, PASCAL, Nicolas Wirth -> Procedure Oriented Programming Language(POPL)

1992 -> Dec -> PL -> OAK -> JAVA

\*7 Product -> Remote Controller

1993 and half of 1994 -> No Customers

Mid 1994 -> www as part of internet -> Browser

Mid 1994 -> Musaik -> Mark Anderson -> UG Student (Illinois’ University)

Browser -> Patrick Naughtan and Janathan Paine

1994 and 1995, Dec -> Hot Java Browser

JAVA -> Applets

1996, JAN 23rd: SUN Microsystems

JDK 1.0

JDK 1.1

JDK 1.2 -> More Enchancements

JDK 1.3

JDK 1.4

JDK 1.5 -> More Enhancements

JAVA 6 Initial Release

ORACLE Corporation

JAVA 6 Later Releases

JAVA 7

JAVA 8 - > More Enhancements

JAVA 9 -> More Enhancements [JPMS]

JAVA 10 -> No Long Term Support

JAVA 11 -> More Enhancements

1. **DIFFERENCES BETWEEN JAVA AND OTHERS[C AND C++]:**

**STATIC VS DYNAMIC**

1. C and C++ are Static Programming Languages but JAVA is a Dynamic Programming Language

* If any programming language allows memory allocation for primitive data types at compilation time, then that programming language is called Static Programming Language.
  + E.g. – C and C++
    - In C and C++ applications, memory will be allocated for primitive data types at compilation time.
* If any PL allows memory allocation for primitive data types at runtime, then that PL is called as Dynamic PL
  + E.g. – JAVA
    - In Java applications, memory will be allocated for primitive data types at runtime only.

1. Pre-Processor is required in C and C++, but , Pre-Processor is not required in JAVA

.C

1. Lexical Analysis

2. Syntax Analysis

file

6. Code Generation

5. Code Optimization

4. Intermediate Code Gen

3. Semantic Analysis

4. Loader/Link-Editor

3. Assembler

2. Complier

1. Pre-Processor

Execution

.exe

Compilation

Job of the Pre-Processor:

Header Files

E.g. –

* stdio.h
* conio.h
* math.h

---------

---------

#include<> statements

E.g. –

* #include<stdio.h.>
* #include<conio.h>
* #include<math.h>

---------

---------

Job of Pre-Processor:

* It will recognize all the #include<> statements which we provide in C and C++ Applications
* It will take all header files names from #include<> statements
* It will search for specified/required header files in C and C++ Software’s
* If the specified header files are not existed in C and C++ Software’s then error will be generated
* If the specified header files are existed in C and C++ Software’s, then pre-processor will load all the header files to the memory.
* **NOTE:** Loading predefined library at compilation time is called as **Static Loading**

Conclusion:

* In C and C++, Pre-processor is required to recognize #include<> statements in order to load header files content to the memory
* In Java, classes and interfaces in form of packages
  + E.g. –
    - java.io
    - java.util
    - java.sql

----------

import java.io \*;

import java.util.\*;

import java.sql.\*;

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

* Compiler will recognize all import statements
* Compiler will take the specified package names from import statements
* Compiler will go to Java Software, where compiler will check whether these packages are existed or not.
* If the specified packages are not existed in java software then compiler will raise an error like “package xxxxxx does not exist”
* If the specified packages are existed in Java software, then compiler will not raise any error, at the same time, compiler will not load any package content to the memory

When we execute Java applications, Job of the JVM is:

* When the predefined classes and interfaces are identified by JVM, then JVM will load up to the respective classes and interfaces into the memory.
* **NOTE :**
* In java applications, the predefined library is loaded by JVM as per the requirement, this type of loading predefined library at runtime is known as Dynamic Loading.

CONCLUSION:

In java, preprocessor is not required, because, in java, header files and #include statements are not existed.

**(Q) What are the differences between #include statements and import statements?**

**(A)**

1. #include<> statements are existed in C and C++

Import statements are existed in Java

1. #include<> statements are used to include the predefined library existed in header files

Import statements are used to include the predefined library existed in classes and interfaces in the form of packages

1. #include<> statements are recognized by pre-processor

Import statements are recognized by both compiler and JVM.

1. #include<> statements is supporting static loading

Import statements are supporting Dynamic Loading

1. By using single #include<> statements, we are able to include only on header file

E.g. - #include<stdio.h> 🡪 valid

#include<conio.h> 🡪 valid

#include<math.h> 🡪 valid

#include< stdio.h, conio.h, math.h> 🡪 Invalid

By using single import statements we are able to import more than one class or interface of the same package.

E.g. – import java.io.\*;

import java.util.\*;

import java.sql.\*;

1. **Platform Dependant VS Platform Independent**

C and C++ are platform dependant programming languages but JAVA is platform independent programming language

If any programming language allows its applications to perform compilation and execution the same operating system then that programming language is called as platform dependant programming language.

E.g. – C and C++

.C

.exe file contains directly executable code and it is having windows representations.

C-Engine can understand only windows Representation at runtime for execution.

.exe

01010101100101

.C

Execution

C-Engine

Compilation

JVM

If any PL allows its applications to perform compilation on one Operating System and execution is on another Operating System then that PL is called as Platform Independent Programming Language.

Execution

E.g. - JAVA

Convertors/Trans

.

JVM

Execution

Execution

Execution

.class

Compilation

.java

Translators/Conv

JVM

Convertors/Trans

JVM

Translators/Convertors

**.class file** contains bytecode, it is not directly executable code, it is an intermediate code and it is not following any OS Representations including Windows

Sun Microsystems has provided the translators/Convertors as part of the **JVM**

**(Q) What are the differences between .exe file and .class file?**

(A)

1. .exe file is existed in C and C++

.class file is existed in JAVA

1. .exe file are able to provide directly executable code.

.class file are able to provide bytecode, it is not directly executable code, and it is an intermediate code.

1. .exe file is platform dependant file.

.class file is platform independent file

1. .exe file is able to provide less security.

.class file is able to provide more security.

1. **Pointers**

* Pointers are existed in C and C++, but pointers are not existed in Java.
* Pointer is a variable which is able to store address locations of the data structure where data structures may be an array, a variable, a struct or another pointer variable
* In general pointer variables are recognized and initialized at the time of compilation.
* int a =10;
* int \*p;
* p =&a;

**(Q) Why pointer variables are not existed in JAVA?**

(A)

1. Pointer variables require memory allocation at compilation time, that is, pointer variables required static memory allocation, but, JAVA is following dynamic memory allocation.
2. Pointers are supported by static programming language, but, JAVA is dynamic programming language.
3. Pointer variables are very much suitable in platform dependant programming languages, but, JAVA is platform independent Programming Language.
4. Pointer variables are providing less security for the application data but JAVA is very good secure programming language, it has to provide very good security for the application data.
5. Pointer is a bit confusion oriented feature, but, JAVA is a simple Programming Language, it must not provide any confusion to the developers.

**(Q) In C and C++ applications, a variable is referring a block of memory, so that variable is pointer variable. Similarly, in JAVA applications, when we create an object for a particular class there also a variable is referring a block of memory [Object], why don’t we call that variable as pointer variable and how can we say pointers are not existed in java?**

(A)

Heap Memory

class A{

1234-HashCode

--------

--------

-------- A a = new A();

Ref Variable

12345

}

Abs6

Ref value

**(Q) What are the differences between pointer variables and reference variables?**

(A)

1. Pointer variables are the variables which are able to refer a block of memory by storing address locations

Reference variables are the variables, which are able to refer a block of memory [Object] by storing object reference value, where object reference value is a Hexa Decimal Form of Hashcode, where Hashcode is a unique identity provided by Heap Manager to recognize the objects individually.

1. Pointer variables require static memory allocation.

Reference variables require dynamic memory allocation.

1. Pointer variables are existed in C and C++

Reference variables are existed in Java Mainly.

**5. Multiple Inheritance**

Multiple inheritance does not exist in Java

**Object Oriented Features**:

1. Class
2. Object
3. **Encapsulation**
4. **Abstraction**
5. **Inheritance**
6. **Polymorphism**
7. Message passing

**Inheritance:** It is a relation between classes, it will bring variables and methods from one class [**Super Class**] to another class [**Sub Class**].

Advantage: Code Reusability

Types of Inheritances:

1. Single Inheritance: It is a relation between classes, it will bring variables and methods from only one super class to one or more number of sub classes.

class A

int i = 10;

void m1(){

}

extends

class B

i = i + 10;

m1();

**JAVA SUPPORTS SINGLE INHERITANCE**

1. Multiple Inheritance: It is a relation between classes, it will bring variables and methods from more than one super class to one or more number of sub classes.

class A class B

int j = 20;

void m2(){

}

int i = 10;

void m1(){

}

extends

class C

i = i + 10; j = j +20;

m1(); m2();

class A class B

int i = 20;

void m1(){

-Y-impl---

}

int i = 10;

void m1(){

-X-impl---

}

extends

class C

i = i + 10; 20? 30?

m1() X? Y?

**JAVA IS A SIMPLE PROGRAMMING LANGUAGE AND IT SHOULD NOT GIVE ANY CONFUSION TO DEVELOPERS, SO JAVA IS NOT SUPPORTING MULTIPLE INHERITANCE AND ITS ASSOCIATED CONCEPTS**

1. Multi Level Inheritance
2. Hierarchical Inheritance
3. Hybrid Inheritance
4. **Operator Overloading is not possible in JAVA:**
5. Class
6. Object
7. Encapsulation
8. Abstraction
9. Inheritance
10. Polymorphism
11. Message Passing

**Polymorphism :**

Poly -> many

Morphism -> structures / forms

If one object is exists in more than one form, then it is called as polymorphism

Advantage: Flexibility

Types :

1. Static Polymorphism
2. Dynamic Polymorphism

**Static Polymorphism :** if the polymorphism is existed/executed at compilation time, then that polymorphism is called as static polymorphism.

e.g. – Overloading

**Dynamic Polymorphism:** if the polymorphism is existed at runtime, then that polymorphism is called as dynamic polymorphism.

e.g. – Method Overriding.

**Overloading:**

1. Method Overloading
2. Operator Overloading

**1. Method Overloading:** if we declare more than one method with the same name and with the different parameter list.

e.g. –

class A {

void add(int I, int j){

System.out.println(i+j);

}

void add(float f1, float f2){

System.out.println(f1+f2);

}

void add(String str1, String str2){

System.out.println(str1+ str2);

}

}

**2. Operator Overloading:** if we declare any operator with more than one functionality, then it is called as operator overloading

e.g. –

int a = 10;

int b =20;

int c = a + b; // + is for Arithmetic Addition

System.Out.println(c); // OUTPUT: 30

string str1 = “abc”;

string str2 = “def”;

string str3 = str1 + str2; // + is for string concatenation

System.Out.println(str3); // OUTPUT: abcdef

**Q) Why Operator overloading is not possible in JAVA?**

A)

1. Operator overloading is a rarely used feature in application development.
2. If we declare more number of operators with more number of functionalities, then operator overloading feature will increase confusion to the developers while tracing which operator is for which operation.

**NOTE:** In java, as per java internal requirement, JAVA made some of the operators like +,\*, %...... are declared as overloaded operators with fixed functionalities, but java has not provided any environment to perform operator overloading explicitly at developer’s level.

1. **Destructors**

Destructors are **existed** in **C++**, but, Destructors are **not existed** in **java**

Create Objects -> Constructor

Destroy Objects -> Destructor

In JAVA: To destroy objects automatically, JAVA has provided an implicit tool in the form of “**Garbage Collector**”.

In C++: No Garbage Collector king of component, where Developers must take explicit responsibility to destroy objects, it is mandatory to use destructors feature in C++.

1. **Call By Value Vs Call By Reference**

**C and C++** are following **call by value and call by reference** parameter passing mechanism, but, **JAVA** is following only **call by value** parameter passing mechanism.

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

In any programming language, if we pass **address locations** 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 the methods then the parameter passing mechanism is called “**Call By Reference**” because in C and C++, pointer variables are able to store address locations.

In **JAVA,** even if we pass **Object Reference Variables** as **parameters** to the methods, then the parameter passing mechanism is “**Call By Value**” only, because Object Reference Variable is not storing address locations, Object Reference Variable is able to store Object Reference Value, where Object Reference Value is hexa decimal form of Hashcode, where Hashcode is an unique identity provided by Heap Manager in the form of an integer value.

1. **Memory Allocation**

In **C and C++,** **integers** will take **2 bytes** 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** (byte, short, int, float, long, double, Boolean, char) types is **not fixed**, it is variable, and it is **depending on the Operating System** which we used.

In **JAVA**, memory allocation for **primitive data types** is **fixed** irrespective of the Operating System which we used.

byte 🡪 1 byte

short 🡪 2 bytes

int 🡪 4 bytes

long 🡪 8 bytes

float 🡪 4 bytes

double 🡪 8 bytes

char 🡪 2 bytes

boolean 🡪 1 bit

**Q) In C and C++, characters are able to take only one byte of memory, then what is the requirement for JAVA to assign 2 bytes of memory for characters?**

A)

In C and C++ 🡺 All characters 🡺 ASCII 🡺 1 bytes

In JAVA 🡺 All characters 🡺 UNICODE 🡺 2 bytes

**Q) What is UNICODE and what is its requirement in JAVA?**

A) UNICODE 🡺 CHARACTER REPRESENTATION 🡺 Represents all the alphabet from all the natural languages like English, Hindi, Telugu…, with this representation it is able to provide very good internalization [II8N] support in Java Applications.

**NOTE:** Designing application as per Local conventions is called as Internalization.