An Introduction to Java

1.1 A Short History Of Java:
- **Java is a general purpose, object oriented** language developed by Sun Microsystems of USA in 1991.
- Java was originally called Oak (Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.) by James Gosling was designed for the development of Software for consumer electronic devices like TVs, VCRs, toasters and such other electronic machine.
- The Java team that included Patrick NAughton Discovered that the existing languages like C And C++ limitations in terms of both reliability and portability. They Modeled their new language Java on C and C++.
- Java is an island of Indonesia where the first coffee was produced (called java coffee). It is a kind of espresso bean. Java name was chosen by James Gosling while having coffee near his office.

1.2 Features of Java:
The primary objective of **Java programming** language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as java *buzzwords*. 

---

**Features of Java**

- **1. Object Oriented**
- **2. Simple**
- **3. Secured**
- **4. Platform Independent**
- **5. Robust**
- **6. Portable**
- **7. Architecture Neutral**
- **8. Dynamic**
- **9. Interpreted**
- **10. High Performance**
- **11. Multithreaded**
- **12. Distributed**
1. Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

2. Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

3. Platform Independent

Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.
The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API (Application Programming Interface)

Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

4. Secured:

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- No explicit pointer
- Java Programs run inside a virtual machine sandbox
- Classloader: Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- Bytecode Verifier: It checks the code fragments for illegal code that can violate access right to objects.
- Security Manager: It determines what resources a class can access such as reading and writing to the local disk.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

5. Robust

Robust simply means strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.
6. **Architecture-neutral:**

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

7. **Portable:**

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

8. **High-performance**

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

9. **Distributed**

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

10. **Multi-threaded**

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

11. **Dynamic:**

Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.
Java supports dynamic compilation and automatic memory management (garbage collection).

### 1.3 Comparison of Java and C++

<table>
<thead>
<tr>
<th>Comparison Index</th>
<th>C++</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform-independent</td>
<td>C++ is platform-dependent.</td>
<td>C++ is platform-dependent.</td>
</tr>
<tr>
<td>Mainly used for</td>
<td>C++ is mainly used for system programming.</td>
<td>Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.</td>
</tr>
<tr>
<td>Design Goal</td>
<td>C++ was designed for systems and applications programming. It was an extension of C programming language.</td>
<td>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.</td>
</tr>
<tr>
<td>Goto</td>
<td>C++ supports the goto statement.</td>
<td>Java doesn't support the goto statement.</td>
</tr>
<tr>
<td>Multiple inheritance</td>
<td>C++ supports multiple inheritance.</td>
<td>Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.</td>
</tr>
<tr>
<td>Operator Overloading</td>
<td>C++ supports operator overloading.</td>
<td>Java doesn't support operator overloading.</td>
</tr>
<tr>
<td>Pointers</td>
<td>C++ supports pointers. You can write pointer program in C++.</td>
<td>Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.</td>
</tr>
<tr>
<td>Compiler and Interpreter</td>
<td>C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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
### Structure and Union

<table>
<thead>
<tr>
<th>C++</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ supports structures and unions.</td>
<td>Java doesn't support structures and unions.</td>
</tr>
</tbody>
</table>

### Thread Support

<table>
<thead>
<tr>
<th>C++</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.</td>
<td>Java has built-in thread support.</td>
</tr>
</tbody>
</table>

### 1.4 Java Environment:

**What is JVM:**

It is,

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is crea

**JVM Architecture**

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.
1) Classloader

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

1. Bootstrap ClassLoader: This is the first classloader which is the super class of Extension classloader. It loads the rt.jar file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.

2. Extension ClassLoader: This is the child classloader of Bootstrap and parent classloader of System classloader. It loads the jar files located inside $JAVA_HOME/jre/lib/ext directory.
3. System/Application ClassLoader: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

2) Class(Method) Area

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap

It is the runtime data area in which objects are allocated.

4) Stack

Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Register

PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack

It contains all the native methods used in the application.

7) Execution Engine

It contains:

1. A virtual processor
2. Interpreter: Read bytecode stream then execute the instructions.
3. Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler"
refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

8) Java Native Interface

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

1.5 Simple Java Program

To create a simple java program, you need to create a class that contains the main method. Let's understand the requirement first.

The requirement for Java Hello World Example

For executing any java program, you need to

- Install the JDK if you don't have installed it, download the JDK and install it.
- Set path of the jdk/bin directory. http://www.javatpoint.com/how-to-set-path-in-java
- Create the java program
- Compile and run the java program

Program:

class Simple{

    public static void main(String args[]){

        System.out.println("Hello Java");

    }

}

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to
create an object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create an object to invoke the main method. So it saves memory.

- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** is used for command line argument. We will learn it later.
- **System.out.println()** is used to print statement. Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class. We will learn about the internal working of System.out.println statement later.

**What happens at runtime?**

At runtime, following steps are performed:

![Java Tool Diagram]

**Java Tools:**

- 1. **JDB:**
Introduction

Java Debugger (JDB) is a command-line, light weight Java debugging tool. The purpose of this tool is to find bugs from Java programs and to fix them. It tests and monitors the complete Java program and then debug them to accomplish this task.

What is the need of Java Debugger?

If a program gets any bug or invalid data in a program then an error occurs. Sometimes, it is difficult to find and remove these bugs by viewing the code. So, in this situation JDB come into existence.

Features of JDB

Following is a list of some important features of JDB:-

- It is very simple and free to use.
- It is a lightweight tool.
- Being an internal part of JDK, JDB is supported by all type of operating systems.
- As it is a command line tool, it is very fast.

JDB Architecture

The Java Debugger architecture consists of three interfaces:-

- JVM Tool Interface
- Java Debugger Wire Protocol
- Java Debugger Interface (JDI)
JDB Architecture

**JVM Tool Interface**

The Java virtual machine tool interface (JVM TI) provides the services that virtual machine (VM) required for debugging. It examines the state and controls the execution of applications running in the JVM. It supports all the tools that need access to JVM state such as debugging, profiling, thread analysis, monitoring and coverage analysis tools.

**Java Debugger Wire Protocol**

The role of JDWP is to define the format of requests and information between the debugger front end and the process being debugged. Thus, it enables a communication between debugger and the JVM.

It allows the debugger to work in a different process either on the same computer or on a remote computer. Although JDWP is optional.

**Java Debugger Interface**
JDI is the uppermost layer of Java debugger that defines the information and requests at the user code level. It has ability to control the execution of virtual machine. Apart from that, it can also suspend and resume threads.

2: javap:

The javap command disassembles a class file. The javap command displays information about the fields, constructors and methods present in a class file.

Syntax to use javap tool

Let's see how to use javap tool or command.

1. javap fully_class_name

Example to use javap tool

1. javap java.lang.Object

Options of javap tool

The important options of javap tool are as follows.

- help : prints the help message.
- l : prints line number and local variable
- c : disassembles the code
- s : prints internal type signature
- sysinfo : shows system info (path, size, date, MD5 hash)
- constants : shows static final constants
- version : shows version information