JAVA Introduction and History

- Java is a high level, robust, object-oriented and secure programming language.
- Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995.
- “James Gosling” is known as the father of Java. Before Java, its name was Oak. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to java.
- James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.
- Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.
- Firstly, it was called "Green talk" by James Gosling, and the file extension was .gt.
- After that, it was called Oak and was developed as a part of the Green project.

Java History from Oak to Java

- Why 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.
- In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.
- Why had they chose the name Java for Java language? The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell, and fun to say.
- According to James Gosling, "Java was one of the top choices along with Silk". Since Java was so unique, most of the team members preferred Java than other names.
- Java is an island in 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 a cup of coffee nearby his office.
- Notice that Java is just a name, not an acronym.
- Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.
- JDK 1.0 was released on January 23, 1996.

JAVA FEATURES

- **Object Oriented:** In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent:** Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Simple:** Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure:** With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral**: Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Portable**: Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
- **Robust**: Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded:** With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- **Interpreted:** Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- **High Performance:** With the use of Just-In-Time compilers, Java enables high performance.
- **Distributed:** Java is designed for the distributed environment of the internet.
- **Dynamic:** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

What is the structure of object-oriented programming?

The structure, or building blocks, of object-oriented programming include the following:

- **Classes** are user-defined data types that act as the blueprint for individual objects, attributes and methods.
- **Objects** are instances of a class created with specifically defined data. Objects can correspond to real-world objects or an abstract entity. When class is defined initially, the description is the only object that is defined.
- **Methods** are functions that are defined inside a class that describe the behaviors of an object. Each method contained in class definitions starts with a reference to an instance object. Additionally, the subroutines contained in an object are called instance methods. Programmers use methods for reusability or keeping functionality encapsulated inside one object at a time.
- **Attributes** are defined in the class template and represent the state of an object. Objects will have data stored in the attributes field. Class attributes belong to the class itself.

What are the main principles of OOP?

Object-oriented programming is based on the following principles:

- **Encapsulation.** This principle states that all important information is contained inside an object and only select information is exposed. The implementation and state of each object are privately held inside a defined class. Other objects do not have access to this class or the authority to make changes. They are only able to call a list of public functions or methods. This characteristic of data hiding provides greater program security and avoids unintended data corruption.
- **Abstraction.** Objects only reveal internal mechanisms that are relevant for the use of other objects, hiding any unnecessary implementation code. The derived class can have its functionality extended. This concept can help developers more easily make additional changes or additions over time.
- **Inheritance.** Classes can reuse code from other classes. Relationships and subclasses between objects can be assigned, enabling developers to reuse common logic while still maintaining a unique hierarchy. This property of OOP forces a more thorough data analysis, reduces development time and ensures a higher level of accuracy.
- **Polymorphism**. Objects are designed to share behaviors and they can take on more than one form. The program will determine which meaning or usage is necessary for each execution of that object from a parent class, reducing the need to duplicate code. A child class is then created, which extends the functionality of the parent class. Polymorphism allows different types of objects to pass through the same interface.

Procedural Oriented Programming	**Object-Oriented Programming**
In procedural programming, the program is divided into small parts called ***functions***.	In object-oriented programming, the program is divided into small parts called ***objects***.
Procedural programming follows a ***top-down approach***.	Object-oriented programming follows a ***bottom-up approach***.
There is no access specifier in procedural programming.	Object-oriented programming has access specifiers like private, public, protected, etc.
Adding new data and functions is not easy.	Adding new data and function is easy.
Procedural programming does not have any proper way of hiding data so it is ***less secure***.	Object-oriented programming provides data hiding so it is ***more secure***.
In procedural programming, overloading is not possible.	Overloading is possible in object-oriented programming.
In procedural programming, there is no concept of data hiding and inheritance.	In object-oriented programming, the concept of data hiding and inheritance is used.
In procedural programming, the function is more important than the data.	In object-oriented programming, data is more important than function.
Procedural programming is based on the ***unreal world***.	Object-oriented programming is based on the ***real world***.
Procedural programming is used for designing medium-sized programs.	Object-oriented programming is used for designing large and complex programs.
Procedural programming uses the concept of procedure abstraction.	Object-oriented programming uses the concept of data abstraction.
Code reusability absent in procedural programming,	Code reusability present in object-oriented programming.
Examples: C, FORTRAN, Pascal, Basic, etc.	**Examples:** C++, Java, Python, C#, etc

Basis	C	C++	Java
Origin	The C language is based on BCPL.	The C++ language is based on the C language.	The Java programming language is based on both C and C++.
Programming Pattern	It is a procedural language.	It is an object-oriented programming language.	It is a pure object-oriented programming language.
Approach	It uses the top-down approach.	It uses the bottom-up approach.	It also uses the bottom-up approach.
Dynamic or Static	It is a static programming language.	It is also a static programming language.	It is a dynamic programming language.
Code Execution	The code is executed directly.	The code is executed directly.	The code is executed by the JVM.
Platform Dependency	It is platform dependent.	It is platform dependent.	It is platform-independent because of byte code.
Translator	It uses a compiler only to translate the code into machine language.	It also uses a compiler only to translate the code into machine language.	Java uses both compiler and interpreter and it is also known as an interpreted language.
File Generation	It generates the .exe, and .bak, files.	It generates .exe file.	It generates .class file.
Number of Keyword	There are **32** keywords in the C language.	There are **60** keywords in the C++ language.	There are **52** keywords in the Java language.
Source File Extension	The source file has a .c extension.	The source file has a .cpp extension.	The source file has a .java extension.
Pointer Concept	It supports pointer.	It also supports pointer.	Java does not support the pointer concept because of security.

Pre-processor Directives	It uses pre-processor directives such as #include, #define, etc.	It uses pre-processor directives such as #include, #define, #header, etc.	It does not use directives but uses packages.
Constructor/ Destructor	It does not support constructor and destructor.	It supports both constructor and destructor.	It supports constructors only.
Exception Handling	It does not support exception handling.	It supports exception handling.	It also supports exception handling.
Memory Management	It uses the calloc(), malloc(), free(), and realloc() methods to manage the memory.	It uses new and delete operator to manage the memory.	It uses a garbage collector to manage the memory.
Overloading	It does not support the overloading concept.	Method and operator overloading can be achieved.	Only method overloading can be achieved.
Used for	It is widely used to develop drivers and operating systems.	It is widely used for system programming.	It is used to develop web applications, mobile applications, and windows applications.

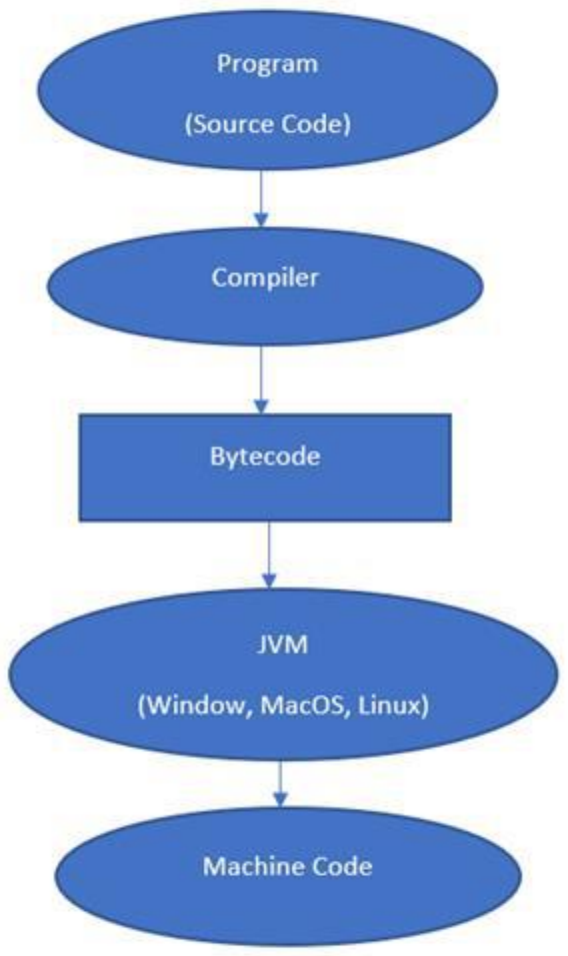
Byte Code

- Byte Code can be defined as an intermediate code generated by the compiler after the compilation of source code (JAVA Program).

This intermediate code makes Java a platform-independent language.

How is Byte Code generated?

- Compiler converts the source code or the Java program into the Byte Code (or machine code), and secondly, the Interpreter executes the byte code on the system.
- The Interpreter can also be called JVM (Java Virtual Machine). The byte code is the common piece between the compiler (which creates it) and the Interpreter (which runs it).



JDK: Java Development Kit

- JDK is an acronym for Java Development Kit.
- The Java Development Kit (JDK) is a software development environment which is used to develop java applications and applets.
- It physically exists and contains JRE + development tools.

JDK is an implementation of any one of the below given Java Platforms released by Oracle corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) etc. to complete the development of a Java Application.

JVM (Java Virtual Machine)

- JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist.
- It is a specification that provides a runtime environment in which Java byte code can be executed.
- It can also run those programs which are written in other languages and compiled to Java byte code.
- JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other.
- However, Java is platform independent. There are three notions of the JVM: specification, implementation, and instance.

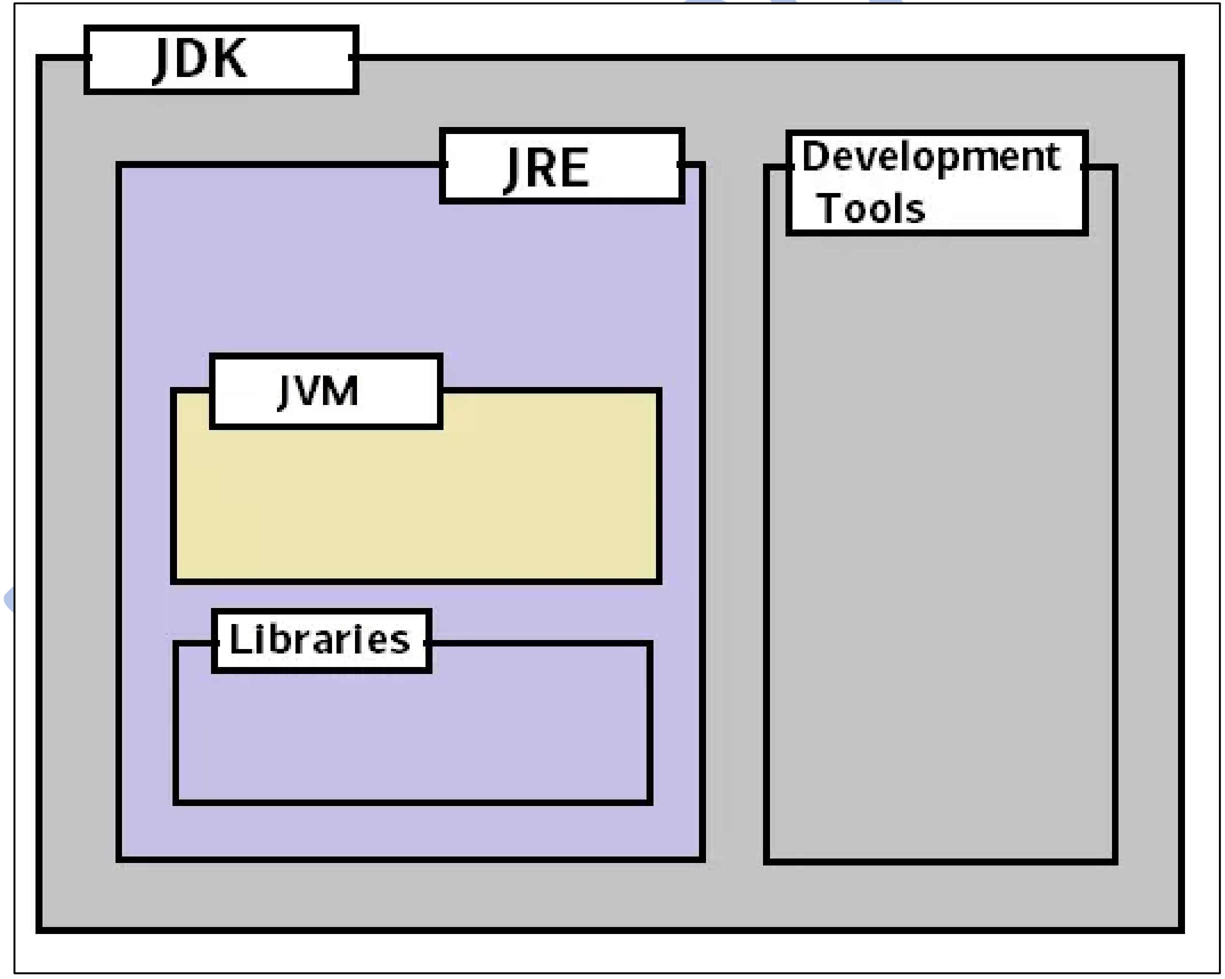
The JVM performs the following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JRE (Java Runtime Environment)

- JRE is an acronym for Java Runtime Environment.
- It is also written as Java RTE.
- The Java Runtime Environment is a set of software tools which are used for developing Java applications.
- It is used to provide the runtime environment. It is the implementation of JVM. It physically exists.
- It contains a set of libraries + other files that JVM uses at runtime.

The implementation of JVM is also actively released by other companies besides Sun Micro Systems.


JDK
JRE
Development Tools
JVM
Libraries