Lesson 1.1

Introducing Java

By the end of this lesson you will learn:

- History of Programming Languages
- The Java Buzzwords
- Importance of OOP
- JDK, JVM & JRE
- JVM Architecture
- A Basic Hello World Program

A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks.

History of Programming Languages

Ever since the invention of Charles Babbage's difference engine in 1822, computers have required a means of instructing them to perform a specific task. This means is known as a programming language. Computer languages were first composed of a series of steps to wire a particular program; these morphed into a series of steps keyed into the computer and then executed; later these languages acquired advanced features such as logical branching and object orientation.

Epoch	Description	Languages
1950's	Creation of high-level languages	Assembly, Fortran
1960's	Expansion of specialized languages	Simula, Lisp, Cobol
1970's	Duel between structured programming with Pascal and efficiency of C language	C, Pascal
1980's	Experimenting other ways including objects	Smalltalk, C++
1990's	Generalization of object-oriented programming with the performance of microcomputers	Java, Perl, Python, PHP
2000's	Internet Programming	C#, Scala
2010's	Concurrency and asynchronicity	JavaScript, Go

Here is the brief description of some programming languages.

1. **Assembly Language (1949):** First widely used in the Electronic Delay Storage Automatic Calculator, assembly language is a type of low-level computer

programming language that simplifies the language of machine code, the specific instructions needed to tell the computer what to do.

2. **Fortran (1957):** In 1957, the first of the major languages appeared in the form of FORTRAN. Its name stands for FORmula TRANslating system. The language was designed at IBM for scientific computing. The components were very simple, and provided the programmer with low-level access to the computers innards. Today, this language would be considered restrictive as it only included IF, DO, and GOTO statements, but at the time, these commands were a big step forward. The basic types of data in use today got their start in FORTRAN, these included logical variables (TRUE or FALSE), and integer, real, and double-precision numbers.

3. **Simula (1962-67):** Ole-Johan Dahl, Kristan Nygaard was the inventor of the language. The Simula project started in 1962. The goal was to build a tool to describe discrete event system, or network, and a language to program simulating real world. Was firstly designed as an Algol extension. In 1964, Simula 1 has been implemented on Univac 1107. It was used to control administrations, airports, planning, transport, or social systems. This was a specialized tool. In 1966, it has been decided to make it a universal language. Several projects has been launched with the help of several makers (IBM, Univac, Digital) and this leaded to Simula 67. This universal language has introduced CLASSES, INHERITANCE and OBJECTS that are instances of classes. Classes allow to link functions (methods) to objects.

4. **LISP (1959):** Created by John McCarthy of MIT, LISP is still in use. It stands for List Processing Language. It was originally created for artificial intelligence research but today can be used in situations where Ruby or Python are used.

5. **COBOL (1959):** It stand for Common Business Oriented Language. Cobol is a classical procedural language aimed at enterprise management, in wich a program is divided in 4 divisions: identification, environment, data, procedure, and they may be divided in sections. It was founded on data and a program must describe precisely the hardware and input/ouput data format. It introduced the RECORD data structure. Programs are documented by the syntax and are all but lightweight.

6. **Pascal (1970):** Developed by Niklaus Wirth, Pascal was named in honor of the French mathematician, physicist, and philosopher Blaise Pascal. It is easy to learn and was originally created as a tool for teaching computer programming. Pascal was the main language used for software development in Apple's early years.

7. **C (1972):** Developed by Dennis Ritchie at Bell Labs. It was firstly destinated to program the UNIX operating system, but has become quickly universal thanks to its portability and speed. Allows incremental compiling. In 1965, ATT programmers were using Bcpl to work on implementing Unix. Displeased with this language, they made it evolve to a new version named B, then to a new language named C.

   This was the evolving of the hardware that instigate to create C. Bcpl and B was using integer for pointers, but this was not working on the new computers.

   Bcpl has no type (as Php or other modern scripting languages!). The declarations int i, char b was created in C. Other types will appear later.

The += operator comes from Algol 68 (but was written =+). In Bcpl, a block of statements was enclosed inside the (* and *) symbols as the comment in /* and */ and sub-expressions inside (and). I suppose this symbolism was intended to denote that anything is an expression in the language, and also to reduce the parsing time. The C language simplified the writting with the { and } symbols, that does the original idea doesn't remains. Union and cast come from Algol 68. ++ was already in the B language. The keyword "include" comes from PL/I. The preprocessor was implemented in 1973, and C has been used from this date to write Unix, but Ritchie worked on it since 1969. The language has evolved until 1980.

8. **Smalltalk (1972):** Developed by Alan Kay, Adele Goldberg, and Dan Ingalls at Xerox Palo Alto Research Center. Smalltalk allowed computer programmers to modify code on the fly and also introduced other aspects now present in common computer programming languages including Python, Java, and Ruby.

9. **C++ (1983):** C++ is an extension of the C language and was developed by Bjarne Stroustrup. It is one of the most widely used languages in the world. C++ is used in game engines and high-performance software like Adobe Photoshop. Most packaged software is still written in C++.

10. **Perl (1987):** It stands for Practical Extracting and Report Language. Created by Larry Wall. Destinated to replace the command line language of Unix, Sh, Sed and Awk, it kept the same ugly syntax. Used mainly for system administration, CGI scripts. Includes lists and associative arrays. The FOREACH control structure allows to scan lists.

11. **Python (1991):** Designed by Guido Van Rossum. Python is easier to read and requires fewer lines of code than many other computer programming languages. It was named after the British comedy group Monty Python. Popular sites like Instagram use frameworks that are written in Python.

12. **PHP (1995):** Created by Rasmus Lerdorf. PHP is used mostly for Web development and is usually run on Web servers. It originally stood for Personal Home Page as it was used by Lerdorf to manage his own online information. But now PHP stands for PHP Hypertext Preprocessor. PHP is now widely used to build websites and blogs. WordPress is a popular website creation tool which is written using PHP.

13. **Java (1995):** Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991. Conceived at the beginning, in 1991, as an interactive language named Oak, was unsuccessful. But in 1994 has been rewritten for Internet and renamed Java. In 1995 navigators can run applets. In January 1996, Javasoft distributes JDK 1.0, the Java Development Kit.

    Java is a object-oriented language, near C++. It compiles in bytecode, interpreted on any computer. It is simpler than C++: one class by file, automatic memory management, no pointers. No multiple inheritance nor operator overloading, but integrated multi-tasking. Unlike C and C++, has only dynamic arrays.

14. **C# (2000):** Developed by Microsoft with the goal of combining the computing ability of C++ with the simplicity of Visual Basic. C# is based on C++ and is like Java in

many aspects. It is used in almost all Microsoft products and is primarily used for developing desktop applications.

15. **Scala (2003):** Created by Martin Odersky. Scala is a computer programming language that combines functional programming, which is mathematical, with object-oriented programming, which is organized around data that controls access to code. Its compatibility with Java makes it helpful in Android development.

16. **JavaScript (1995):** Scripting language to embedd procedural code into web pages. May be used to other applications, XML based languages for example. Share the syntax of C or Java, but with untyped variables. The element of the web page (window, table, etc...) are accessed through the Document Object Model.

17. **Go (2009):** Go was developed by Google to address problems that can occur in large software systems. Since computer and technology use is much different today than it was when languages such as C++, Java, and Python were introduced and put to use, problems arose when huge computer systems became common. Go was intended to improve the working environment for programmers so they could write, read, and maintain large software systems more efficiently.

18. **Kotlin (2011):** In July 2011, JetBrains unveiled Project Kotlin, a new language for the JVM, which had been under development for a year. In February 2012, JetBrains open sourced the project under the Apache 2 license. The name comes from Kotlin Island which is near the St. Petersburg. Andrey Breslav mentioned that the team decided to name it after an island just like Java was named after the Indonesian island of Java.

19. **Swift (2014):** Developed by Apple as a replacement for C, C++, and Objective-C, Swift is supposed to be easier to use and allows less room for mistakes. It is versatile and can be used for desktop and mobile apps and cloud services.

The Creation of JAVA

- Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991.
- It took 18 months to develop the first working version.
- This language was initially called Oak but was renamed Java in 1995.
- The primary motivation was the need for a platform-independent language (that is, architecture-neutral), not the internet. As, trouble with C and C++ (and most other languages) is that they are designed to be compiled for a specific target. Although it is possible to compile a C++ program for just about any type of CPU, to do so requires a full C++ compiler targeted for that CPU.
- Java is just a name, not an acronym.
- In 1995, Time magazine called Java one of the Ten Best Products of 1995.

Version History of JAVA:

Many java versions have been released till now. The current stable release of Java is Java SE 11. The versions are:JDK Alpha and Beta (1995)

- JDK 1.0 (23rd Jan 1996)
- JDK 1.1 (19th Feb 1997)
- J2SE 1.2 (8th Dec 1998)
- J2SE 1.3 (8th May 2000)
- J2SE 1.4 (6th Feb 2002)
- J2SE 5.0 (30th Sep 2004)
- Java SE 6 (11th Dec 2006)
- Java SE 7 (28th July 2011)
- Java SE 8 (18th March 2014)
- Java SE 9 (21st Sep 2017)
- Java SE 10 (20th March 2018)
- Java SE 11 (September 2018)
- Java SE 12 (19th March 2019)
- Java SE 13 (17th September 2019)
- Java SE 14 (17th March 2020)

The Java Buzzwords

No discussion of Java's history is complete without a look at the Java buzzwords.

- Simple
- Secure
- Portable
- Object-oriented
- Robust
- Multithreaded
- Architecture-neutral
- Interpreted
- High performance
- Distributed
- Dynamic

Simple: Java was designed to be easy for the professional programmer to learn and use effectively. If you have some programming experience, you will not find Java hard to master. If you already understand the basic concepts of object-oriented programming, learning Java will be even easier.

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

Java is secured because:

- No explicit pointer
- Java Program runs inside a virtual machine sandbox
- 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.
- Byte code verifier checks the code fragments for illegal code that can violate access right to objects.
- Security manager determines what resources a class can access such as reading and writing to the local disk.

Portable: Java programs are portable because of its ability to run the program on any platform and no dependency on the underlying hardware / operating system.

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 have been briefly described later.

Robust: To better understand how Java is robust, consider two of the main reasons for program failure: memory management mistakes and mishandled exceptional conditions (that is, run-time errors). Memory management can be a difficult, tedious task in traditional programming environments. For example, in C/C++, the programmer will often manually allocate and free all dynamic memory. This sometimes leads to problems, because programmers will either forget to free memory that has been previously allocated or, worse, try to free some memory that another part of their code is still using. Java virtually eliminates these problems by managing memory allocation and deallocation for you. (In fact, deallocation is completely automatic, because Java provides garbage collection for unused objects.) Exceptional conditions in traditional environments often arise in situations such as division by zero or “file not found,” and they must be managed with clumsy and hard-toread constructs. Java helps in this area by providing object-oriented exception handling. In a well-written Java program, all run-time errors can—and should—be managed by your program.

Multithreaded: Java supports multithreaded programming, which allows you to write programs that do many things simultaneously. The Java run-time system comes with an elegant yet sophisticated solution for multiprocess synchronization that enables you to construct smoothly running interactive systems.

Architecture-neutral: The major challenge when Java was developing is to have programming language with which a program can be developed and executed anytime in future. With changing environments of hardware, processor, Operating system there was need to have program still adopt to this architecture changes. Java code does not depend on the underlying architecture and only depends on it JVM thus accomplish the architecture neutral programming language.

Interpreted and High Performance: Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. the Java bytecode was

carefully designed so that it would be easy to translate directly into native machine code for very high performance by using a just-in-time compiler. Java run-time systems that provide this feature lose none of the benefits of the platform-independent code.

Distributed: Java is distributed because it facilitates users to create distributed applications in Java. RMI (Remote Method Invocation) and EJB (Enterprise Java Beans) 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.

Dynamic: Java programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe and expedient manner. This is crucial to the robustness of the Java environment, in which small fragments of bytecode may be dynamically updated on a running system.

Importance of OOP

Here are few aspects why OOP is important.

- Modularity for easier troubleshooting: When working with object-oriented programming languages, if something goes wrong, you know exactly where to look. For example, the car object broke down. The problem must be in the Car class. You do not have to muck through anything else. That is the beauty of encapsulation. Objects are self-contained, and each bit of functionality does its own thing while leaving the other bits alone. Also, this modality allows an IT team to work on multiple objects simultaneously while minimizing the chance that one person might duplicate someone else's functionality.

- Reuse of code through inheritance: Suppose that in addition to your Car object, one colleague needs a RaceCar object, and another needs a Limousine object. Everyone builds their objects separately but discover commonalities between them. In fact, each object is just a different kind of Car. This is where the inheritance technique saves time: Create one generic class (Car), and then define the subclasses (RaceCar and Limousine) that are to inherit the generic class's traits. Of course, Limousine and RaceCar still have their unique attributes and functions. If the RaceCar object needs a method to "fireAfterBurners" and the Limousine object requires a Chauffeur, each class could implement separate functions just for itself. However, because both classes inherit key aspects from the Car class, for example the "drive" or "fillUpGas" methods, your inheriting classes can simply reuse existing code instead of writing these functions all over again. What if you want to make a change to all Car objects, regardless of type? This is another advantage of the OO approach. Simply make a change to your Car class, and all car objects will simply inherit the new code.

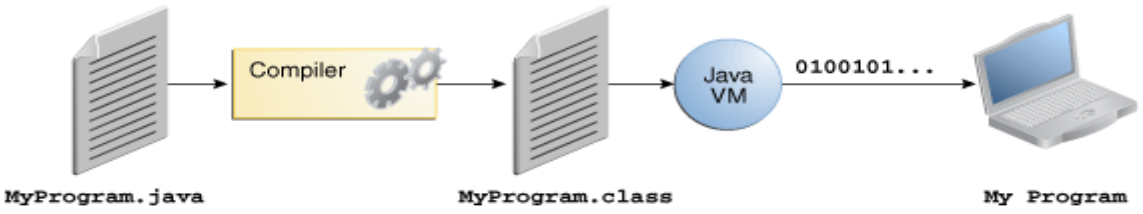
- Flexibility through polymorphism: Riffing on this example, you now need just a few drivers, or functions, like "driveCar," driveRaceCar" and "DriveLimousine." RaceCarDrivers share some traits with LimousineDrivers, but other things, like RaceHelmets and BeverageSponsorships, are unique. This is where object-oriented programming's sweet polymorphism comes into play. Because a single function can

shape-shift to adapt to whichever class it is in, you could create one function in the parent Car class called “drive” - not “driveCar” or “driveRaceCar,” but just “drive.” This one function would work with the RaceCarDriver, LimousineDriver, etc. In fact, you could even have “raceCar.drive(myRaceCarDriver)” or “limo.drive(myChauffeur).”

❖ Effective problem-solving A language like C has an amazing legacy in programming history but writing software in a top-down language is like playing Jenga while wearing mittens. The more complex it gets, the greater the chance it will collapse. Meanwhile, writing a functional-style program in a language like Haskell or ML can be a chore. Object-oriented programming is often the most natural and pragmatic approach. Once you get the hang of it. OOP languages allows you to break down your software into bite-sized problems that you then can solve one object at a time.

JDK, JVM and JRE

- **JVM:** JVM is the heart of Java programming language. JVM (Java Virtual Machine) is called a virtual machine because it does not physically exist. It is a specification that provides a runtime environment in which Java byte code can be executed. When we run the Java program, Java compiler first compiles the Java file to **.class** file from which **byte code** is generated. Then, the JVM translates byte code into native machine code. Because of JVM, java is platform independent. Because when we write Java code, it's ultimately written for JVM but not for the physical machine (computer) we are using.



- **JRE:** JRE (Java Runtime Environment) is a software package that provides Java class libraries, along with Java Virtual Machine (JVM), and other components to run (not develop) applications written in Java programming. JRE is the superset of JVM. It contains a set of libraries + other files that JVM uses at runtime. JRE is also written as Java RTE.

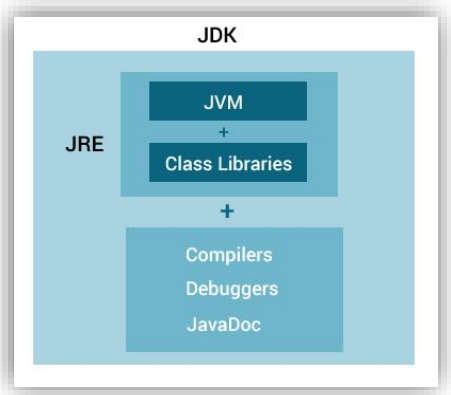
If we need to run Java programs, but not develop them, JRE is what we need.

- **JDK:** JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development kit which is used to develop and execute (run) Java

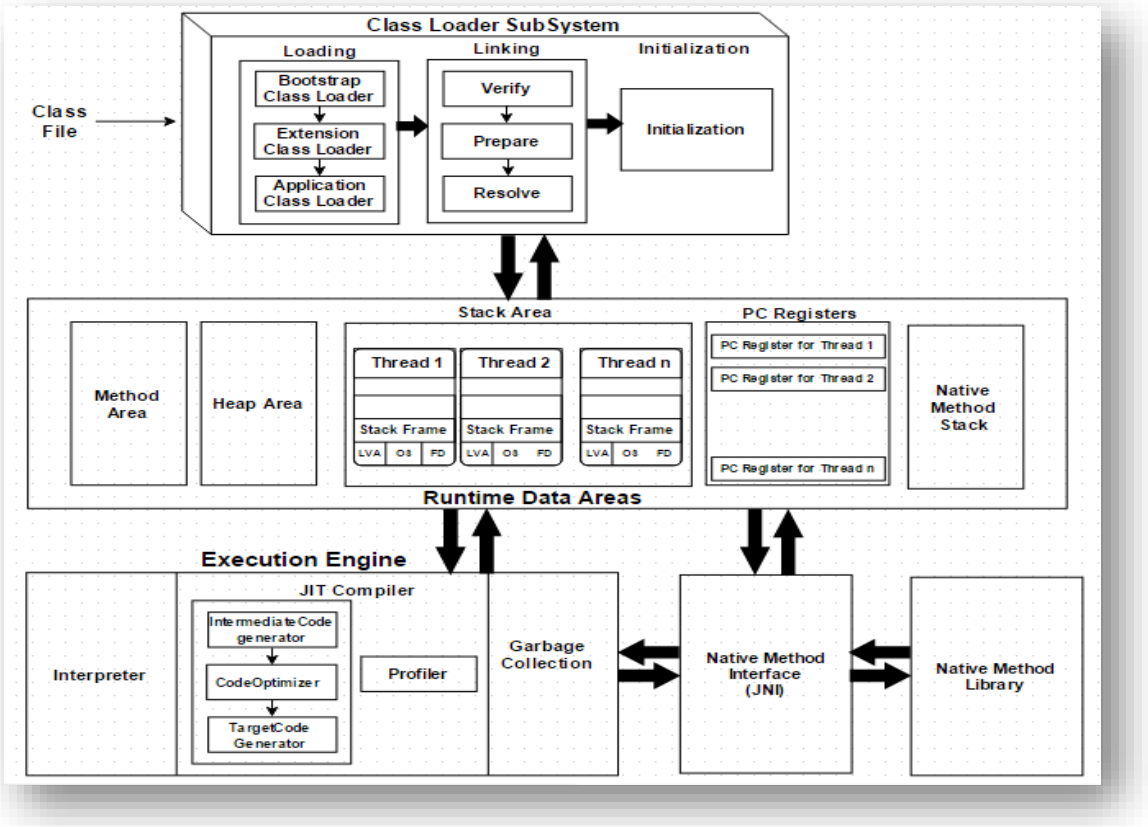
applications. It contains JRE + development tools. When we download JDK, JRE is also downloaded. We don't need to download it separately. JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- 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 Architecture



As shown in the above architecture diagram, the JVM is divided into three main subsystems:

- Class Loader Subsystem
- Runtime Data Area
- Execution Engine

Class Loader Subsystem: Java's dynamic class loading functionality is handled by the class loader subsystem. It loads, links and initializes the class file when it refers to a class for the first time at runtime but not at compile time.

- **Loading:** Classes will be loaded by this component. There are three class loaders which will help in achieving it.
    - **BootStrap ClassLoader:** Responsible for loading classes from the bootstrap class path, nothing but **rt.jar.** Highest priority will be given to this loader.
    - **Extension ClassLoader:** Responsible for loading classes which are inside the ext folder **(jre\lib).**
    - **Application ClassLoader:** Responsible for loading application level class path, path mentioned Environment Variable etc.

  The above Class Loaders will follow Delegation Hierarchy Algorithm while loading the class files.

- **Linking:** Here there are three steps. They are:
    - **Verify:** Byte code verifier will verify whether the generated byte code is proper or not if verification fails we will get the verification error.
    - **Prepare:** For all static variables memory will be allocated and assigned with default values.
    - **Resolve:** All symbolic memory references are replaced with the original references from Method Area.

- **Initialization:** This is the final phase of Class Loading, here all static variables will be assigned with the original values, and the static block will be executed.

Runtime Data Area: The Runtime Data Area is divided into 5 major components:

- **Method Area:** All the class level data will be stored here, including static variables. There is only one method area per JVM, and it is a shared resource.

- **Heap Area:** All the Objects and their corresponding instance variables and arrays will be stored here. There is also one Heap Area per JVM. Since the Method and Heap areas share memory for multiple threads, the data stored is not thread-safe.

- **Stack Area:** For every thread, a separate runtime stack will be created. For every method call, one entry will be made in the stack memory which is called as Stack Frame. All local variables will be created in the stack memory. The stack area is thread-safe since it is not a shared resource. The Stack Frame is divided into three sub entities:
    - **Local Variable Array:** Related to the method how many local variables are involved and the corresponding values will be stored here.
    - **Operand Stack:** If any intermediate operation is required to perform, operand stack acts as runtime workspace to perform the operation.

    - **Frame Data:** All symbols corresponding to the method is stored here. In the case of any **exception**, the catch block information will be maintained in the frame data.

- **PC Registers:** Each thread will have separate PC Registers, to hold the address of current executing instruction once the instruction is executed the PC register will be updated with the next instruction.

- **Native Method Stacks:** Native Method Stack holds native method information. For every thread, a separate native method stack will be created.

Execution Engine: The byte code which is assigned to the **Runtime Data Area** will be executed by the Execution Engine. The Execution Engine reads the byte code and executes it piece by piece.

- **Interpreter:** The interpreter interprets the byte code faster, but executes slowly. The disadvantage of the interpreter is that when one method is called multiple times, every time a new interpretation is required.

- **JIT Compiler:** The JIT Compiler neutralizes the disadvantage of the interpreter. The Execution Engine will be using the help of the interpreter in converting byte code, but when it finds repeated code it uses the JIT compiler, which compiles the entire byte code and changes it to native code. This native code will be used directly for repeated method calls, which improve the performance of the system.
    - **Intermediate Code Generator:** Produces intermediate code
    - **Code Optimizer:** Responsible for optimizing the intermediate code generated above.
    - **Target Code Generator:** Responsible for Generating Machine Code or Native Code.
    - **Profiler:** A special component, responsible for finding hotspots, i.e. whether the method is called multiple times or not.

- **Garbage Collector:** Collects and removes unreferenced objects. Garbage Collection can be triggered by calling System.gc(), but the execution is not guaranteed. Garbage collection of the JVM collects the objects that are created.

JAVA Native Interface (JNI): JNI will be interacting with the Native Method Libraries and provides the Native Libraries required for the Execution Engine.

Native Method Libraries: This is a collection of the Native Libraries which is required for the Execution Engine.

A Basic Hello World Program

A "Hello, World!" is a simple program that outputs Hello, World! on the screen. Since it's a very simple program, it's often used to introduce a new programming language to a newbie.

Let us explore how Java "Hello, World!" program works.

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

Save the file name as HelloWorld.java. It's because the name of the class and filename should match in Java.

`public class HelloWorld { ... }`

In Java, every application begins with a class definition. In the program, HelloWorld is the name of the class, and the class definition is:

```
class HelloWorld {

... .. ...

}
```

For now, just remember that every Java application has a class definition, and the name of the class should match the filename in Java.

`public static void main(String[] args) { ... }`

This is the main method. Every application in Java must contain the main method. The Java compiler starts executing the code from the main method.

`System.out.println("Hello, World!");`

The following code prints the string inside quotation marks Hello, World! to standard output (your screen). Notice, this statement is inside the main method, which is inside the class definition.