Java Architecture combines the process of compilation and interpretation. It explains the various processes involved whilst formulating a [Java program](https://www.edureka.co/blog/java-programs/). Before I begin with the topic let me introduce you with the agenda for this article.

Below mentioned pointers will be our topics of discussion:

* [What is Java Architecture?](https://www.edureka.co/blog/java-architecture/" \l "whatisjavaarchitecture)
* [Components of Java](https://www.edureka.co/blog/java-architecture/#componentsofjava)
* [How is Java platform independent?](https://www.edureka.co/blog/java-architecture/#platformindependent)
* [JIT in Java](https://www.edureka.co/blog/java-architecture/#jitinjava)

Let us begin by understanding what exactly is Java Architecture?

**What is Java Architecture?**

Here, I will explain you the java architecture in simple steps.

* In Java, there is a process of compilation and interpretation.
* The code written in [Java](https://www.edureka.co/blog/java-tutorial/), is converted into byte codes which is done by the Java Compiler.
* The byte codes, then are converted into machine code by the JVM.
* The Machine code is executed directly by the machine.

This diagram illustrates the internal working of a Java code, or precisely, Java Architecture!

Now, let us dig a little deeper in java architecture and talk about various [components of Java](https://www.edureka.co/blog/what-is-java/#ComponentsinJava).

**Components of Java Architecture**

There are three main components of Java language: ***JVM, JRE, and JDK***.

Java Virtual Machine, Java Runtime Environment and Java Development Kit respectively.

Let me elaborate each one of them one by one:

**Java Virtual Machine:**

Ever heard about WORA? (Write once Run Anywhere).  Well, Java applications are called WORA because of their ability to run a code on any platform. This is done only because of JVM. The JVM is a Java platform component that provides an environment for executing Java programs. JVM interprets the bytecode into machine code which is executed in the machine in which the Java program runs.

So, in a nutshell, JVM performs the following functions:

* Loads the code
* Verifies the code
* Executes the code
* Provides runtime environment

Now, let me show you the JVM architecture. Here goes!

****

**Explanation:**

**Class Loader**: Class loader is a subsystem of JVM. It is used to load class files. Whenever we run the java program, class loader loads it first.

**Class method area**: It is one of the Data Area in JVM, in which Class data will be stored. Static Variables, Static Blocks, Static Methods, Instance Methods are stored in this area.

**Heap**: A heap is created when the JVM starts up. It may increase or decrease in size while the application runs.

**Stack**: JVM stack is known as a thread stack. It is a data area in the JVM memory which is created for a single execution thread. The JVM stack of a thread is used by the thread to store various elements i.e.; local variables, partial results, and data for calling method and returns.

**Native stack**: It subsumes all the native methods used in your application.

**Execution Engine:**

* JIT compiler
* Garbage collector

**JIT compiler:** The [Just-In-Time (JIT) compiler](https://www.edureka.co/blog/just-in-time-compiler/) is a part of the runtime environment. It helps in improving the performance of Java applications by compiling bytecodes to machine code at run time. The JIT compiler is enabled by default. When a method is compiled, the JVM calls the compiled code of that method directly. The JIT compiler compiles the bytecode of that method into machine code, compiling it “just in time” to run.

**Garbage collector:** As the name explains that [Garbage Collector](https://www.edureka.co/blog/garbage-collection-in-java/) means to collect the unused material. Well, in JVM this work is done by Garbage collection. It tracks each and every object available in the JVM heap space and removes unwanted ones.  
Garbage collector works in two simple steps known as Mark and Sweep:

* Mark – it is where the garbage collector identifies which piece of memory is in use and which are not
* Sweep – it removes objects identified during the “mark” phase.

**Java Runtime Environment:**

The JRE software builds a runtime environment in which Java programs can be executed. The JRE is the on-disk system that takes your Java code, combines it with the needed libraries, and starts the JVM to execute it. The JRE contains libraries and software needed by your Java programs to run. JRE is a part of JDK (which we will study later) but can be downloaded separately.

**Java Development Kit:**

The Java Development Kit (JDK) is a software development environment used to develop Java applications and applets. It contains JRE and several development tools, an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) accompanied with another tool.

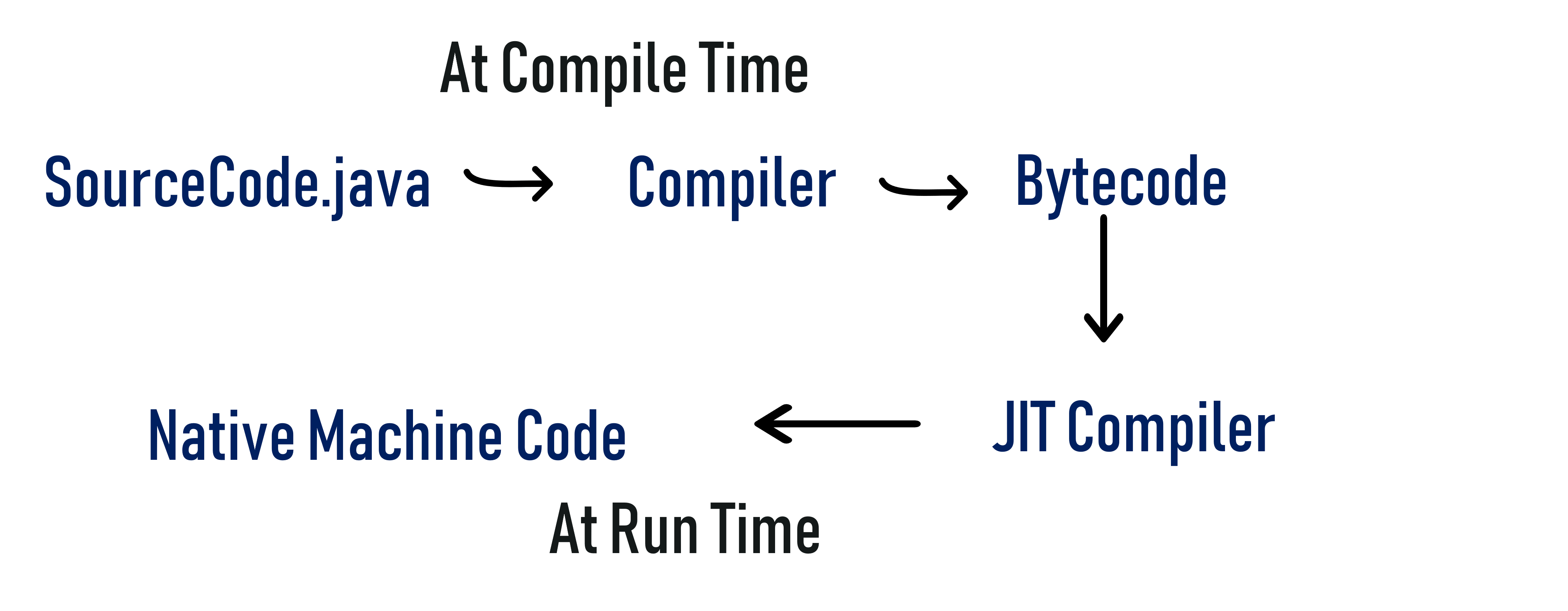
The blue area shown in the diagram is JDK. Now, let me elaborate the development tools to you all.

**java** : it is the launcher for all the java applications.  
**javac** : complier of the java programming languages.  
**javadoc**: it is the API documentation generator.  
**jar**: creates and manage all the JAR files.

Moving ahead with Java architecture, let us understand how Java platform is independent?

## ****How is Java platform independent?****

When is any programming language called as platform-independent? Well, if and only if it can run on all available operating systems with respect to its development and compilation.  
Now, [Java](https://docs.oracle.com/javase/tutorial/)is platform-independent just because of the bytecode. Let me tell you what exactly is a bytecode? In simple terms,  
Bytecode is a code of the JVM which is machine-understandable.  
Bytecode execution in Java proves it is a platform-independent language.  
Here, I will show you the steps involved in the process of java bytecode execution.

Below is the explanation of the steps involved:

***sample.java → javac (sample. class) → JVM(sample.obj) → final output***

First source code is used by java compiler and is converted in .class file. The class file code is in byte code form and that class file is used by JVM to convert into an object file. After that, you can see the final output on your screen.

Moving ahead in Java architecture article, let us understand the concept of [JIT in Java](https://www.edureka.co/blog/just-in-time-compiler/).

## ****JIT in Java****

Just In Time compiler commonly known as JIT, is basically responsible for performance optimization of java based applications at run time. The performance of an application is dependent on a compiler.  
Here is a simple diagram showing you the internal process going on.

