How Java Works

For most programming languages, a program is translated (compiled) into a machine-language program. Then, the machine language program can be executed (or run).

But different computers, or machines, can have different instruction sets. Also, for example, integers may be implemented in words with different sizes on different computers, so that integer overflow might happen differently. So there are many ways in which the same program run on different computers could give different results.

The Java Virtual Machine —JVM

Java is different. When Java was released in 1995, it had the goal that a Java program developed on any computer would run exactly the same on any other computer. It had to, because many Java programs would be written as *applets*, to be run in a browser page on the internet, on any computer.

To achieve what is called *platform independence*, *Java bytecode* was designed: A fake machine language for a fake machine —also called a fake computer, a *Virtual Machine*, or *VM*. To use Java on a new machine, one has to

1. Write a compiler on that machine that translates Java into Java bytecode.
2. Write an *interpreter* for Java bytecode —that is, write a program that reads and executes programs written in bytecode. That interpreter is, then a Java Virtual Machine.

Of course, running an interpreter, a software program, is much slower than executing a machine-language program, and that was a big drawback initially. But people found ways to improve the speed. One important way was the introduction of *Just-In-Time* compilers that could quickly translate some Java bytecode into the real machine language and then execute the bytecode. Moreover, once Java was firmly established, hardware manufacturers began offering hardware support for Java.

However, languages like C and C++, which are always compiled directly into machine language, will always be somewhat faster than Java. Because of this, methods have been developed to allow calling C or C++ procedures from a Java program. This allows one to use Java for most of the coding but to write crucial methods that have to be fast in C or C++.

To read more about Java, its implementation, and its history, visit

https://en.wikipedia.org/wiki/Java_(programming_language)

JREs and JDKs

A *Java Runtime Environment* (JRE) is a software package that contains everything needed to execute a Java program. It includes the interpreter for Java bytecode —the Java Virtual Machine. It also includes the class library —Java bytecode for all the classes that accompany Java, like String, Character, and JFrame. And there is also a *class loader*, which among other things, allocates memory for static variables and calls code to initialize them.

Some small computers —like smart phones— won't have a Java compiler on them (see below), but they might have a JRE so that they can execute or run Java programs that have been translated into Java bytecode.

A Java Development Kit (JDK) is a software package that contains not only a JRE but a Java compiler, so that Java programs can be compiled/translated into Java bytecode. There are several versions of the JDK. The major one can be found here (as of 2018 January): http://www.oracle.com/technetwork/java/javase/downloads/index.html.