How Java Works

A Brief History of How Programming Languages Work

All high-level or third-generation programming languages allow you to write programs in a language similar to but much simpler than natural language. This high level program is called the source code. On the contrary, low-level programming language will make more sense to a computer.

Compilers

Most computer languages use the "compile-link-execute" format, you start with source code and the compiler converts the program into a low-level program.

In most compiled languages, the file containing the resulting low-level code is called an **object file.** Object files are linked together to create an **executable file** (i.e., the operating system can load the executable into RAM to run the program). Another term for an executable is **"(relocatable) machine code".**

Interpreters

There are a smaller number of languages that avoid the **"compile-link-execute"** sequence and instead try to do the conversion **"on-the-fly" or "as needed".**

In other words, an interpreted language takes each high-level statement, determines its low-level version and executes (while linking if need be) the result. This is done for each statement in succession before the next high-level statement is even looked at.

How Java Works

Java is the first substantial language which is **neither truly interpreted or compiled**; instead, a combination of the two forms is used. This method has advantages which were not present in earlier languages.

Platform-Independence

To understand the primary advantage of Java, you'll have to learn about platforms. In most programming languages, a compiler (or interpreter) generates code that can execute on a specific target machine. A platform is determined by the target machine (along with its operating system). For earlier languages, **language designers needed to create a specialized version of the compiler (or interpreter) for every platform.** If you wrote a program that you wanted to

make available on multiple platforms, you, as the programmer, would have to do quite a bit of additional work. You would have to create multiple versions of your source code for each platform.

Java succeeded in eliminating the platform issue for high-level programmers (such as you) because **it has reorganized the compile-link-execute sequence** at an underlying level of the compiler. Details are complicated but, essentially, **the designers of the Java language isolated those programming issues which are dependent on the platform and developed low-level means to abstractly refer to these issues.** Consequently, **the Java compiler doesn't create an object file**, but instead it creates a **bytecode file** which is, essentially, an object file for a virtual machine. In fact, the Java compiler is often called the **JVM compiler (for Java Virtual Machine).**

Consequently, you can write a Java program (on any platform) and use the **JVM compiler (called javac)** to generate a bytecode file (bytecode files use the extension .class). This bytecode file can be used on any platform that has installed Java. However, **bytecode is not an executable file.** To execute a bytecode file, you actually **need to invoke a Java interpreter called java.** Every platform has its own Java interpreter which will automatically address the platform-specific issues that can no longer be put off. When platform-specific operations are required by the bytecode, the Java interpreter links in appropriate code specific to the platform.

To summarize how Java works (to achieve platform independence), think about the **compile-link-execute cycle.** In earlier programming languages, the cycle is more closely defined as **"compile-link then execute".** In Java, the cycle is closer to **"compile then link-execute".**

Other Advantages of Java

Most of the other features of Java had previously existed in various other programming languages (but never all at once). Most explanations of these advantages (e.g., distributed programming, multi-threading, security) are well beyond the scope of this course. However, there are two features that I will briefly address.

Another feature that was introduced with the Java language is the ability to write **special Java programs called applets** that are designed to run on the World Wide Web. You could write a Java applet and put the bytecode on a web page; if anyone with a Java-enabled web browser goes to your web page, that applet bytecode will be downloaded to the browsing computer and executed within the web browser. Of course, this would not be possible without platform independence. This feature had a major effect on the rapid proliferation of Java and many instructors (including myself) taught programming using applet-based programs. However, for a variety of reasons, I have decided to postpone a discussion of applets (and more generally, graphic user interfaces) to the end of the course.

Java is also one of the first languages to be **"library-based"** in that **the designers of the language have included a large number of pre-existing programs.** A programmer can connect their program to these general purpose programs as needed. It frees up the programmer's time since s/he doesn't have to write as much code.

Java vs. C#

Microsoft's recent language, **C#** (a variation of C/C++), was the first major language after Java to have a **compile then link-execute cycle.** C# has also addressed most of the other issues that were Java advantages and C# has introduced other advantages as well. However, **Microsoft does not seem to be interested in true platform independence.** Also, **sections of the C# software libraries require additional purchases** beyond the basic cost of the language.