Chapter 1. Getting Started: Compiling, Running, and Debugging

1.0 Introduction

This chapter covers some entry-level tasks that you need to know how to do before you can go on—it is said you must crawl before you can walk, and walk before you can ride a bicycle. Before you can try out anything in this book, you need to be able to compile and run your Java code, so I start there, showing several ways: the JDK way, the Integrated Development Environment (IDE) way, and the build tools (Ant, Maven, etc.) way. Another issue people run into is setting CLASSPATH correctly, so that's dealt with next. Deprecation warnings follow after that, because you're likely to encounter them in maintaining "old" Java code. The chapter ends with some general information about conditional compilation, unit testing, assertions, and debugging.

If you don't already have Java installed, you'll need to download it. Be aware that there are several different downloads. The JRE (Java Runtime Environment) is a smaller download for end users. The JDK or Java SDK download is the full development environment, which

you'll want if you're going to be developing Java software.

Standard downloads for the current release of Java are available at Oracle's website.

You can sometimes find prerelease builds of the next major Java version on *http://java.net*. The entire (almost) JDK is maintained as an open source project, and the OpenJDK source tree is used (with changes and additions) to build the commercial and supported Oracle JDKs.

If you're already happy with your IDE, you may wish to skip some or all of this material. It's here to ensure that everybody can compile and debug their programs before we move on.

1.1 Compiling and Running Java: JDK

Problem

You need to compile and run your Java program.

Solution

This is one of the few areas where your computer's operating system impinges on Java's portability, so let's get it out of the way first.

JDK

Using the command-line Java Development Kit (JDK) may be the best way to keep up with the very latest improvements in Java. Assuming you have the standard JDK installed in the standard location and/or have set its location in your PATH, you should be able to run the command-line JDK tools. Use the commands *javac* to compile and *java* to run your program (and, on Windows only, *javaw* to run a program without a console window). For example:

```
C:\javasrc>javac HelloWorld.java

C:\javasrc>java HelloWorld
Hello, World

C:\javasrc>
```

If the program refers to other classes for which source is available (in the same directory) and a compiled .class file is not, `javac` will automatically compile it for you. Effective with Java 11, for simple programs that don't need any such co-compilation, you can combine the two operations, simply passing the Java source file to the `java` command:

```
java HelloWorld.java
```

As you can see from the compiler's (lack of) output, this compiler works on the Unix "no news is good news" philosophy: if a program was able to do what you asked

it to, it shouldn't bother nattering at you to say that it did so. Many people use this compiler or one of its clones.

There is an optional setting called CLASSPATH, discussed in Recipe 1.4, that controls where Java looks for classes. CLASSPATH, if set, is used by both `javac` and `java`. In older versions of Java, you had to set your CLASSPATH to include ".", even to run a simple program from the current directory; this is no longer true on current Java implementations.

Sun/Oracle's *javac* compiler is the official reference implementation. There were several alternative open source command-line compilers, including Jikes and Kaffe but they are, for the most part, no longer actively maintained.

There have also been some Java runtime clones, including Apache Harmony, Japhar, the IBM Jikes Runtime (from the same site as Jikes), and even JNODE, a complete, standalone operating system written in Java, but since the Sun/Oracle JVM has been open-sourced (GPL), most of these projects have become unmaintained. Harmony was retired by Apache in November 2011.

MAC OS X

The JDK is pure command line. At the other end of the spectrum in terms of keyboard-versus-visual, we have the Apple Macintosh. Books have been written about how great the Mac user interface is, and I won't step

into that debate. Mac OS X (Release 10.x of Mac OS) is built upon a BSD Unix (and "Mach") base. As such, it has a regular command line (the Terminal application, hidden away under */Applications/Utilities*), as well as both the traditional Unix command-line tools and the graphical Mac tools. Mac OS X users can use the command-line JDK tools as above or any of the modern build tools. Compiled classes can be packaged into "clickable applications" using the Jar Packager discussed in [Link to Come]. Mac fans can use one of the many full IDE tools discussed in Recipe 1.2. Apple provides XCode as their IDE, but out of the box it isn't very Java-friendly.

GRAALVM

A new VM implementation called GraalVM is now available. Graal promises to offer better performance, the ability to mix-and-match programming languages, and the ability to pre-compile your Java code into executable form for a given platform. See The Graal VM web site for more information on GraalVM.

1.2 Compiling, Running, and Testing with an IDE

Problem

It is cumbersome to use several tools for the various development tasks.

Solution

Use an integrated development environment (IDE), which combines editing, testing, compiling, running, debugging, and package management.

Discussion

Many programmers find that using a handful of separate tools—a text editor, a compiler, and a runner program, not to mention a debugger—is too many. An IDE *integrates* all of these into a single toolset with a graphical user interface. Many IDEs are available, ranging all the way up to fully integrated tools with their own compilers and virtual machines. Class browsers and other features of IDEs round out the ease-of-use feature sets of these tools. It has been argued many times whether an IDE really makes you more productive or if you just have more fun doing the same thing. However, today most developers use an IDE because of the productivity gains. Although I started as a command-line junkie, I do find that the following IDE benefits make me more productive:

Code completion

> *Ian's Rule* here is that I never type more than three characters of any name that is known to the IDE; let the computer do the typing!

"Incremental compiling" features

> Note and report compilation errors as you type, instead of waiting until you are finished typing.

Refactoring

> The ability to make far-reaching yet behavior-preserving changes to a code base without having to manually edit dozens of individual files.

Beyond that, I don't plan to debate the IDE versus the command-line process; I use both modes at different times and on different projects. I'm just going to show a few examples of using a couple of the Java-based IDEs.

The three most popular Java IDEs, which run on all mainstream computing platforms and quite a few niche ones, are *Eclipse*, *NetBeans*, and *IntelliJ IDEA*. Eclipse is the most widely used, but the others each have a special place in the hearts and minds of some developers. If you develop for Android, the ADT has traditionally been developed for Eclipse, but it has now transitioned IntelliJ as the basis for "Android Studio," which is the standard IDE for Android, and for Google's other mobile platform, Flutter. All three are plug-in based and offer a wide selection of optional and third-party plugins to enhance the IDE, such as supporting other programming languages, frameworks, file types, and so on. While the following shows creating and running a program with Eclipse, the IntelliJ IDea and Netbeans IDEs all offer similar capabilities.

Perhaps the most popular cross-platform, open source IDE for Java is Eclipse, originally from IBM and now shepherded by the Eclipse Foundation, the home of many software projects including Jakarta, the follow-on

to the Java Enterprise Edition. Eclipse is also used as the basis of other tools such as SpringSource Tool Suite (STS) and IBM's Rational Application Developer (RAD). All IDEs do basically the same thing for you when getting started; see, for example, the Eclipse New Java Class Wizard shown in Figure 1-1. Eclipse also features a number of refactoring capabilities, shown in Figure 1-2.