Chapter 1

Why Programming and Why Java™ Programming?

This chapter provides two central premises for the rest of the book. First, why would a linguist, psycholinguist, literary theorist, and so on want to know anything about programming? Second, why would Java programming be a good choice?

1.1 Why Programming?

Working with language data is nearly impossible these days without a computer. Data are massaged, analyzed, sorted, and distributed on computers. There are various software packages available for language researchers, but to truly take control of this domain, some amount of programming expertise is essential. Consider the following simple examples.

Imagine you are a syntactician interested in the use of present-tense verbs. You have an electronic corpus and want to find all the cases of verbs in the present tense. How do you do it?

You're a literary stylist and want to investigate the distribution of words with iambic stress in Milton's poetry.

Imagine you are a phonologist. You're interested in consonant clusters. You have an electronic dictionary and want to find the largest word-final consonant cluster. Do you go through it by hand?

Finally, you're a psycholinguist and you want to perform an experiment to investigate how people syllabify nonsense words.

All of these are fairly typical research tasks. If you don't know how to program yourself, you have only limited options. One possibility is to do the job by hand. For example, the syntactician could simply print out the corpus and go through it line by line. If the corpus is small enough, this might not be

so onerous, but if the corpus is large, or if one really wants to be sure of one's results, then this method is fraught with peril (and really boring). Another solution is to hire somebody else to do the job, but the same considerations apply. Yet a third possibility is to make use of some existing software package.

This last option is occasionally workable, but can fall short in several ways. First, an existing package is restricted by its design. That is, your needs may not match what the software was designed to do, rendering your task impossible or very difficult. Moreover, the software may not be intuitive, and may require learning some arcane set of commands or some difficult control language. Finally, while software may exist to do what you want, it may be unavailable on the platform you work on (Windows, Mac, or Unix), or it may be too costly.

1.2 Why Java Technology?

The **Java** programming language may provide an answer. First, it is a complete programming language, with all the bells and whistles. It can do all the file manipulation and text searching one might want, while at the same time, it has all the graphical capabilities of a language like **C**.

Moreover, it's *free*. There are free Java implementations for every type of computer. In addition, code written and compiled on one type of machine will run on any other type. In other words, you can write your code at home on your Mac, and run it at work on your Windows machine, or send it to a colleague to run under Unix. This also suggests that Java programs you write should continue to be usable for many years, since the language is so widely used.

Finally, one of the most compelling features of Java programs is the fact that they can be run over the web in your web browser. What this means is that you can write a program, put it on the web, and allow others to run your program on their own machines simply by going to your web page.

The only downside is that since the Java language can do so much, it can be quite complex. There are lots and lots of features that enable Java programs to do pretty much anything you want. For the novice programmer, this can be intimidating.

We won't let this deter us though. My strategy will be to pick and choose. I'll introduce those bits of Java technology that are necessary to do the kinds of things people who work with language typically want to do. The rest – all the bells and whistles that we don't need on our train – we'll leave for later. I'll let you know where they are and how to find out more, but we won't digress to deal with them here.

1.3 Download and Install the Java Development Kit

Before going on to actually writing Java programs, your computer must be properly configured so that the software for developing programs is available. You must make sure that the **Java Development Kit** (JDK) is installed on your computer. If it is, the command java -version, when typed at the system prompt, will print out appropriate version information. (For a Mac, where there is no system prompt, you must do a search for the program javac, using the Find File command in the Finder.)

If you don't find java or javac on your system, the JDK can be downloaded for free over the web. The appropriate URL is http://www.javasoft.com. Versions of the JDK for Windows and Solaris can be downloaded directly from that site. For other computer types, there are links there to the appropriate site.

It's really very easy to install the JDK, but if you find it daunting at the beginning, you might find it easier to work on a computer that already has the JDK installed. Most mainframe computers have it already installed. If you have access to one – through work or school – you might try running the early programs in the book there.

Let's briefly go over how to install the JDK under Windows. The first step is to download the JDK installer over the web. This is a huge file and you should arrange carefully how to do this. If you have a dedicated very fast connection, there's no problem, but if you want to install the JDK at home over the phone, you need to plan carefully. Depending on the speed of your modem, the download could take *hours*.

If you plan to download the file by phone, you might want to curl up with a book while doing so. Alternatively, it is possible, for a fee, to obtain the JDK on CD-ROM. Finally, if you have ethernet access at work, download the file there, and bring it home on a zip disk.

The JDK installer is typically downloaded in compressed form. For Windows, zip compression is the easiest one to deal with. (For example, in Windows 98, the compressed zip file can be uncompressed automatically.)

Once uncompressed, the second step is to run the install program. This will create a directory called something like j2dk1.x, where x is the version number.

The third step is to add the java and javac programs to your path. This is done by editing your autoexec.bat file. Note that this is something you have to be very careful with. Make sure to make a backup copy of your autoexec.bat file *before* editing it, and make sure to consult your documentation if you've never done this before. First, select the MS-DOS prompt from the Programs menu. Second, type cd \ to switch to the root directory. Type

copy autoexec.bat autoexec.bak to make a copy of your autoexec.bat file. Third, type edit autoexec.bat to edit the file. Use the arrow keys to move to the first empty line at the end of the file. Add the following line set path=%PATH%;\j2dk1.x\bin, where x is the appropriate version number in the directory installed. Finally, select quit from the file menu to quit the edit program, making sure to save your changes when prompted. The change will take place when Windows is next started.

Analogous steps are needed to install the JDK under Unix. For Macs, there are different compression options, and the relevant directory is MRJ SDK x, where x is the appropriate version number. There is no Mac path to alter.

1.4 How to Read this Book

Learning to program isn't really hard, but you do need to do it the right way. The key is to start programming right away. As you read this book, you should make sure to try out the programs as we go through them. In fact, it would be ideal to read the book *at the computer*. Also, don't forget to try the exercises! You'll note that answers are not given at the end of the book. This is for two reasons. First, having answers is a big temptation. More importantly, however, most of the exercises involve revising or writing programs. There are often many ways to achieve the same goal and I would rather you find *some* way to answer an exercise question than feel you have to find *my* way of answering one of them.

Start by running the programs exactly as given, either by downloading them from the website or – even better – by typing them in yourself. (Typing them in yourself will make the task familiar and draw your attention to aspects of the code you might miss otherwise.)

When you start to feel more comfortable, try varying the code a bit. The programs up through chapter 3 are perfectly safe and variations can't harm your computer. After that point, certain operations should be handled with care, but I'll warn you about those as we go through.

The key, though, is to have fun!