Preface

If you know a little Java™, great. If you know more Java, even better! This book is ideal for anyone who knows some Java and wants to learn more.

I started programming in C in 1980 while working at the University of Toronto, and C served me quite well through the 1980s and into the 1990s. In 1995, as the nascent language Oak was being renamed Java, I had the good fortune to be told about it by my colleague J. Greg Davidson. I sent an email to the address Greg provided, and got this mail back:

```
From scndprsn.Eng.Sun.COM!jag Wed Mar 29 19:43:54 1995
Date: Wed, 29 Mar 1995 16:47:51 +0800
From: jag@scndprsn.Eng.Sun.COM (James Gosling)
To: ian@scooter.Canada.Sun.COM, ian@darwinsys.com
Subject: Re: WebRunner
Cc: goltz@sunne.East.Sun.COM
Content-Length: 361
Status: RO
X-Lines: 9

> Hi. A friend told me about WebRunner(?), your extensible network
> browser. It and Oak(?) its extention language, sounded neat. Can
> you please tell me if it's available for play yet, and/or if any
> papers on it are available for FTP?

Check out http://java.sun.com
(oak got renamed to java and webrunner got renamed to
 hotjava to keep the lawyers happy)
```

I downloaded HotJava and began to play with it. At first I wasn't sure about this newfangled language, which looked like a mangled C/C++. I wrote test and demo programs, sticking them a few at a time into a directory that I called *javasrc* to keep it separate from my C source (as often the programs would have the same name). And as I learned more about Java, I began to see its advantages for many kinds of work, such as the automatic memory reclaim and the elimination of pointer calculations. The *javasrc* directory kept growing. I wrote a Java course for Learning Tree, and the directory kept growing faster, reaching the point where it needed subdirectories. Even then, it became increasingly difficult to find things, and it soon became evident that some kind of documentation was needed.

In a sense, this book is the result of a high-speed collision between my *javasrc* directory and a documentation framework established for another newcomer language. In O'Reilly's *Perl Cookbook,* Tom Christiansen and Nathan Torkington worked out a very successful design, presenting the material in small, focused articles called "recipes." The original model for such a book is, of course, the familiar kitchen cookbook. There is a long history of using the term "cookbook" to refer to an enumeration of how-to recipes relating to computers. On the software side, Donald Knuth applied the "cookbook" analogy to his book *The Art of Computer Programming* (Addison Wesley), first published in 1968. On the hardware side, Don Lancaster wrote *The TTL Cookbook* (Sams). (Transistor-transistor logic, or TTL, was the small-scale building block of electronic circuits at the time.) Tom and Nathan worked out a successful variation on this, and I recommend their book for anyone who wishes to, as they put it, "learn more Perl." Indeed, the work you are now reading intends to be a book for the person who wishes to "learn more Java."

The code in each recipe is intended to be self-contained; feel free to borrow bits and pieces of any of it for use in your own projects.

Who This Book Is For

I'm going to assume that you know the basics of Java. I won't tell you how to `println` a string and a number at the same time, or how to write a class that extends `Applet` and prints your name in the window. I'll presume you've taken a Java course or studied an introductory book such as O'Reilly's *Learning Java* or *Java in a Nutshell*. However, Chapter 1 covers some techniques that you might not know very well and that are necessary to understand some of the later material. Feel free to skip around! Both the printed version of the book and the (eventual) electronic copy are heavily cross-referenced.

What's in This Book?

Unlike my Perl colleagues Tom and Nathan, I don't have to spend as much time on the oddities and idioms of the language; Java is refreshingly free of strange quirks. But that doesn't mean it's trivial to learn well! If it were, there'd be no need for this book. My main approach, then, is to concentrate on the Java APIs: I'll teach you by example what the APIs are and what they are good for.

Like Perl, Java is a language that grows on you and with you. And, I confess, I use Java most of the time nowadays. Things I'd once done in C are now -- except for device drivers and legacy systems -- done in Java.

But Java is suited to a different range of tasks than Perl. Perl (and other scripting languages such as awk and Python) are particularly suited to the "one-liner" utility task. As Tom and Nathan show, Perl excels at things like printing the 42nd line from a file. While it can certainly do these things, Java, because it is a compiled, object-oriented language, seems more suited to "development in the large" or enterprise applications development. Indeed, much of the API material added in Java 2 was aimed at this type of development. However, I will necessarily illustrate many techniques with shorter examples and even code fragments. Be assured that every line of code you see here has been compiled and run.

Many of the longer examples in this book are tools that I originally wrote to automate some mundane task or another. For example, `MkIndex` (described in Chapter 1) reads the top-level directory of the place where I keep all my Java example source code and builds a browser-friendly *index.html* file for that directory. For another example, the body of the book itself was partly composed in XML, a recent simplification that builds upon a decade of experience in SGML (the parent standard that led to the tag-based syntax of HTML). It is not clear at this point if XML will primarily be useful as a publishing format or as a data manipulation format, or if its prevalence will further blur that distinction, though it seems that the blurring of distinctions is more likely. However, I used XML here to type in and mark up the original text of some of the chapters of this book. The text was then converted to FrameMaker input by the `XmlForm` program. This program also handles -- by use of another program, `GetMark` -- full and partial code insertions from the source directory. `XmlForm` is discussed in Chapter 21.

Let's go over the organization of this book. I start off Chapter 1 by describing some methods of compiling your program on different platforms, running them in different environments (browser, command line, windowed desktop), and debugging. Chapter 2 moves from compiling and running your program to getting it to adapt to the surrounding countryside -- the other programs that live in your computer.

The next few chapters deal with basic APIs. Chapter 3 concentrates on one of the most basic but powerful data types in Java, showing you how to assemble, dissect, compare, and rearrange what you might otherwise think of as ordinary text.

Chapter 4 teaches you how to use the powerful regular expressions technology from Unix in many string-matching and pattern-matching problem domains. This is the first chapter that covers a non-standard API -- there is not yet a regular expression API in standard Java -- so I talk about several regular expression packages.

Chapter 5 deals both with built-in types such as `int` and `double`, as well as the corresponding API classes (`Integer`, `Double`, etc.) and the conversion and testing facilities they offer. There is also brief mention of the "big number" classes. Since Java programmers often need to deal in dates and times, both locally and internationally, Chapter 6 covers this important topic.

The next two chapters cover data processing. As in most languages, *arrays* in Java are linear, indexed collections of similar-kind objects, as discussed in Chapter 7. This chapter goes on to deal with the many "Collections" classes: powerful ways of storing quantities of objects in the `java.util` package. Additional data structuring and programming tips appear in Chapter 8.

The next few chapters deal with aspects of traditional input and output. Chapter 9 details the rules for reading and writing files. (Don't skip this if you think files are boring, as you'll need some of this information in later chapters: you'll read and write on serial or parallel ports in Chapter 11 and on a socket-based network connection in Chapter 15!) Chapter 10 shows you everything else about files -- such as finding their size and last-modified time -- and about reading and modifying directories, creating temporary files, and renaming files on disk. Chapter 11 shows how you can use the `javax.comm` API to read/write on serial and parallel ports without resorting to coding in C.

Chapter 12 leads us into the GUI development side of things. This chapter is a mix of the lower-level details, such as drawing graphics and setting fonts and colors, and very high-level activities, such as controlling a playing video clip or movie. Then, in Chapter 13 I cover the higher-level aspects of a GUI, such as buttons, labels, menus, and the like -- the GUI's predefined components. Once you have a GUI (really, before you actually write it), you'll want to read Chapter 14 so your programs can work as well in Akbar, Afghanistan, Algiers, Amsterdam, or Angleterre as they do in Alberta or Arkansas or Alabama . . .

Since Java was originally promulgated as "the programming language for the Internet," it's only fair that we spend some of our time on networking in Java. Chapter 15, covers the basics of network programming from the client side, focusing on sockets. We'll then move to the server side in Chapter 16. In Chapter 17, you'll learn more client-side techniques. Some specialized server-side techniques for the Web are covered in Chapter 18. Finally, programs on the Net often need to generate electronic mail, so this section ends with Chapter 19.

Chapter 20 covers the Java Database Connectivity package (JDBC), showing how you can connect to local or remote relational databases, store and retrieve data, and find out information about query results or about the database.

Another form of storing and exchanging data is XML. Chapter 21 discusses XML's formats and some operations you can apply using SAX and DOM, two standard Java APIs.

Chapter 22 takes the distributed notion one step further and discusses Remote Methods Invocation, Java's standard remote procedure call mechanism. RMI lets you build clients, servers,

and even "callback" scenarios, using a standard Java mechanism -- the Interface -- to describe the contract between client and server.

Chapter 23 shows how to create packages of classes that work together. This chapter also talks about "deploying" or distributing and installing your software.

Chapter 24 tells you how to write classes that appear to do more than one thing at a time and let you take advantage of powerful multiprocessor hardware.

Chapter 25 lets you in on such big secrets as how to write API cross reference documents mechanically and how web browsers are able to load any old applet -- never having seen that particular class before -- and run it.

Sometimes you already have code written and working in another language that can do part of your work for you, or you want to use Java as part of a larger package. Chapter 26 shows you how to run an external program (compiled or script) and also interact directly with "native code" in C/C++.

There isn't room in an 800-page book for everything I'd like to tell you about Java. The Chapter 27 presents some closing thoughts and a link to my online summary of Java APIs that every Java developer should know about.

No two programmers or writers will agree on the best order for presenting all the Java topics. To help you find your way around, there are extensive cross-references, mostly by recipe number.

Platform Notes

In its short history, Java has gone through four major versions. The first official release is known as Java JDK 1.0, and its last bug-fixed version is 1.0.2. The second major release is Java JDK 1.1, and the latest bug-fixed version is 1.1.9, though it may be up from that by the time you read this book. The third major release, in December 1998, was to be known as Java JDK 1.2, but the Sun marketing gremlins abruptly renamed JDK 1.2 at the time of its release to Java 2, and the implementation is known as Java SDK 1.2. The current version as of this writing is Java 2 SDK 1.3 (JDK 1.3), which was released in 2000. Around the same time, two other packages, one low-end and one high-end, were announced. At the low end, Java Micro Edition (JME) is designed for tiny devices, such as Palm computers, telephones, and the like. At the high end, the Java 2 Enterprise Edition (J2EE) extends Java 2 by adding additional features for enterprise or large-scale distributed commercial applications. One of the key features of the Enterprise Edition is Enterprise JavaBeans™ (EJB). EJB has little in common with client-side JavaBeans except the name. Many Java pundits (including myself) believe that EJB will become a significant player in the development of large commercial applications, perhaps the most significant development of this era.

As we go to press, Java 2 Version 1.4 is about to appear. It entered beta (which Sun calls "early access") around the time of the book's completion, so I can only mention it briefly. You should cast your sights on http://java.sun.com to see what's new in 1.4 and how it affects the programs in the book.

This book is aimed at the Java 2 platform. By the time of publication, I expect that all Java implementations will be fairly close to conforming to the Java 2 specification. I have used four platforms to test this code for portability. The official "reference platform" is Sun's Java 2 Solaris Reference Implementation, which I used on a Sun SPARCStation running Solaris. To give a second Unix flavor, I've tested with Kaffe[1] and with Sun's Linux JDK running under the

OpenBSD Unix-like system. For the mass market, I've used Sun's Java 2 Win32 (Windows 95/98/NT) implementation. And, "for the rest of us," I've run some of the programs on Apple's MacOS Runtime for Java (MRJ) running under MacOS 8 on a Power Macintosh and a few on MacOS X (which Apple wants you to pronounce "Oh Ess Ten," despite the way they've been writing it for the last three years). However, since Java is portable, I anticipate that the examples will work on MacOS X except where extra APIs are required. Not every example has been tested on every platform, but all have been tested on at least one, and most on more than one.

[1] Kaffe, the Swedish word for coffee, is an open source (GNU Public License) Java implementation that runs on just about any Unix or Unix-like system, and has been ported to other platforms such as Win32.

The Java API consists of two parts, core APIs and non-core APIs. The core is, by definition, what's included in the JDK that you download for free from http://java.sun.com. Non-core is everything else. But even this "core" is far from tiny: it weighs in at around 50 packages and well over a thousand public classes, each with up to 30 or more public methods. Programs that stick to this core API are reasonably assured of portability to any Java 2 platform.

The non-core APIs are further divided into standard extensions and non-standard extensions. All standard extensions have package names beginning with `javax.`,[2] and reference implementations are available from Sun. A Java licensee (like, say, Apple or Microsoft) is not required to implement every standard extension, but if they do, the interface of the standard extension should be adhered to. This book will call your attention to any code that depends on a standard extension. There is little code that depends on non-standard extensions other than code listed in the book itself (the major exception is the Regular Expressions API used in Chapter 4). My own package, `com.darwinsys.util`, contains some utility classes used here and there; you will see an import for this at the top of any file that uses classes from it.

[2] Note that not all packages named `javax.` are extensions: `javax.swing` and its sub-packages -- the Swing GUI packages -- used to be extensions, but are now core.

Other Books

There is a lot of useful information packed into this book. However, due to the breadth of topics, it is not possible to give book-length treatment to any one topic. Because of this, the book also contains references to many web sites and other books. This is in keeping with my target audience: the person who wants to learn more about Java.

O'Reilly & Associates publishes one of the largest -- and, I think, the best -- selection of Java books on the market. As the API continues to expand, so does the coverage. You can find the latest versions and ordering information on O'Reilly's Java books in the back pages of this book or online at http://java.oreilly.com, and you can buy them at most bookstores, both physical and virtual. You can also read them online through a paid subscription service; see http://safari.oreilly.com. While many are mentioned at appropriate spots in the book, a few deserve special mention here.

First and foremost, David Flanagan's *Java in a Nutshell* offers a brief overview of the language and API, and a detailed reference to the most essential packages. This is handy to keep beside your computer.

Learning Java, by Patrick Niemeyer and Joshua Peck, contains a slightly more leisurely introduction to the language and the APIs.

A definitive (and monumental) description of programming the Swing GUI is *Java Swing*, by Robert Eckstein, Marc Loy, and Dave Wood.

Java Servlets, by Jason Hunter, and *JavaServer Pages,* by Hans Bergsten, are both ideal for the server-side web developer.

Java Virtual Machine, by Jon Meyer and Troy Downing, will intrigue the person who wants to know more about what's under the hood.

Java Network Programming and *Java I/O*, by Elliotte Rusty Harold, and *Database Programming with JDBC and Java*, by George Reese, are also useful references.

There are many more; see the O'Reilly web site for an up-to-date list.

Other Java Books

Never consider releasing a GUI application unless you have read Sun's official *Java Look and Feel Design Guidelines* (Addison Wesley). This work presents the views of a large group of human factors and user-interface experts at Sun who have worked with the Swing GUI package since its inception; they tell you how to make it work well.

Finally, while authors at other publishing houses might be afraid to mention a book that their publisher might think of as competition to their own, I have found Patrick Chan's *Java Developer's Almanac* (Addison Wesley) a useful addition to my library and a natural complement to my book. While my book features much more detail and discussion than his short "examplets," the main part of Patrick's book is a large alphabetical (by class, not by package) reference to the core API. As the core part of his book was produced mechanically using Reflection, the book has a relatively low cover price. By the way, I show you how to generate books like Patrick's (see Section 25.8), but he doesn't show you how to write a book like mine.

General Programming Books

Donald E. Knuth's *The Art of Computer Programming* has been a source of inspiration to students of computing since its first publication by Addison Wesley in 1968. Volume 1 covers *Fundamental Algorithms*, Volume 2 is *Seminumerical Algorithms,* and Volume 3 is *Sorting and Searching*. The remaining four volumes in the projected series were never completed. Although his examples are far from Java (he invented a hypothetical assembly language for his examples), many of his discussions of algorithms -- of how computers ought to be used to solve real problems -- are as relevant today as 30 years ago.[3]

> [3] With apologies for algorithm decisions that are less relevant today given the massive changes in computing power now available.

The Elements of Programming Style, by Kernighan and Plauger, set the style (literally) for a generation of programmers with examples from various structured programming languages. Brian Kernighan also wrote (with P. J. Plauger) a pair of books, *Software Tools* and *Software Tools in Pascal*, which demonstrated so much good advice on programming that I used to advise all programmers to read them. However, these three books are somewhat dated now; many times I wanted to write a follow-on book in a more modern language, but instead defer to *The Practice of Programming*, Brian's follow-on (co-written by Rob Pike) to the *Software Tools* series. This book continues the Bell Labs (now part of Lucent) tradition of excellence in software textbooks. I have even adapted one bit of code from their book, in Section 3.14.

Design Books

Peter Coad's *Java Design* (PTR-PH/Yourdon Press) discusses the issues of object-oriented analysis and design specifically for Java. Coad is somewhat critical of Java's implementation of the observable-observer paradigm and offers his own replacement for it.

One of the most famous books on object-oriented design in recent years is *Design Patterns,* by Gamma, Helm, Johnson, and Vlissides (Addison Wesley). These authors are often collectively called "the gang of four," resulting in their book sometimes being referred to as "the GOF book." One of my colleagues called it "the best book on object-oriented design ever," and I think he's probably not far off the mark.

Another group of important books on object-oriented design is the UML series by "the Three Amigos" (Booch, Jacobson, and Rumbaugh). Their major works are the *UML User Guide*, *UML Process*, and others. A smaller and more approachable book in the same series is Martin Fowler's *UML Distilled*.

Conventions Used in This Book

This book uses the following conventions.

Programming Conventions

I use the following terminology in this book. A *program* means either an applet, a servlet, or an application. An *applet* is for use in a browser. A *servlet* is similar to an applet but for use in a server. An *application* is any other type of program. A desktop application (a.k.a. *client*) interacts with the user. A server program deals with a client indirectly, usually via a network connection.

The examples shown are in two varieties. Those that begin with zero or more import statements, a Javadoc comment, and a public class statement are complete examples. Those that begin with a declaration or executable statement, of course, are excerpts. However, the full versions of these excerpts have been compiled and run, and the online source includes the full versions.

Recipes are numbered by chapter and number, so, for example, Recipe 7.5 refers to the fifth recipe in Chapter 7.

Typesetting Conventions

The following typographic conventions are used in this book:

Italic

> is used for commands, filenames, and sample URLs. It is also used to define new terms when they first appear in the text.

Constant width

> is used in code examples to show partial or complete Java source code program listings. It is also used for class names, method names, variable names, and other fragments of Java code.

Many programs are accompanied by an example showing them in action, run from the command line. These will usually show a prompt ending in either `$` for Unix or `>` for Microsoft, depending on

which computer I was using that day. Text before this prompt character can be ignored; it will be a pathname or a hostname, again depending on the system.

As mentioned earlier, I've tested all the code on at least one of the reference platforms, and most on several. Still, there may be platform dependencies, or even bugs, in my code or in some important Java implementation. Please report any errors you find, as well as your suggestions for future editions, by writing to:

O'Reilly & Associates, Inc.
101 Morris Street
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

To ask technical questions or comment on the book, send email to:

bookquestions@oreilly.com

There is an O'Reilly web site for the book, listing errata, examples, or any additional information. You can access this page at:

http://www.oreilly.com/catalog/javacook/

I also have a personal web site for the book:

http://javacook.darwinsys.com

Both sites will list errata and plans for future editions. You'll also find the source code for all the Java code examples to download; *please* don't waste your time typing them in again! For specific instructions, see the next section.

Getting the Source Code

From my web site http://javacook.darwinsys.com, just follow the Download link and you will be presented with three choices:

1. Download the entire source archive as a single large zip file
2. Download individual source files, indexed alphabetically as well as by chapter
3. Download the binary JAR file for the `com.darwinsys.util` package needed to compile many of the other programs

Most people will choose either #1 or #2, but anyone who wants to compile my code will need #3. See Section 1.5 for information on using these files.

Downloading the entire source archive (#1) gives a large zip file that contains all the files from the book (and more). This archive can be unpacked with *jar* (see Section 23.4), the free zip program from Info-ZIP, the commercial WinZip or PKZIP, or any compatible tool. The files are organized into subdirectories by topic; there is one for strings (Chapter 3), regular expressions (Chapter 4), numbers (Chapter 5) and so on. The archive also contains the index by name and index by chapter files from the download site, so you can easily find the files you need.

Downloading individual files is easy too: simply follow the links either by the file/subdirectory name or by chapter. Once you see the file you want in your browser, use File->Save or the equivalent, or just copy and paste it from the browser into an editor or IDE.

The files will be updated periodically, so if there are differences between what's printed in the book and what you get, be glad, for you'll have received the benefit of hindsight.

Acknowledgments

My life has been touched many times by the flow of the fates bringing me into contact with the right person to show me the right thing at the right time. Steve Munroe, with whom I've long since lost touch, introduced me to computers -- in particular an IBM 360/30 at the Toronto Board of Education that was bigger than a living room, had 32 or 64K of memory, and had perhaps the power of a PC/XT -- in 1970. (Are you out there somewhere, Steve?) Herb Kugel took me under his wing at the University of Toronto while I was learning about the larger IBM mainframes that came later. Terry Wood and Dennis Smith at the University of Toronto introduced me to mini- and micro-computers before there was an IBM PC. On evenings and weekends, the Toronto Business Club of Toastmasters International (http://www.toastmasters.org) and Al Lambert's Canada SCUBA School allowed me to develop my public speaking and instructional abilities. Several people at the University of Toronto, but especially Geoffrey Collyer, taught me the features and benefits of the Unix operating system at a time when I was ready to learn it.

Greg Davidson of UCSD taught the first Learning Tree course I attended, and welcomed me as a Learning Tree instructor. Years later, when the Oak language was about to be released on Sun's web site, Greg encouraged me to write to James Gosling and find out about it. James's reply of March 29th, 1995, that the lawyers had made them rename the language to Java and that it was "just now" available for download, is the prized first entry in my saved Java mailbox. Mike Rozek took me on as a Learning Tree course author for a Unix course and two Java courses. After Mike's departure from the company, Francesco Zamboni, Julane Marx, and Jennifer Urick in turn provided product management of these courses. Jennifer also arranged permission for me to "reuse some code" in this book that had previously been used in my Java course notes. Finally, thanks to the many Learning Tree instructors and students who showed me ways of improving my presentations. I still teach for "The Tree" and recommend their courses for the busy developer who wants to zero in on one topic in detail over four days. Their web site is http://www.learningtree.com.

Closer to this project, Tim O'Reilly believed in "the little Lint book" when it was just a sample chapter, enabling my early entry into the circle of O'Reilly authors. Years later, Mike Loukides encouraged me to keep trying to find a Java book idea that both he and I could work with. And he stuck by me when I kept falling behind the deadlines. Mike also read the entire manuscript and made many sensible comments, some of which brought flights of fancy down to earth. Jessamyn Read turned many faxed and emailed scratchings of dubious legibility into the quality illustrations you see in this book. And many, many other talented people at O'Reilly & Associates helped put this book into the form in which you now see it.

I also must thank my reviewers, first and foremost my dear wife Betty Cerar, who may still think Java is some kind of caffeinated beverage that I drink while programming, but whose passion for clear expression and correct grammar has benefited much of my writing. Jonathan Knudsen, Andy Oram, and David Flanagan commented on the outline when it was little more than a list of chapters and recipes, and yet were able to see the kind of book it could become, and to suggest ways to make it better. Learning Tree instructor Jim Burgess read most of the book with a very critical eye on locution, formulation, and code. Bil Lewis and Mike Slinn (mslinn@mslinn.com) made helpful comments on multiple drafts of the book. Ron Hitchens (ron@ronsoft.com) and Marc Loy carefully read the entire final draft. Editor Sue Miller helped shepherd the manuscript

through the somewhat energetic final phases of production. Sarah Slocombe read the XML chapter in its entirety and made many lucid suggestions, though unfortunately time did not permit me to include all of them. Each of these people made this book better in many ways, particularly by suggesting additional recipes or revising existing ones. Any faults that remain are surely my own.

I've used a variety of tools and operating systems in preparing, compiling, and testing the book. The developers of OpenBSD (http://www.openbsd.org), "the proactively secure Unix-like system," deserve thanks for making a stable and secure Unix clone that is also closer to traditional Unix than other freeware systems. I used the *vi* editor (*vi* on OpenBSD and *vim* on MS-Windows) while inputting the original manuscript in XML, and Adobe FrameMaker to format the documents. Each of these is an excellent tool in its own way. If you're wondering how I got from XML to Frame, the answer will be given in Chapter 21.

No book on Java would be complete without a quadrium[4] of thanks to James Gosling for inventing the first Unix Emacs, the *sc* spreadsheet, the NeWS window system, and Java. Thanks also to his employer Sun Microsystems (NASDAQ SUNW) for creating not only the Java language but an incredible array of Java tools and API libraries freely available over the Internet.

[4] It's a good thing he only invented four major technologies, not five, or I'd have to rephrase that to avoid infringing on an Intel trademark.

Thanks to Tom and Nathan, for the *Perl Cookbook*. Without them I might never have come up with the format for this book.

Willi Powell of Apple Canada provided MacOS X access.

Thanks to the Tim Horton's Donuts in Bolton, Ontario for great coffee and for not enforcing the 20-minute table limit on the weird guy with the computer.

To each and every one of you, my sincere thanks.