Preface

"I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I–
I took the one less traveled by,
And that has made all the difference."

—Robert Frost

GENTLE READER, it gives us great pleasure to bring this book to you. This work is the product of two decades of discussions about the art of programming, as well as the art of *teaching* programming. We feel the result is a positive step toward introducing our craft to students with all levels of interest in computer science. Teaching, as we do, at liberal arts institutions, it is interesting that frequently the best students in introductory classes dabble in programming as passersby. Our special hope is that this book conveys to those individuals what makes us computer scientists.

Pedagogy. At the time of this writing, the academic community is considering many fundamental questions; their answers will dictate the immediate future of computer science education.

Is Java an appropriate teaching language? We believe it is. Historically, it has been unusual that a single language has determined the direction of both academics and industry. Today, Java[1] plays an important role in bringing relatively modern programming design concepts to both the classroom and the office. To the extent that this is true, it is easy to see that concepts learned in the classroom can become lifelong skills for young programmers. Most importantly, though, we see that Java is an effective pedagogical tool that should be used to teach students the successful programming strategies that have been developed over the past several decades.

Should object-oriented languages be taught early in the professional life of a programmer? We guardedly answer *yes*. Java is a powerful object-oriented language, and many of the features of Java are important to understanding how effective programming can be done in light of, say, portable computing and the Internet. Yet, when considering the limited time students have to learn basic material, it behooves the teacher to limit features to the most potent and palatable few. This book does just that.

Why not use graphics to motivate students? Indeed: why not! We have used graphical examples in our classes for many years, and have been quite successful. Now that a rich graphical environment is widely available within

[1] Java is a trademark of Sun Microsystems Corporation.

a language suitable for teaching, we are all the more likely to be successful. We must not, however, be distracted from teaching students the fundamental principles of programming that are so often overlooked.

To help instructors develop a stable environment (in light of the sifting sands of Java), we developed the `element` package—a free, open, clean, well-defined environment that supports a simple toolbox for object-oriented graphics. It may be downloaded from `http://www.mhhe.com/javaelements`, the McGraw-Hill web site. Avoiding direct use of the AWT (Abstract Windowing Toolkit) has enabled us to insulate the sprouting programmer from a system whose motivation in design is not necessarily ease of use in the classroom. When the student is ready to make the transition to using the AWT, we have sought to maintain as consistent a viewpoint on graphics as possible. This package also provides textual input and output facilities that (we believe) are necessary in one's first programming environment.

Using the Text. The path through the text involves a number of important steps. First, we introduce the *use* of objects almost immediately. The *design* of objects occurs relatively late in the book. At that time, we believe the student will have the maturity necessary to construct larger projects.

Recursion is introduced relatively early and can be avoided, but our experience is that students enjoy the naïve novelty of the use of recursion to solve all sorts of problems. At one time, recursion was considered wasteful of machine resources, but now it is an important programming tool at nearly every level of programming.

We have had to select a particular order for Chapters 5 (strings) through 8 (classes), but there is nothing particularly important to approaching this material in this order. For example, we introduce strings before arrays, but nothing within the chapters obviates approaching the material in the reverse order. In particular, some instructors looking for an "objects-first" approach will find they can get a significant jump on the development of classes by addressing the material of Chapter 8 first.

The later chapters should be considered hooks into subsequent parts of the computer science curriculum—recursive data structures (here, lists) are, of course, a useful introduction to the subtleties of data structure design. They provide, in fact, an alternative implementation to the one discussed in the follow-on book, *Java Structures*. The chapter on threads is provided for those students interested in programming multiple-agent programs. Concurrency is an increasingly potent tool for solving problems, and Java makes that possible within the first semester. Finally, our chapter on machines is a light introduction to three architectures—the Java Virtual Machine, the Turing Machine, and the P-RAM. Each is a *virtual machine* that has had a significant effect on the course of computer science.

We have worked hard to build in a number of resources for students to test their progress.

- Within the running text of each chapter are a number of *exercises*. These are the type of problem often written in the corner of a board during

lecture, and are meant to test progress as you read along. Some exercises are solved immediately, and some are left completely to the reader. Timing is, we believe, everything.

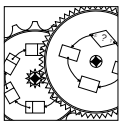
- Near the end of each chapter we include *problems*. Earlier problems are usually more easily solved than the later problems. The final problems in each chapter may be potentially very difficult to solve at this level. They are proposed to demonstrate that not everything interesting can be solved by a semester of programming. Most have been recently considered by one of the authors in a context other than this text. Answers (or distillations of answers (or hints)) to the starred ($\star$) problems are found in Appendix A (for *A*nswers).

- At the very end of each chapter are more formally presented *laboratory projects*. Many institutions teach this course with an attendant lab section; these labs are important forms of mental exercise, and incorporate the material of the preceding chapter or chapters. Resources associated with these labs are also available on-line at the McGraw-Hill web site.

- Finally, we include, in Appendix C (for *C*ontest), some of our favorite programming problems. We believe that solutions to these problems can be developed by thoughtful, first-time programmers. With time, we expect that a student's solution to these problems will become more informed. These problems are the type often considered in programming contests, and we cast them in that light here.

Successful students of this text will have become *programmers*. The *practice* of programming is only a matter of development. As with players of games like chess or go, there should be some pride in simply having learned the rules in the process of having played a few experimental games. From then on, the process involves a considerable amount of continued experimentation. We hope we've provided the motivation to write lots of Java programs. The process, it seems, rarely ends. We write programs as frequently as possible and we hope we never drop the possibility of honing our skills.

Following Along. A number of features of this book make it unique. First, all the software in this book was extracted from on-line code. Thus, we can be reasonably sure that everything we say here is "as true as possible." We've placed two different icons in the margin (see right) to highlight the details of the text. The "meshing gears" icon is used to highlight working programs that can be found at our web site. The "compass" icon identifies important principles of program design we have sprinkled throughout the text. For example, we suggest that



`example`

Principle 1 *A useful principle is not fact, but a guide.*

The observant reader realizes that even *with* a compass, true north is slightly elusive. We hope our principles set you in the right direction.

Acknowledgments. The process of bringing a book to life is unlike any other. We found our best conversations on Java were held at the many informal restaurants in and about Amherst. The establishments mentioned in this text have literally given us the energy to attack a sticky problem, and we recommend them to you. Our collegial working environments—Department of Mathematics and Computer Science at Amherst and the Department of Computer Science at Williams—have provided a fruitful medium for discussing nearly every detail of this text. David Jacobs (Clemson), John Rager (Amherst), Jim Roberts (CMU), Dale Skrien (Colby), Roman Swiniarski (San Diego State), Deborah Trytten (University of Oklahoma), reviewed our text. Their comments have made this a more accurate and accessible text, and we thank them for sharing their time. Discussions with John Rager, Lyle McGeoch (Amherst), and Andrea Danyluk (Williams) have directly shaped this text. Lyle, in fact, authored the original console windowing model; we thank him for his insights there. Our editors, Betsy Jones, Kelley Butcher, and Christine Parker have been extraordinarily patient with this work. We are indebted to Betsy Blumenthal for her careful proofreading of these pages. Gino Cieslik designed our cover. Finally, we would thank our families—Leeta, Mary, Megan, Kate, Duane "Ryan"—for their love and enduring support.

Enjoy!

Duane A. Bailey, Williams
Duane W. Bailey, Amherst