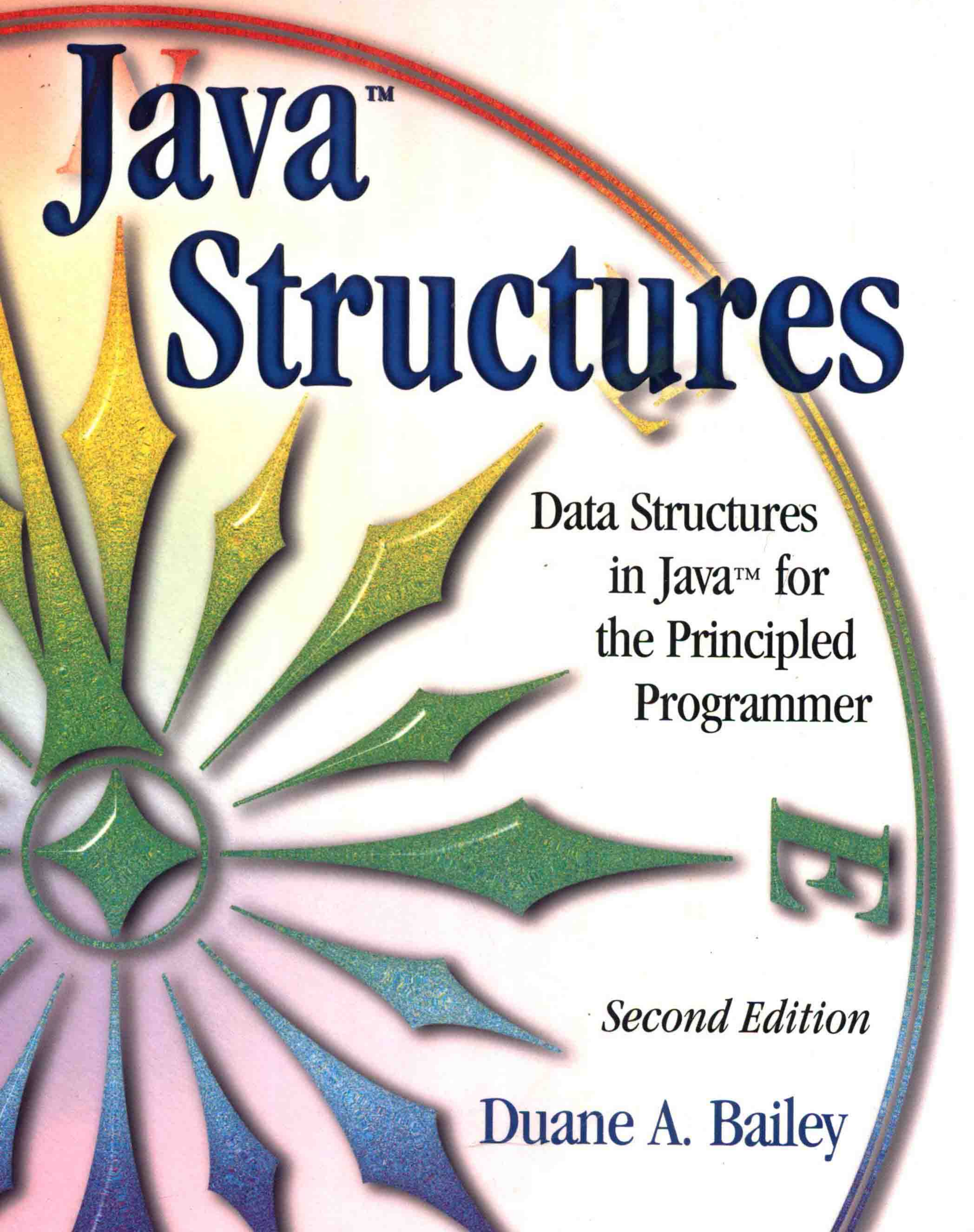
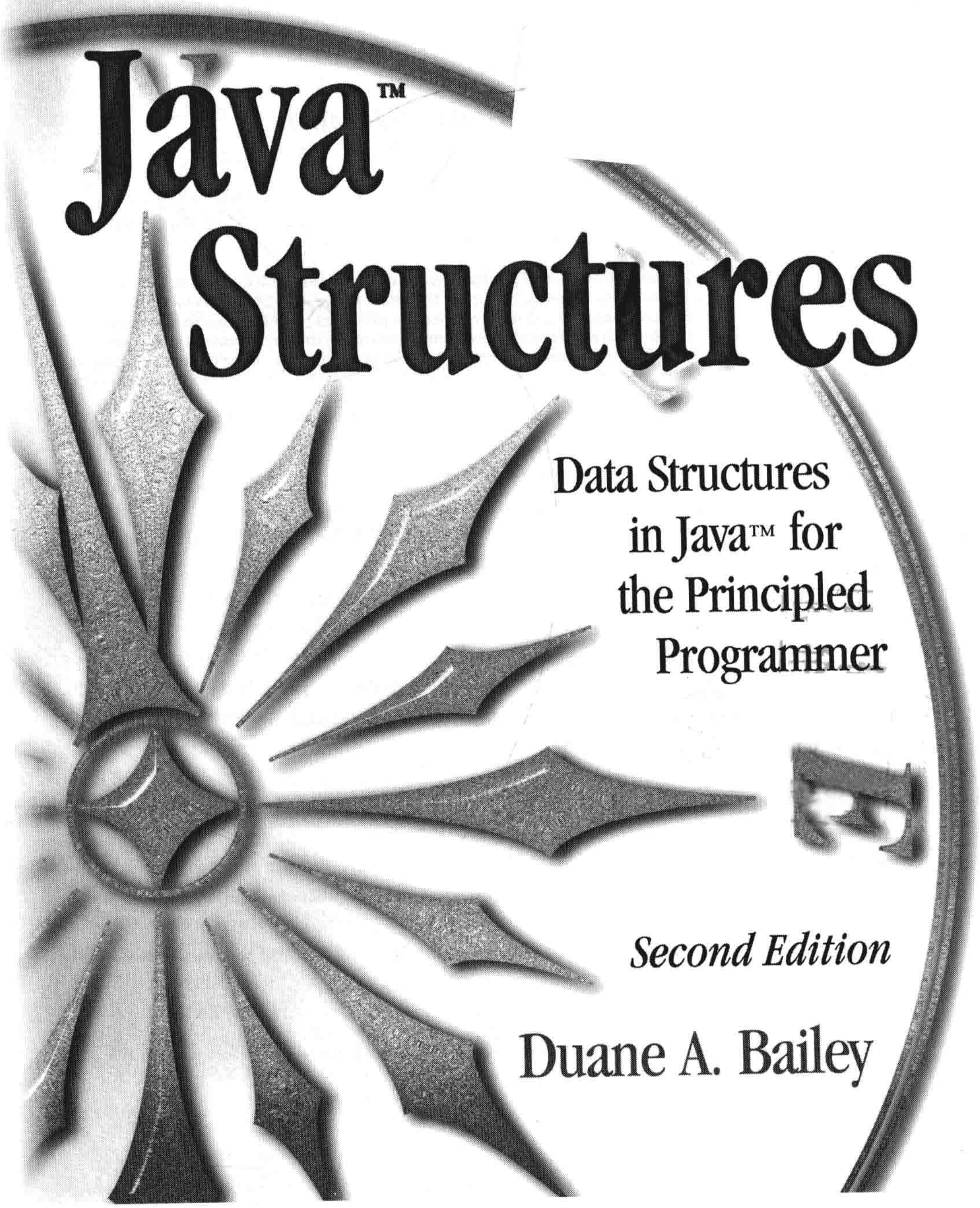
Java™ Structures

Data Structures in Java™ for the Principled Programmer

Second Edition

Duane A. Bailey





Java™ Structures

Data Structures in Java™ for the Principled Programmer

Second Edition

Duane A. Bailey

Mc Graw Hill

Boston Burr Ridge, IL Dubuque, IA Madison, WI New York San Francisco St. Louis
Bangkok Bogotá Caracas Kuala Lumpur Lisbon London Madrid Mexico City
Milan Montreal New Delhi Santiago Seoul Singapore Sydney Taipei Toronto

McGraw-Hill Higher Education

A Division of The **McGraw-Hill** *Companies*

JAVA™ STRUCTURES: DATA STRUCTURES IN JAVA™ FOR THE PRINCIPLED PROGRAMMER, SECOND EDITION

This book is printed on acid-free paper.

International	1 2 3 4 5 6 7 8 9 0 QPF/QPF 0 9 8 7 6 5 4 3 2
Domestic	1 2 3 4 5 6 7 8 9 0 QPF/QPF 0 9 8 7 6 5 4 3 2

ISBN 0–07– 239909–0
ISBN 0–07–112163–3 (ISE)

Publisher: *Elizabeth A. Jones*
Sponsoring editor: *Kelly Lowery*
Developmental editor: *Melinda Dougharty*
Executive marketing manager: *John Wannemacher*
Senior project manager: *Joyce M. Berendes*
Senior production supervisor: *Laura Fuller*
Media project manager: *Jodi K. Banowetz*
Senior media technology producer: *Phillip Meek*
Designer: *David W. Hash*
Cover designer: *Rokusek Design*
Compositor: *Interactive Composition Corporation*
Typeface: *10/12 Computer Modern*
Printer: *Quebecor World Fairfield, PA*

Library of Congress Cataloging-in-Publication Data

Bailey, Duane A.
Java™ structures : data structures in Java™ for the principled programmer / Duane A. Bailey. — 2nd ed.
p. cm.
Includes bibliographical references and index.
ISBN 0–07–239909–0 — ISBN 0–07–112163–3 (ISE)
1. Java (Computer program language). 2. Data structures (Computer science). I. Title.

QA76.73.J38 B34 2003

2002070870
CIP

www.mhhe.com

for Mary,
my wife and best friend

without
the model of my mentors,
the comments of my colleagues,
the support of my students,
the friendship of my family
this book would never be

thank you!

Preface to the First Edition

"IT'S A WONDERFUL TIME TO BE ALIVE." At least that's what I've found myself saying over the past couple of decades. When I first started working with computers, they were resources used by a privileged (or in my case, persistent) few. They were physically large, and logically small. They were cast from iron. The challenge was to make these behemoths solve complex problems quickly.

Today, computers are everywhere. They are in the office and at home. They speak to us on telephones; they zap our food in the microwave. They make starting cars in New England a possibility. Everyone's using them. What has aided their introduction into society is their diminished size and cost, and increased capability. The challenge is to make these behemoths solve complex problems quickly.

Thus, while the computer and its applications have changed over time, the challenge remains the same: *How can we get the best performance out of the current technology?* The design and analysis of data structures lay the fundamental groundwork for a scientific understanding of what computers can do efficiently. The motivations for data structure design work accomplished three decades ago in assembly language at the keypunch are just as familiar to us today as we practice our craft in modern languages on computers on our laps. The focus of this material is the identification and development of relatively *abstract* principles for structuring data in ways that make programs efficient in terms of their consumption of resources, *as well as efficient in terms of "programmability."*

In the past, my students have encountered this material in Pascal, Modula-2, and, most recently, C++. None of these languages has been ideal, but each has been met with increasing expectation. This text uses The Java Programming Language[1]—"Java"—to structure data. Java is a new and exciting language that has received considerable public attention. At the time of this writing, for example, Java is one of the few tools that can effectively use the Internet as a computing resource. That particular aspect of Java is not touched on greatly in this text. Still, Internet-driven applications in Java will need supporting data structures. This book attempts to provide a fresh and focused approach to the design and implementation of classic structures in a manner that meshes well with existing Java packages. It is hoped that learning this material in Java will improve the way working programmers craft programs, and the way future designers craft languages.

Pedagogical Implications. This text was developed specifically for use with CS2 in a standard Computer Science curriculum. It is succinct in its approach, and requires, perhaps, a little more effort to read. I hope, though, that this text

[1] Java is a trademark of Sun Microsystems, Incorporated.

becomes not a brief encounter with object-oriented data structure design, but a touchstone for one's programming future.

The material presented in this text follows the syllabus I have used for several years at Williams. As students come to this course with experience using Java, the outline of the text may be followed directly. Where students are new to Java, a couple of weeks early in the semester will be necessary with a good companion text to introduce the student to new concepts, and an introductory Java language text or reference manual is recommended. For students that need a quick introduction to Java we provide a tutorial in Appendix B. While the text was designed as a whole, some may wish to eliminate less important topics and expand upon others. Students may wish to drop (or consider!) the section on induction (Section 4.2.2). The more nontraditional topics—including, for example, iteration and the notions of symmetry and friction—have been included because I believe they arm programmers with important mechanisms for implementing and analyzing problems. In many departments the subtleties of more advanced structures—maps (Chapter 14) and graphs (Chapter 15)—may be considered in an algorithms course. Chapter 5, a discussion of sorting, provides very important motivating examples and also begins an early investigation of algorithms. The chapter may be dropped when better examples are at hand, but students may find the refinements on implementing sorting interesting.

Associated with this text is a Java package of data structures that is freely available over the Internet for noncommercial purposes. I encourage students, educators, and budding software engineers to download it, tear it down, build it up, and generally enjoy it. In particular, students of this material are encouraged to follow along with the code online as they read. Also included is extensive documentation gleaned from the code by `javadoc`. All documentation—within the book and on the Web—includes pre- and postconditions. The motivation for this style of commenting is provided in Chapter 2. While it's hard to be militant about commenting, this style of documentation provides an obvious, structured approach to minimally documenting one's methods that students can appreciate and users will welcome. These resources, as well as many others, are available from McGraw-Hill at `http://www.mhhe.com/javastructures`.

`List`

Three icons appear throughout the text, as they do in the margin. The top "compass" icon highlights the statement of a *principle*—a statement that encourages abstract discussion. The middle icon marks the first appearance of a particular class from the `structure` package. Students will find these files at McGraw-Hill, or locally, if they've been downloaded. The bottom icon similarly marks the appearance of example code.

`nim`

Finally, I'd like to note an unfortunate movement away from studying the implementation of data structures, in favor of studying applications. In the extreme this is a disappointing and, perhaps, dangerous precedent. The design of a data structure is like the solution to a riddle: the process of developing the answer is as important as the answer itself. The text may, however, be used as a reference for using the `structure` package in other applications by selectively avoiding the discussions of implementation.

Preface to the Second Edition

Since the first edition of *Java Structures* support for writing programs in Java[2] has grown considerably. At that time the Java Development Toolkit consisted of 504 classes in 23 packages[3] In Java 1.2 (also called Java 2) Sun rolled out 1520 classes in 59 packages. This book is ready for Java 1.4, where the number of classes and packages continues to grow.

Most computer scientists are convinced of the utility of Java for programming in a well structured and platform independent manner. While there are still significant arguments about important aspects of the language (for example, support for generic types), the academic community is embracing Java, for example, as the subject of the Computer Science Advanced Placement Examination.

It might seem somewhat perplexing to think that many aspects of the original Java environment have been retracted (or *deprecated*) or reconsidered. The developers at Sun have one purpose in mind: to make Java the indispensable language of the current generation. As a result, documenting their progress on the development of data structures gives us valuable insight into the process of designing useful data structures for general purpose programming. Those students and faculty considering a move to this second edition of *Java Structures* will see first-hand some of the decisions that have been made in the intervening years. During that time, for example, the `Collection`-based classes were introduced, and are generally considered an improvement. Another force—one similar to calcification—has left a trail of backwards compatible features that are sometimes difficult to understand. For example, the `Iterator` class was introduced, but the `Enumeration` class was not deprecated. One subject of the first edition—the notion of `Comparable` classes—has been introduced into a number of important classes including `String` and `Integer`. This is a step forward and a reconsideration of what we have learned about that material has lead to important improvements in the text.

Since the main purpose of the text is to demonstrate the design and behavior of traditional data structures, we have not generally tracked the progress of Java where it blurs the view. For example, Java 2 introduces a `List` interface (we applaud) but the `Vector` class has been extended to include methods that are, essentially, motivated by linked lists (we wonder). As this text points out frequently, the purpose of an interface is often to provide *reduced* functionality. If the data structure does not *naturally* provide the functionality required by the application, it is probably not an effective tool for solving the problem: search elsewhere for an effective structure.

[2] The Java Programming Language is a trademark of Sun Microsystems, Incorporated.
[3] David Flanagan, et al., *Java in a Nutshell*, O'Reilly & Associates.

As of this writing, more than 100,000 individuals have searched for and downloaded the `structure` package. To facilitate using the comprehensive set of classes with the Java 2 environment, we have provided a number of features that support the use of the `structure` package in more concrete applications. Please see Appendix C.

Also new to this edition are more than 200 new problems, several dozen exercises, and over a dozen labs we regularly use at Williams.

Acknowledgments. Several students, instructors, and classes have helped to shape this edition of *Java Structures*. Parth Doshi and Alex Glenday—diligent Williams students—pointed out a large number of typos and stretches of logic. Kim Bruce, Andrea Danyluk, Jay Sachs, and Jim Teresco have taught this course at Williams over the past few years, and have provided useful feedback. I tip my hat to Bill Lenhart, a good friend and advisor, who has helped improve this text in subtle ways. To Sean Sandys I am indebted for showing me new ways to teach new minds.

The various reviewers have made, collectively, hundreds of pages of comments that have been incorporated (as much as possible) into this edition: Eleanor Hare and David Jacobs (Clemson University), Ram Athavale (North Carolina State University), Yannick Daoudi (McGill University), Walter Daugherty (Texas A&M University), Subodh Kumar (Johns Hopkins University), Toshimi Minoura (Oregon State University), Carolyn Schauble (Colorado State University), Val Tannen (University of Pennsylvania), Frank Tompa (University of Waterloo), Richard Wiener (University of Colorado at Colorado Springs), Cynthia Brown Zickos (University of Mississippi), and my good friend Robbie Moll (University of Massachusetts). Deborah Trytten (University of Oklahoma) has reviewed both editions! Still, until expert authoring systems are engineered, authors will remain human. Any mistakes left behind or introduced are purely those of the author.

The editors and staff at McGraw-Hill–Kelly Lowery, Melinda Dougharty, John Wannemacher, and Joyce Berendes–have attempted the impossible: to keep me within a deadline. David Hash, Phil Meek, and Jodi Banowetz are responsible for the look and feel of things. I am especially indebted to Lucy Mullins, Judy Gantenbein, and Patti Evers whose red pens have often shown me a better way.

Betsy Jones, publisher and advocate, has seen it all and yet kept the faith: thanks.

Be aware, though: long after these pages are found to be useless folly, my best work will be recognized in my children, Kate, Megan, and Ryan. None of these projects, of course, would be possible without the support of my best friend, my north star, and my partner, Mary.

Enjoy!

Duane A. Bailey
Williamstown, May 2002

Contents

Chapter 0

Introduction

Concepts:
- ▷ Approaches to this material
- ▷ Principles

This is an important notice.
Please have it translated.
—The Phone Company

YOUR MOTHER probably provided you with constructive toys, like blocks or Tinker Toys[1] or Legos. These toys are educational: they teach us to think spatially and to build increasingly complex structures. You develop modules that can be stuck together and rules that guide the building process.

If you are reading this book, you probably enjoyed playing with constructive toys. You consider writing programs an artistic process. You have grown from playing with blocks to writing programs. The same guidelines for building structures apply to writing programs, save one thing: there is, seemingly, no limit to the complexity of the programs you can write.

I lie.

Well, almost. When writing large programs, the *data structures* that maintain the data in your program govern the space and time consumed by your running program. In addition, large programs take time to write. Using different structures can actually have an impact on how long it takes to *write* your program. Choosing the wrong structures can cause your program to run poorly or be difficult or impossible to implement effectively.

Thus, part of the program-writing process is choosing between different structures. Ideally you arrive at solutions by analyzing and comparing their various merits. This book focuses on the creation and analysis of traditional data structures in a modern programming environment, The Java Programming Language, or Java for short.

0.1 Read Me

As might be expected, each chapter is dedicated to a specific topic. Many of the topics are concerned with specific data structures. The structures we will investigate are abstracted from working implementations in Java that are available to you if you have access to the Internet.[2] Other topics concern the

[1] All trademarks are recognized.
[2] For more information, see `http://www.cs.williams.edu/JavaStructures`.

"tools of the trade." Some are mathematical and others are philosophical, but all consider the process of programming well.

The topics we cover are not all-inclusive. Some useful structures have been left out. Instead, we will opt to learn the *principles of programming data structures*, so that, down the road, you can design newer and better structures yourself.

Perhaps the most important aspect of this book is the set of problems at the end of each section. *All are important for you to consider*. For some problems I have attempted to place a reasonable hint or answer in the back of the book. Why should you do problems? Practice makes perfect. I could show you how to ride a unicycle, but if you never practiced, you would never learn. If you study and understand these problems, you will find your design and analytical skills are improved. As for your mother, she'll be proud of you.

Unicycles: the ultimate riding structure.

Sometimes we will introduce problems in the middle of the running text—these problems do not have answers (sometimes they are repeated as formal problems in the back of the chapter, where they *do* have answers)—they should be thought about carefully as you are reading along. You may find it useful to have a pencil and paper handy to help you "think" about these problems on the fly.

Exercise 0.1 *Call*[3] *your Mom and tell her you're completing your first exercise. If you don't have a phone handy, drop her a postcard. Ask her to verify that she's proud of you.*

This text is brief and to the point. Most of us are interested in experimenting. We will save as much time as possible for solving problems, perusing code, and practicing writing programs. As you read through each of the chapters, you might find it useful to read through the source code online. As we first consider the text of files online, the file name will appear in the margin, as you see here. The top icon refers to files in the `structure` package, while the bottom icon refers to files supporting examples.

`Structure`

`Example`

One more point—this book, like most projects, is an ongoing effort, and the latest thoughts are unlikely to have made it to the printed page. If you are in doubt, turn to the website for the latest comments. You will also find online documentation for each of the structures, generated from the code using `javadoc`. It is best to read the online version of the documentation for the most up-to-date details, as well as the documentation of several structures not formally presented within this text.

0.2 He Can't Say That, Can He?

Sure! Throughout this book are little political comments. These remarks may seem trivial at first blush. Skip them! If, however, you are interested in ways

[3] Don't e-mail her. Call her. Computers aren't everything, and they're a poor medium for a mother's pride.

to improve your skills as a programmer and a computer scientist, I invite you to read on. Sometimes these comments are so important that they appear as *principles*:

Principle 1 *The principled programmer understands a principle well enough to form an opinion about it.*

Self Check Problems

Solutions to these problems begin on page 427.

0.1 Where are the answers for "self check" problems found?

0.2 What are features of large programs?

0.3 Should you read the entire text?

0.4 Are *principles* statements of truth?

Problems

Solutions to the odd-numbered problems begin on page 437.

0.1 All odd problems have answers. Where do you find answers to problems? (Hint: See page 437.)

0.2 You are an experienced programmer. What five serious pieces of advice would you give a new programmer?

0.3 Surf to the website associated with this text and review the resources available to you.

0.4 Which of the following structures are described in this text (see Appendix D): `BinarySearchTree`, `BinaryTree`, `BitSet`, `Map`, `Hashtable`, `List`?

0.5 Surf to `http://www.javasoft.com` and review the Java resources available from Sun, the developers of Java.

0.6 Review documentation for Sun's `java.util` package. (See the Core API Documentation at `http://www.javasoft.com`.) Which of the following data structures are available in this package: `BinarySearchTree`, `BinaryTree`, `BitSet`, `Dictionary`, `Hashtable`, `List`?

0.7 Check your local library or bookstore for Java reference texts.

0.8 If you haven't done so already, learn how to use your local Java programming environment by writing a Java application to write a line of text. (Hint: Read Appendix B.)

0.9 Find the local documentation for the `structure` package. If none is to be found, remember that the same documentation is available over the Internet from `http://www.cs.williams.edu/JavaStructures`.

0.10 Find the examples electronically distributed with the `structure` package. Many of these examples are discussed later in this text.