大学计算机教育丛书（影印版）

Java™ Structures

Data Structures in Java™ for the Principled Programmer

数据结构Java™描述

数据结构的设计原则与Java™实现

Duane A. Bailey

清华大学出版社 · WCB/McGraw-Hill
http://www.tup.tsinghua.edu.cn

Java™ Structures

Data Structures in Java™ for the Principled Programmer

数据结构 Java™ 描述

数据结构的设计原则与 Java™ 实现

Duane A. Bailey
Williams College

清华大学出版社
WCB/McGraw-Hill

(京)新登字 158 号

JAVA™ STRUCTURES: DATA STRUCTURES IN JAVA™ FOR THE PRINCIPLED PROGRAMMER/Duane A. Bailey

北京市版权局著作权合同登记号: 01-98-0003

图书在版编目(CIP)数据

Java™ Structures 数据结构 Java™描述:英文/贝利(Bailey, D.A.)著. - 影印版. - 北京:清华大学出版社,1999.10
(大学计算机教育丛书)
ISBN 7-302-02192-9

Ⅰ.J… Ⅱ.贝… Ⅲ.①数据结构-程序设计 ②Java 语言-程序设计 Ⅳ.TP311.12

中国版本图书馆 CIP 数据核字(1999)第 63356 号

出版者: 清华大学出版社(北京清华大学学研大厦,邮编 100084)
http://www.tup.tsinghua.edu.cn
印刷者: 清华大学印刷厂
发行者: 新华书店总店北京发行所
开　本: 787×960 1/16　印张: 24.25
版　次: 1999 年 12 月第 1 版　2000 年 12 月第 2 次印刷
书　号: ISBN 7-302-02192-9/TP・2142
印　数: 5001～8000
定　价: 32.00 元

出版者的话

今天，我们的大学生、研究生和教学、科研工作者，面临的是一个国际化的信息时代。他们将需要随时查阅大量的外文资料；会有更多的机会参加国际性学术交流活动；接待外国学者；走上国际会议的讲坛。作为科技工作者，他们不仅应有与国外同行进行口头和书面交流的能力，更为重要的是，他们必须具备极强的查阅外文资料获取信息的能力。有鉴于此，在国家教委所颁布的“大学英语教学大纲”中有一条规定：专业阅读应作为必修课程开设。同时，在大纲中还规定了这门课程的学时和教学要求。有些高校除开设“专业阅读”课之外，还在某些专业课拟进行英语授课。但教、学双方都苦于没有一定数量的合适的英文原版教材作为教学参考书。为满足这方面的需要，我们陆续精选了一批国外计算机科学方面最新版本的著名教材，进行影印出版。我社获得国外著名出版公司和原著作者的授权将国际先进水平的教材引入我国高等学校，为师生们提供了教学用书，相信会对高校教材改革产生积极的影响。

我们欢迎高校师生将使用影印版教材的效果、意见反馈给我们，更欢迎国内专家、教授积极向我社推荐国外优秀计算机教育教材，以利我们将《大学计算机教育丛书（影印版）》做得更好，更适合高校师生的需要。

清华大学出版社

《大学计算机教育丛书（影印版）》项目组

1999.6

Preface

Envoi

God, thou great symmetry,
Who put a biting lust in me
From whence my sorrows spring,
For all the frittered days
That I have spent in shapeless ways,
Give me one perfect thing.

—Anna Wickham

"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."*

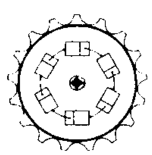
In the past, my students have encountered this material in Pascal, Modula-2, and, most recently, C++. None of these languages has been ideal, but each has been met with increasing expectation. This text uses The Java Programming Language[1]—"Java"—to structure data. Java is a new and exciting language 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

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

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.

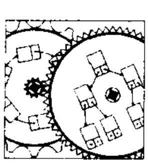
Pedagogical Implications. This text was developed specifically for use with CS2 in a standard Computer Science curriculum. It is succinct in its approach, and requires, perhaps, a little more effort to read. I hope, though, that this text becomes not a brief encounter with object-oriented data structure 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 A. 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—dictionaries (Chapter 13) and graphs (Chapter 14)—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.



`List`

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`.



`nim`

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.

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.

Acknowledgments. The trajectory of this manuscript is the product of many forces. First, all of the eating establishments mentioned within this text are real and, after extensive testing, I recommend them to Berkshire programmers, authors, and tourists. My gratitude goes to colleagues Kim Bruce and Bill Lenhart, whose use of various versions of this text in their courses has generated productive feedback. Long discussions with Kim and Bill have also had the greatest impact on design of the `structure` package. Changes to many parts of this text come from students, including Udai Haraguchi, James Rowe, Qiang Sun, and Robin Yan (all of Williams), as well as Sarah Peterson (Grinnell). The reviewers—Zoran Duric (George Mason University), William Hankley (Kansas State University), Van Howbert (Colorado State), Brian Malloy (Clemson University), Daniel D. McCracken (City College of New York), David A. Poplawski (Michigan Technical University), John E. Rager (Amherst College), Stuart Reges (University of Arizona), Susan Rodger (Duke University), Dale Skrien (Colby College), Louis Steinberg (Rutgers University), Deborah A. Trytten (University of Oklahoma), and Allen Tucker (Bowdoin College)—all worked hard to meet fast and strict deadlines. Their efforts directly shape this work. I am particularly indebted to Dale, who used this text in its rawest form at Colby, and to the institutions using this text in preprint and beta editions. Kimberly Tabtiang (University of Wisconsin) and Claude Anderson (Rose-Hulman Institute of Technology) scoured the text and code and found more errors than seemed possible. Kim is also responsible for much of the careful design and implementation of the `Graph` classes. Adams Technologies is responsible for our Web-site design. My editors at McGraw-Hill—Betsy Jones, Kelley Butcher, and Christine Parker—have kindly kept this multithreaded project on an understanding schedule. Finally, despite this work, life with Ryan, Kate, and Megan (tolerant children) and Mary (reviewer, therapist, and loving wife) remains, for me, that perfect thing.

Enjoy!

Duane A. Bailey
Williamstown, May 1998

Contents

Chapter 0

Introduction

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

---

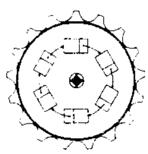
[1] All trademarks are recognized.
[2] For more information, see `http://www.mhhe.com/javastructures`.

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 probably better) structures yourself.

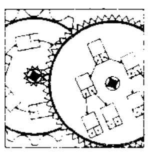
Perhaps the most important aspect of this book is the set of problems at the end of each section. *All are important for you to consider*. For some problems I have attempted to place a reasonable hint or answer in the back of the book. Why should you do problems? Practice makes perfect. I could show 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. And your mother will be proud.

Unicycles: the ultimate riding structure.

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, I'll refer to the file name 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 Web 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 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, on first blush, may not be interesting. Skip them! If, however, you are interested in ways 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.*

Now, let's get to work!

Problems

0.1⋆ All starred problems have answers. Where do you find answers to problems? (Hint: See page 355)

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

0.3 Surf to the Web site 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, "Structure Package Hierarchy"): **`BinarySearchTree`**, **`BinaryTree`**, **`BitSet`**, **`Dictionary`**, **`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 A.)

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.mhhe.com/javastructures`.

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

Chapter 1

The Object-Oriented Method

"'I will pick up the hook.
You will see something new.
Two things. And I call them
Thing One and Thing Two.
These Things will not bite you.
They want to have fun.'"
—Theodor Seuss Geisel

COMPUTER SCIENCE DOES NOT SUFFER the great history of many other disciplines. While other subjects have well-founded paradigms and methods, computer science still struggles with one important question: *What is the best method to write programs?* To date, we have no best answer. The focus of language designers is to develop programming languages that are simple to use but provide the power to accurately and efficiently describe the details of large programs and applications. The development of Java is one such effort.

Throughout this text we focus on developing data structures using *object-oriented programming*. Using this paradigm the programmer spends time developing templates for structures called *classes*. The templates are then used to construct *instances* or *objects*. A majority of the statements in object-oriented programs involve *sending messages* to objects to have them change their state. Programs involve, then, the construction and coordination of objects. In this way languages like Java are *object-oriented*.

OOP: Object-oriented programming.

In all but the smallest programming projects, *abstraction* is a useful tool for writing working programs. In programming languages such as Pascal and C, the details of a program's implementation are hidden away in its procedures or functions. This approach involves *procedural abstraction*. In object-oriented programming the details of the implementation of data structures are hidden away within its objects. This approach involves *data abstraction*. Many modern programming languages use object orientation to support basic abstractions of data. We review the details of this support in this chapter.

1.1 Data Abstraction and Encapsulation

If you purchase a muffin from the Clarksburg Bread Company you can identify it as a muffin without knowing its ingredients. Muffins are dome-shaped, bread-like, and sometimes sweet. Whether or not there's baking powder in a muffin is of little concern to you. Of course, the baker is free to switch from baking