大学计算机教育国外著名教材系列（影印版）

PEARSON
Addison Wesley

CLASSIC DATA STRUCTURES IN JAVA

经典数据结构（Java语言版）

Timothy Budd 著

清华大学出版社

图书在版编目(CIP)数据

经典数据结构 = Classic Data Structures in Java/(美)巴德(Budd, T.)著. —影印本. —北京:清华大学出版社, 2005.7
(大学计算机教育国外著名教材系列)
ISBN 7-302-11154-5

Ⅰ. 经… Ⅱ. 巴… Ⅲ. ①数据结构—高等学校—教材—英文②JAVA 语言—程序设计—高等学校—教材—英文 Ⅳ. TP311.12

中国版本图书馆 CIP 数据核字(2005)第 058947 号

出 版 者:清华大学出版社 地 址:北京清华大学学研大厦
http://www.tup.com.cn 邮 编:100084
社 总 机:010-62770175 客户服务:010-62776969
印 刷 者:北京市清华园胶印厂
装 订 者:三河市新茂装订有限公司
发 行 者:新华书店总店北京发行所
开 本:148×210 印张:19.375
版 次:2005 年 7 月第 1 版 2005 年 7 月第1 次印刷
书 号:ISBN 7-302-11154-5/TP·7370
印 数:1~3000
定 价:43.00 元

出 版 说 明

进入21世纪，世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的竞争。谁拥有大量高素质的人才，谁就能在竞争中取得优势。高等教育，作为培养高素质人才的事业，必然受到高度重视。目前我国高等教育的教材更新较慢，为了加快教材的更新频率，教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从1996年开始，与国外著名出版公司合作，影印出版了“大学计算机教育丛书(影印版)”等一系列引进图书，受到国内读者的欢迎和支持。跨入21世纪，我们本着为我国高等教育教材建设服务的初衷，在已有的基础上，进一步扩大选题内容，改变图书开本尺寸，一如既往地请有关专家挑选适用于我国高等本科及研究生计算机教育的国外经典教材或著名教材，组成本套“大学计算机教育国外著名教材系列(影印版)”，以飨读者。深切期盼读者及时将使用本系列教材的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材，以利我们把“大学计算机教育国外著名教材系列(影印版)”做得更好，更适合高校师生的需要。

清华大学出版社

PREFACE

The study of data structures is historically one of the first steps a student takes in the process of learning computer science as a discipline. Data structures play a fundamental role in almost every aspect of computer science and are found in almost every nontrivial computer program. Furthermore, the analysis of data structures may represent the first time the student encounters a number of key concepts:

- The relationship between an *algorithm* (an abstract approach used to solve a specific problem) and a *method* (the code written in a specific language, such as Java, that embodies the ideas of an algorithm)
- The division of a large program into smaller independent units and, more important, the idea that these units can be characterized by an interface independently of their implementation
- The extension of this separation of interface and implementation to the concept of an *abstract data type*, or ADT
- The reuse of standard software components in the development of new applications
- The fact that several different reusable software abstractions may satisfy the same interface and how one goes about selecting the proper component for a particular application
- The use of formal techniques to reason about both algorithms and programs—for example, how to argue that an algorithm will always terminate, regardless of its input values
- The techniques used to frame a proof of correctness for an algorithm or a method and the observation that such a proof is an essential part of software development
- The idea that an algorithm can be characterized by its execution-time performance in a fashion that is independent of any particular machine or programming language

- The relationship between mathematical concepts, such as induction, and software concepts, such as recursive procedures

And last, but not least, the student becomes familiar with a variety of specific tools that have proved themselves, over a period of many years, to be useful techniques for maintaining large collections of elements. The ideas embodied in these tools and techniques are so ingrained in computer science that they can truly be said to represent the essential foundation on which all the rest of computer science is built and the essential vocabulary in which all of computer science is discussed.

FEATURES OF THIS BOOK

Features of this book that make it different from many other data structures textbooks in common use include the following:

- The object-oriented mindset is applied throughout. Object-oriented organization is discussed from the very beginning, and object-oriented techniques are used in the development of all examples.
- The clear separation between ADT (interface) and implementation is emphasized throughout. The various categories of containers are first described as data types, independent of any implementation. These ADT descriptions are given as narrow (that is, small) interfaces. Data structures are then introduced as realizations of the interfaces. Most data structures implement several different interfaces, showing that the same container can be used in many different ways.
- A rigorous approach to development is encouraged. Techniques for developing formal proofs of programs are presented, and proofs of programs appear throughout the text.
- Sprinkled throughout the book are suggested experiments that will lead the student in exploring further aspects of data structures in more detail. A typical experiment will investigate, for example, the impact that data ordering will have on the execution time of a sorting procedure.
- As an aid to performing these experiments, the book provides tools that make it easy to create a visual record of execution time. The combination of a hands-on experiment coupled with a visual display is an effective tool for reinforcing the lessons gained from an analytical dissection of the algorithms embedded in a data structure.
- Important software patterns are highlighted, and the use of patterns is emphasized from the beginning.

- Graphical elements of Java are used to the reader's advantage. Many students are attracted to the graphical elements of Java. Yet graphical applications can quickly become complex, and the important lessons about data structures must not be lost in the process of creating a nice visual output. The graphical examples have been carefully chosen to minimize the unnecessary detail and to emphasize the important lessons.
- Sidebars are used for additional information. Throughout the text, a key linear narrative is almost always being presented. Often, additional or incidental information would help the reader but might detract from the narrative flow. In these situations, the information has been placed into boxes. Readers can then peruse the additional information as they choose.
- Each chapter ends with a chapter summary, a collection of key terms, and a series of study questions that the student can use to evaluate his or her understanding of the material. Study questions are designed so that a student who understands the material should be able to give an immediate and simple answer. In constrast, exercises require more thought and time, even if a student has understood the chapter. Finally, each chapter ends with a series of programming projects. These can be used as jumping-off points for instructors in the creation of projects for their own courses.

ASSUMED BACKGROUND

It is assumed that the student using this book will have had at least one course that introduced programming using Java. The book also assumes that the student has some mathematical background, generally no more than is typically found in high school mathematics. For example, we assume that the student has seen the definition of a logarithm and is familiar with the terms polynomial and exponential. Students who have some background in programming but not in Java should read Appendix A to learn more about the language.

HOW THIS BOOK IS ORGANIZED

- **Tools and techniques:** Chapters 1 through 5 provide the foundation on which everything else will be built. These chapters define the essential concept of the abstract data type (ADT) and describe the tools used in the evaluation and analysis of data structures.

- **Basic data abstractions:** Chapters 6 and 8 provide detailed descriptions of the two most important fundamental data abstractions: the vector and the linked list. Chapters 7 and 9 each provide an explanation of some of the more common variations on these fundamental ideas. Many of the sections in these latter chapters are marked as optional material and can be omitted at the discretion of the instructor.
- **Insertion order—preserving containers:** Chapters 10, 11, and 12 consider data structures applicable to problems in which the order that values are added to a collection is important. These data structures can be divided into three broad categories, each considered in a separate chapter.
- **Trees and their uses:** Chapters 13, 14, and 15 consider the various ways in which binary trees are used in the creation of data structures. Binary trees are important in providing a natural way to repeatedly reduce the size of a problem by half and thus permit efficient solution to many problems.
- **Advanced data structures:** Chapters 16 through 19 consider a sequence of more advanced data structures. Most are constructed as *adapters* built on top of earlier abstractions. Hash tables are introduced first as a technique for implementing simple collections and only later (in Chapter 17) as a tool for developing efficient maps. Many data structure texts explore hash tables only as a key/value data structure, leaving the mistaken impression that this is their only use.
- **Algorithms:** Chapter 20 considers the graph data type. Here, several alternative data structure representations are in common use, and the emphasis in this chapter is more on the development and analysis of useful algorithms than on any particular data structuring technique.

Appendix A provides a quick overview of Java syntax. This material is useful for students who come to this course with a background in a different programming language.

TOPICS BY PREREQUISITES

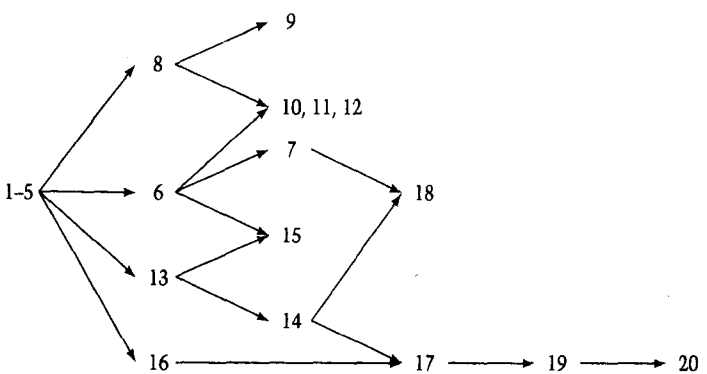
No one order of presentation can hope to suit all situations. Students in different institutions will have different backgrounds. Some instructors will want to take a more mathematical approach to this material; others, a more software engineering approach. Some instructors will think that `SkipLists` (to take just one example) are eye opening and therefore fundamental to the purpose of this course; others may think that the same abstraction is far too complex and should be omitted altogether.

I have endeavored to satisfy these diverse requirements in a number of ways. Many of the interesting but nevertheless tangential ideas encountered during the development of the classic data structures have been placed in sidebars, which can be emphasized or ignored at the instructor's discretion. Similarly, many sections are marked as optional. The instructor is free to pick and choose those optional sections as interest and schedules dictate.

Even the order of entire chapters can be rearranged. The order given here seemed most natural to me, but I nevertheless realize that natural in some cases may reflect more than a little personal bias. I will therefore explicitly describe the prerequisite chain, leaving open the possibility of approaching the material in a slightly different order.

The first five chapters are fundamental and should be covered in the order presented. However, the amount of time one elects to spend on these chapters can be adjusted, depending on the background the students will have acquired in their earlier courses. (The same can be said of the first appendix. Appendix A can be omitted if students have had previous courses in Java programming, but may need to be covered in detail if students' previous programming has been in a different language.)

Once past the introductory material, the order of dependencies between chapters can be described as follows:



Chapters 6 (vectors) and 8 (linked lists) need no more than an understanding of abstract data types, which is developed in Chapter 2. Chapter 7 (sorted vectors) presents variations on the topics of Chapter 6. In a similar fashion, Chapter 9 (list variations) discusses variations on the theme of linked lists introduced in Chapter 8. Much of the material in Chapters 7 and 9 is marked as optional; although interesting, it can be omitted with little impact on understanding the rest of the material in the text.

Chapters 10, 11, and 12 (stacks, deques, and queues, respectively) represent one large unit divided over three chapters. They require only the basic ideas developed in Chapters 6 and 8. The order of Chapters 11 and 12 is largely arbitrary. Doing 12 before 11 makes more sense from a formal or mathematical perspective, whereas doing 11 before 12 makes more sense from a software engineering perspective. However, either order is possible.

Chapter 13, on trees, is fundamental to both Chapters 14 (binary search trees) and 15 (priority queues, heaps). However, the order of the latter two chapters is arbitrary and can easily be reversed.

Chapter 16, on hash tables, does not depend on anything other than the basic ADT definitions (from Chapter 2) and a certain level of comfort in the analysis of algorithms (as will be gained throughout the book). Chapter 16 can be easily moved earlier in the course if the instructor so desires.

Chapter 17, on maps, depends loosely on Chapter 16 and more strongly on Chapters 8 and 14. Chapter 17, too, could easily be moved earlier in the course if the instructor so desires.

Chapter 18, on sets, depends on Chapter 7 and slightly more loosely on Chapters 8 and 14. If the material in Chapter 7 was slighted or omitted earlier in the course, it can easily be brought forth into this chapter.

Chapter 19, on matrices, depends only loosely on Chapter 17. Many instructors like to present the first part of this chapter much earlier, linking the matrix idea to the concept of the vector developed in Chapter 6. This can be accommodated by omitting Sections 19.7 and 19.8, which contain the only real dependencies on Chapter 17.

Finally, Chapter 20 does build on the matrix concepts discussed in Chapter 19. However, if Chapter 19 is moved earlier in the course, so too can Chapter 20.

ADVANTAGES OF USING JAVA

As a vehicle for teaching, Java is a great advance over many of the alternative langauges, such as C++ or Pascal. Unlike Pascal, Java forces the student to adopt an object-oriented mindset. The error messages from Java compilers are typically much more useful than error messages from, for example, the majority of C++ compilers. Similarly, Java performs many more runtime checks, making debugging less of a burden.

Java is a relatively simple language, having far fewer subtle dark corners than does C++. And Java is largely platform independent. This allows textbooks, such as this one, to be independent of any platform. It also allows students to move their code from one system to another. Many colleges

now permit students to develop programs on any platform of their choice, so a single class will have some students working on Windows systems, others on Macintoshes, and others on UNIX or Linux variants.

Java also allows students to create graphical applications much more easily than in most other languages. Students often find these graphical programs more appealing than the traditional text-centered example programs. Of course, it is important that the instructor ensure that the message does not become lost in the medium.

PROBLEMS INHERENT IN JAVA

Both students and instructors who have used Java in an earlier course will have encountered the fact that the language imposes the need to explain many ideas far earlier than an instructor might desire. Consider this classic first program:

```
class HelloWorld {
  public static void main (String [ ] args) {
    System.out.println("hello world!");
  }
}
```

Understanding it requires an explanation of `static` methods, classes and objects, command line arguments, and system output. Similarly, a few concepts must be mentioned in the early part of this book, although they are not relevant to the point at hand and are not used until much later. The three most notable concepts in this category are serialization, synchronization, and stream I/O.

Declaring an object as `Serializable` simply means that it can be written in binary form to a stream: for example, placed into a file. Because this is a useful property for objects to have, we declare, in Chapter 2, all our containers as subclasses of `Serializable`. However, having made this statement, nothing more is made of this property until Chapter 17, and there only fleetingly.

Similarly, honesty dictates that when it is important for a method to be declared `synchronized`, we do so, even if it occurs in an early chapter—despite the fact that few students will have ever written a program that uses multiple threads, and we will not ourselves create such a program until Chapter 12. But although we don't expect the student to be able to write multithreaded applications right from the start, the idea that such a thing is *possible* is not difficult to grasp, and hence our discussion of the

problems that the `synchronized` modifier is intended to solve should be accessible to all students, regardless of their background.

The Java I/O library is structured differently from similar libraries found in other languages. Seen in the right light, it is an excellent illustration of object-oriented design and reuse. However, at least initially, it is simply a confusingly large collection of seemingly unrelated classes. This has been made doubly so by the introduction, in Java 1.2, of two parallel libraries: one based on streams (for processing bytes) and one based on `Readers` (for processing characters). We have tried to minimize this confusion by introducing features slowly and one at a time.

An attraction of Java is that it permits the creation of graphical interfaces with far less effort than is required by many other languages. Nevertheless, it takes some time to master the AWT (Abstract Windowing Toolkit), the Java GUI building library. The author (not to mention the instructor) is faced with a quandary. Students like creating more graphical programs, such as games, but time devoted to teaching GUI concepts comes at the expense of time that could be spent teaching data structures. The examples presented in this book try to walk a middle ground. Some examples are graphical, but many are not. Those that are tend to use very simple graphics, so that the issues relating to data structures do not become lost.

WHY NOT USE THE STANDARD LIBRARY?

When I started contemplating writing a data structures textbook using Java, my first thought was to base the material on an examination of the data types found in the Java standard library as distributed by Sun Microsystems. Doing so would have many advantages. The library would be found as part of any Java implementation, thereby avoiding the necessity of distributing code along with the text. Students would be much more likely to use the standard library after finishing the course than to continue to use a textbook-specific set of classes, and so on.

However, after carefully examining the Java collection classes, I decided that to limit myself to these abstractions would simply not do justice to the material. My reasons for coming to this conclusion were as follows:

The Java library is incomplete. Major abstractions, even whole families of data types, are not covered by the code found in the standard library.

The Java library is misleading and in some places poorly designed. A good example is the `HashTable` data type. In the Java library, this is a dictionary-like data structure that is indeed one of the classic *applications* for this data type, but it prevents any discussion of the equally historical

use of the data type as a simple *set*. Furthermore, the data type is an odd mishmash of the two classic hash table techniques.

The library has wide interfaces. This is perhaps an understandable decision on the part of the designers of the Java library, as they want to get the most mileage out of the fewest data structures. But for a student audience, wide interfaces tend to obscure the important points. In my own data structures, I have purposely made very narrow interfaces, thus permitting me to spend the majority of my time addressing the key concepts.

The library confuses interfaces and implementations. Although the Java library has both interfaces and implementations, they tend to be very close to each other. Thus, there is a great tendency to program to an implementation rather than selecting the appropriate interface. It is difficult to say what, in the Java library, corresponds to an abstract data type. In my own code, I have used a larger number of smaller interfaces and consistently in my example programs illustrate the idea of programming to an interface (an ADT) rather than to an implementation.

The library shows too much evidence of evolution. The code in the standard library underwent a major revision in the 1.2 version of the language yet retained backward compatibility with the earlier library. This introduced several inconsistencies into the library, such as in naming conventions. By writing my own library, I can use a consistent and, I hope, easy-to-remember set of conventions.

The library has unsupported operations. A key idea in the Java standard library is that an implementation may claim to support an interface and nevertheless decline to respect all the methods specified by the interface. The implementation can do this by throwing an `UnsupportedOperationException` when asked to perform a method it is unable to honor. Although occasionally useful, this is a bad idea to hold up to the student as a paradigm. In the library of abstractions I describe in this book, only when I discuss open hashing and the `remove` operation do I not provide a working implementation for a method declared in the interface.

Nevertheless, I have provided, in Appendix C, a discussion of the data structures found in the Java standard library and how they relate to the categories described in this text.

OBTAINING THE CODE

I can be reached by e-mail at `budd@cs.orst.edu`. My personal Web pages are found at `http://www.cs.orst.edu/~budd/`. The library of abstractions described in this book can be downloaded from `ftp://ftp.cs.orst.edu/pub/budd/jds`.

Supplementary material for qualified instructors is available. Contact your Addison-Wesley representative or send e-mail to `\verb+aw.cse @awl.com+` for details.

ACKNOWLEDGMENTS

Many people have seen earlier drafts of this text, and their comments and suggestions have been most helpful. In particular, I wish to acknowledge Paul Benjamin (Pace University), Rebecca Djang (Oregon State University), Peter Gabrovsky (California State University), Dean Kelley (Minnesota State University, Mankato), Martha Klems (Western Illinois University), Robert Moll (University of Massachusetts, Amherst), Jim Morrison (Mankato State University), Thaddeus F. Pawlicki (University of Rochester), Carolyn Schauble (Colorado State University), Frank Tompa (University of Waterloo), Jane Turk (La Salle University), Jack Wileden (University of Massachusetts), and Salih Yurttas (Texas A&M University).

Of course, a great deal of rewriting occurred after I received all of these comments, so any remaining errors are solely my responsibility.

My editor from Addison-Wesley has once again been Susan Hartman, now Susan Hartman Sullivan. Despite some important changes going on in her life, she somehow still found time to discuss *Data Structures* with me and with the reviewers. I wish her and her new husband, Pat, all the best. Her able assistant has been Lisa Kalner, with whom I have also had the pleasure to work with on previous projects. As always, I have found Susan and Lisa, and all the rest of the Addison-Wesley team, to be professional and a pleasure to work with. As with several of my recent books, composition and layout has been produced by Paul Anagnostopoulos of Windfall Software. I am continually amazed at how Paul and his team make a job that I know is so complex look so easy.

CONTENTS