Data Structures & Problem Solving Using JAVA™

MARK ALLEN WEISS

Data Structures and Problem Solving Using Java™

Mark Allen Weiss
Florida International University

ADDISON-WESLEY
An imprint of Addison Wesley Longman, Inc.
Reading, Massachusetts • Harlow, England • Menlo Park, California • Berkeley, California
Don Mills, Ontario • Sydney • Bonn • Amsterdam • Tokyo • Mexico City

Acquisitions Editor: Susan Hartman
Associate Editor: Katherine Harutunian
Production Editor: Patricia A.O. Unubun
Design Editor: Alywn R. Velásquez
Packager: Sarah Hallet-Corey
Manufacturing Coordinator: Judy Sullivan
Copyeditor: Laura K. Michaels
Cover Designer: Syndi Hirsch

Access the latest information about Addison Wesley titles from our
World Wide Web site: `http://www.awl.com/cseng/`

Reprinted with corrections, March 1999.

Library of Congress Cataloging-in-Publication Data

```
Weiss, Mark Allen.
    Data structures and problem solving using Java / Mark A. Weiss.
       p. cm.
    Includes index.
    ISBN 0-201-54991-3
    1. Java (Computer program language)  2. Data structures (Computer
  science)  3. Problem solving -- Data processing.   I. Title
QA76.73.J38W45 1998
005.13'3--dc21                                                  97-30970
                                                                    CIP
```

ISBN 0-201-54991-3

8 9 10 MA 0100

Data Structures and Problem Solving Using Java™

Preface

This book is designed for a two-semester sequence in computer science, beginning with what is typically known as Data Structures (CS-2).

The content of the CS-2 course has been evolving for some time. While there is some general consensus concerning topic coverage, there still exists considerable disagreement over the details. One uniformly accepted topic is principles of software development, most notably the concepts of encapsulation and information hiding. Algorithmically, all CS-2 courses tend to include an introduction to running time analysis, recursion, basic sorting algorithms, and elementary data structures. An advanced course is offered at many universities that covers topics in data structures, algorithms, and running time analysis at a higher level. The material in this text has been designed for use in both levels of courses, thus eliminating the need to purchase a second textbook.

Although the most passionate debates in CS-2 center around the choice of a programming language, there are other fundamental choices that need to be made, including these:

- Whether to introduce object-oriented design or object-based design early
- The level of mathematical rigor
- The appropriate balance between the implementation of data structures and their use
- Programming details related to the language chosen (for instance, should GUIs be used early)

My goal in writing this text is to provide a practical introduction to data structures and algorithms from the viewpoint of abstract thinking and problem solving. I try to cover all of the important details concerning the data structures, their analyses, and their Java implementations, while staying away from data structures that are theoretically interesting but not widely used. It is impossible to cover all of the different data structures, including their uses and the analysis, described in this text in a single course. So, I have designed the textbook to allow instructors flexibility in topic coverage. The instructor will need to decide on an appropriate balance between practice and theory and then choose those topics

that best fit the course. As I discuss later in the Preface, the text is organized in a way that tends to minimize dependencies among the various chapters.

A Unique Approach

The text takes a unique approach by separating the data structures into their specification (via a Java `interface`) and subsequent implementation. This approach provides several benefits, including the promotion of abstract thinking. Class interfaces are written and used before the implementation is known, thus forcing the student to think at an early stage about the functionality and potential efficiency of the various data structures. For example, students will see programs using a hash table hundreds of pages before the hash table is implemented. In this textbook, the interfaces for the data structures are discussed in a single chapter in Part II. Part II also describes basic analysis techniques, recursion, and sorting. Part III contains a host of applications that use the data structures. Implementation of the basic data structures is not shown until Part IV, once the data structures have already been used. Since all of the textbook code is available (see *Code Availability*, page xii), students can design large projects early on, using existing software components. Software development tools in all languages come with large libraries, and most data structures will eventually be part of these libraries. I envision an eventual shift in emphasis of data structures courses from implementation to use.

Many instructors will prefer a more traditional approach in which each data structure is defined, implemented, and then used. Because there is no dependency between material in Parts III and IV, a traditional course can easily be taught from this book.

Prerequisites

Students using this book should have knowledge of either an object-oriented or procedural programming language. Knowledge of basic features, including primitive data types, operators, control structures, functions (methods), and input and output (but not necessarily arrays and classes) is assumed.

Students who have taken a first course using Java can begin at Chapter 3 (or perhaps later). Students who have had a first course in another language should begin at Chapter 1. They also should use the Appendix, which, combined with Part I, provides plenty of Java information. If a student would like also to use a Java reference book, some recommendations are given in Chapter 1, page 25.

Knowledge of discrete math is helpful but is not an absolute prerequisite. Although there are several mathematical proofs, many are preceded by a brief math review. Chapters 7 and 18 through 23 require some degree of mathematical sophistication. The instructor may easily elect to skip mathematical aspects of the proofs by presenting only the results. All proofs in the text are clearly marked and are separate from the body of the text.

Java

This textbook presents material using the Java programming language. Java is a relatively new language that is often examined in comparison with C++. Java offers many benefits, and programmers often view Java as a safer, more portable, and easier-to-use language than C++.

The use of Java requires that some decisions be made when writing a textbook. Some of the decisions made are as follows:

1. *Java 1.1 constructs are used exclusively*: Although at the time of this writing there is only one Java 1.1 compiler available, others are sure to follow. Please make sure you are using a compiler that is Java 1.1-compatible. The most noticeable difference in the main text is the use of Java 1.1 classes for I/O instead of deprecated classes from Java 1.0.2, including the `BufferedReader`, `FileReader`, and `InputStreamReader` classes.
2. *GUIs are not emphasized*: GUIs are a nice feature in Java, and their use is described in Appendix D. However, they seem to be an implementation detail rather than a core CS-2 topic. In keeping with the goal of the book, the Appendix is the most appropriate place to discuss GUIs. Also, this is the part of the language that changed the most in Java 1.1. It further is the most unstable part of the language.
3. *Applets are not emphasized*: Applets use GUIs. Further, the focus of the course is on data structures, rather than language features. A discussion of applets is in the Appendix. Instructors can elect to have students design applets as part of the use (or simulation) of data structures.
4. *Newer Java 1.1 features are not used*: These features include inner classes, new rules for final variables, and so on. I have attempted to use core Java and avoid excessively fancy features.
5. *The concept of a pointer is discussed when reference variables are introduced*: Java does not have a pointer type. Instead, it has a reference type. However, pointers have traditionally been an important CS-2 topic that needs to be introduced. I illustrate the concept of pointers in other languages when discussing reference variables.
6. *Threads are mentioned only in the Appendix in the context of animations*: Some members of the CS community argue that multithreaded computing should become a core CS-1/2 topic. Coverage in the Appendix will allow flexibility if professors want to cover it.

As with every programming language, Java also has some disadvantages. It does not directly support generic programing; a workaround is required that is discussed in Chapter 3. I/O support when using Java is minimal. The examples herein make minimal use of the Java I/O facilities.

Text Organization

This text introduces Java and object-oriented programming (particularly abstraction) in Part I. I discuss primitive types, reference types, and some of the predefined classes and exceptions before proceeding to the design of classes and inheritance.

Part II discusses Big-Oh and algorithmic paradigms, including recursion and randomization. An entire chapter is devoted to the topic of sorting, and a separate chapter contains a description of basic data structures. The interfaces and running times of the data structures are presented *without* the implementations being given. At this point in the text, the instructor may take several approaches to present the remaining material, including these two:

1. Use the corresponding implementations in Part IV as each data structure is described. The instructor can ask students to extend the classes in various ways, as suggested in the exercises.
2. Show how the interface is used and cover implementation at a later point in the course. The case studies in Part III can be used to support this approach. Since complete implementations are available on the Internet, the instructor can provide a library of classes for use in programming projects. Details on using this approach are given shortly.

Part V describes advanced data structures such as splay trees, pairing heaps, and the disjoint set data structure, which can be covered if time permits or, more likely, in a follow-up course.

Chapter-by-Chapter Text Organization

Part I consists of four chapters that describe the basics of Java used throughout the text. Chapter 1 describes primitive types and illustrates how to write basic programs in Java. Chapter 2 discusses reference types and illustrates the general concept of a pointer — even though Java does not have pointers — so that students learn this important CS-2 topic. Several of the basic reference types (strings, arrays, files, and string tokenizers) are illustrated, and the use of exceptions is discussed. Chapter 3 continues this discussion by describing how a class is implemented. Chapter 4 illustrates the use of inheritance in designing hierarchies (including exception classes) and generic components.

Part II focuses on the basic algorithms and building blocks. Chapter 5 provides a complete discussion of time complexity and Big-Oh notation. Binary search is also discussed and analyzed. Chapter 6 is a crucial chapter that discusses the interface to the data structures and argues intuitively what the running time of the supported operations should be for each data structure. (The implementation of these data structures is provided in Part IV.) Chapter 7 describes recursion by first introducing the notion of proof by induction. It also discusses divide-and-conquer, dynamic programming, and backtracking. A section describes several

recursive numerical algorithms that are used to implement an important encryption algorithm, the RSA cryptosystem. For many students, the material in the second half of Chapter 7 is more suitable for a follow-up course. Chapter 8 describes, codes, and analyzes several basic sorting algorithms, including the insertion sort, Shellsort, mergesort, and quicksort. It also proves the classic lower bound for sorting and discusses the related problem of selection. Finally, Chapter 9 is a short chapter that discusses random numbers, including their generation and use in randomized algorithms.

Part III provides several case studies, with each chapter organized along a general theme. Chapter 10 illustrates several important techniques by examining games. Chapter 11 discusses the use of stacks in computer languages by examining an algorithm to check for balanced symbols and the classic operator precedence parsing algorithm. Complete implementations with code are provided for both algorithms. Chapter 12 discusses the basic utilities of file compression and cross-reference generation and provides a complete implementation of the cross-reference generator. Chapter 13 broadly examines simulation by looking at one problem that can be viewed as a simulation and then at the more classic event-driven simulation. Finally, Chapter 14 illustrates how data structures are used to implement several shortest-path algorithms efficiently for graphs.

Part IV presents the data structure implementations that correspond to the interfaces in Chapter 6. Some mathematics is used in this part, especially in Chapters 18 to 20, and can be skipped at the discretion of the instructor. Chapter 15 provides implementations for both stacks and queues. These data structures are first implemented using an expanding array. Then they are implemented using linked lists. General linked lists are described in Chapter 16. Extensions such as doubly linked lists, circular linked lists, and cursor implementations are left as exercises. Chapter 17 describes trees and illustrates the basic traversal schemes. Chapter 18 is a detailed chapter that provides several implementations of binary search trees. Initially, the basic binary search tree is shown, and then a binary search tree that supports order statistics is derived. AVL trees are discussed but not implemented; however, the more practical red-black trees and AA-trees are. Finally, the B-tree is examined. Chapter 19 discusses hash tables and implements the quadratic probing scheme, after examination of a simpler alternative. Chapter 20 describes the binary heap and examines heapsort and external sorting.

Part V contains material that is suitable for a more-advanced course or for general reference. The algorithms are accessible even at the first-year level. For completeness, sophisticated mathematical analyses have been included. Chapter 21 describes the splay tree, which is a binary search tree that performs well in practice and is competitive with the binary heap in some applications that require priority queues. Chapter 22 describes priority queues that support merging operations and provides an implementation of the pairing heap. Finally, Chapter 23 examines the classic disjoint set data structure.

The Appendix contains additional Java reference material. Appendix A illustrates how to compile and run Java programs on several platforms. Appendix B lists the operators and their precedence. Appendix C summarizes the Java libraries

used in the text. Appendix D describes the AWT and applets. It also discusses threads in the context of animation.

Chapter Dependencies

Generally speaking, most chapters are independent of each other. Here are some of the notable dependencies:

- *Part I*: With the exception of the `Shape` case study in Chapter 4, all material in Part I should be covered, in sequence, prior to continuing to the rest of the text. The `interface` must be introduced, but the details of inheritance can be covered in broad detail if the instructor so chooses.
- *Chapter 5* (*Algorithm Analysis*): This should be covered prior to Chapters 6 and 8. Recursion (Chapter 7) can be covered prior to this chapter, but the instructor will have to gloss over some details about avoiding inefficient recursion.
- *Chapter 6* (*Data Structures*): This can be covered prior to, or in conjunction with, material in Part III or IV.
- *Chapter 7* (*Recursion*): The material in Sections 7.1–7.3 should be covered prior to discussing recursive sorting algorithms, trees, the tic-tac-toe case study, and shortest-path algorithms. Material such as the RSA cryptosystem, dynamic programming, and backtracking (unless tic-tac-toe is discussed) is otherwise optional.
- *Chapter 8* (*Sorting*): This should follow Chapters 5 and 7. However, it is possible to cover Shellsort without Chapters 5 and 7, since Shellsort is not recursive (hence, there is no need for Chapter 7) and a rigorous analysis of its running time is too complex and is not covered in the book (hence, there is little need for Chapter 5).
- *Chapters 15 and 16* (*Stacks/Queues/Lists*): These may be covered in either order. However, I prefer to cover Chapter 15 first, since I believe it has a simpler example of linked lists.
- *Chapters 17 and 18* (*Trees/Search trees*): These can be covered in either order or simultaneously.

Separate Entities

The other chapters have little or no dependencies:

- *Chapter 9* (*Randomization*): The material on random numbers can be covered at any point as needed.
- *Part III* (*Case Studies*): This can be covered at any point in roughly any order. There are a few references to earlier chapters that can easily be followed. These include Section 10.2 (tic-tac-toe), which references a discussion in Section 7.7, and Section 12.2 (cross-reference generation), which

references similar lexical analysis code in Section 11.1 (balanced symbol checking).

- *Chapters 19 and 20* (*Hashing/Priority Queues*): These can be covered at any point.
- *Part V* (*Advanced Data Structures*): This material is self-contained and is typically covered in a follow-up course.
- *Appendices* (*More Java*): The material on GUIs in Appendix D can be covered at any point after Chapter 4, as needed.

Mathematics

I have attempted to provide mathematical rigor for use in CS-2 courses that emphasize theory and for follow-up courses that require more analysis. However, this material stands out from the main text in the form of separate theorems and, in some cases, separate sections (or subsections). Thus it can be skipped in courses that choose to deemphasize theory.

In all cases, the proof of a theorem is not necessary to the understanding of the theorem's meaning. This is another illustration of the separation of an interface (the theorem statement) from its implementation (the proof). Some inherently mathematical material, such as Section 7.4 (*Numerical Applications of Recursion*), can be skipped without affecting comprehension of the rest of the chapter.

Course Organization

A crucial issue in teaching the course is deciding how the materials in Parts II to IV are to be used. The material in Part I should be covered in depth, and the student should write one or two programs that illustrate the design, implementation, and testing of classes and generic classes, and perhaps object-oriented design using inheritance. Chapter 5 discusses Big-Oh notation. An exercise in which the student writes a short program and compares the running time with an analysis can be given to test comprehension.

In the separation approach, the key concept of Chapter 6 is that different data structures support different access schemes with different efficiency. Students can be asked first to write an inefficient data structure. Any case study (except the tic-tac-toe example that uses recursion) can be used to test their programs, and the students can compare their inefficient data structures with an efficient library routine (provided by anonymous ftp, as discussed later in the Preface). In this scheme, all of the case studies (except tic-tac-toe) can be examined to see how each of the particular data structures is used. In this way, the student can see the interface for each data structure and how it is used but not see how it is efficiently implemented. This is truly a separation. Viewing things this way will greatly enhance the ability of students to think abstractly. Students then can be asked to extend the case study, but, once again, they are not required to know any of the details of the data structures.

The implementation of the data structures can be discussed afterward, and recursion can be introduced whenever the instructor feels it is appropriate, provided it is prior to binary search trees. The details of sorting can be discussed at any time after recursion. At this point, the course can continue by using the same case studies and experimenting with modifications to the implementations of the data structures. For instance, the student can experiment with various forms of balanced binary search trees.

Instructors who opt for a more traditional approach can simply discuss a case study in Part III after discussing a data structure implementation in Part IV. The book's chapters are designed to be as independent of each other as possible.

Exercises

Exercises come in various flavors; I have provided four varieties. The basic *In Short* exercise asks a simple question or requires hand-drawn simulations of an algorithm described in the text. The *In Theory* section asks questions that either require mathematical analysis, or perhaps, asks for theoretically interesting solutions to problems. The *In Practice* section contains simple programming questions, including questions about syntax or particularly tricky lines of code. Finally, the *Programming Projects* section contains ideas for extended assignments and suggestions for Java applets.

Pedagogical Features

- Margin notes are used to highlight important topics.
- The *Objects of the Game* section lists important terms along with definitions and page references.
- The *Common Errors* section at the end of each chapter provides a list of common errors.
- References for further reading are provided at the end of most chapters.

Code Availability

The code in the text is fully functional and has been tested on Sun's JDK 1.1. It is available via anonymous ftp at `aw.com` and through the World Wide Web at `http://www.aw.com/cseng/titles/0-201-54991-3/` (follow the links from there). The *On the Internet* section at the end of each chapter lists the filenames for the chapter's code.

Instructor's Resource Guide

An Instructor's Guide is available that illustrates several approaches to the material. It includes samples of test questions, assignments, and syllabi. Answers to select exercises are also provided. Instructors should contact their Addison-Wesley local sales representative for information on the Guide's availability.

Acknowledgments

Many, many people have helped me in the preparation of this book. Many have already been acknowledged in the previous work, *Algorithms, Data Structures, and Problem Solving with C++*, on which this book is based. Others, too numerous to list, have sent e-mail messages and pointed out errors or inconsistencies in explanations that I have tried to fix in this version.

For this book, I would like to thank all of the folks at Addison-Wesley: my Editor, Susan Hartman, and Associate Editor, Katherine Harutunian, helped me make some difficult decisions regarding the organization of the Java material and were very helpful in bringing this book to fruition. My copyeditor and proofreaders suggested numerous rewrites that improved the text. They are Laura K. Michaels, Phyllis Coyne, and Sarah Hallet-Corey. Syndi Hirsch did a lovely cover design. As always, Tom Ziolkowski has done a superb job in the marketing department. I would especially like to thank Pat Unubun, my production editor, for her outstanding effort coordinating the entire project.

I also thank the reviewers, who provided valuable comments, many of which have been incorporated into the text: John Chenoweth, Rob Clark, John Franco, Susanne Hupfer, Steven L. Jenkins, Josephine DeGuzman Mendoza, Viera K. Proulx, and Amr Sabry.

Some of the material in this text is adapted from my textbook *Efficient C Programming: A Practical Approach* (Prentice-Hall, 1995) and is used with permission of the publisher. I have attempted to place end-of-chapter references where appropriate.

My World Wide Web page, `http://www.cs.fiu.edu/~weiss`, will contain updated source code, an errata list, and a link to submit bug reports.

Contents