Data Structures and the Java Collections Framework

William J. Collins

Data Structures and the Java Collections Framework

William J. Collins
Lafayette College

McGraw 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

DATA STRUCTURES AND THE JAVA COLLECTIONS FRAMEWORK

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

ISBN 0–07–236964–7
ISBN 0–07–112184–6 (ISE)

General manager: *Thomas E. Casson*
Publisher: *Elizabeth A. Jones*
Developmental editor: *Emily J. Gray*
Executive marketing manager: *John Wannemacher*
Project manager: *Jane E. Matthews*
Senior production supervisor: *Sandy Ludovissy*
Coordinator of freelance design: *David W. Hash*
Cover designer: *Rokusek Design*
Cover image: © *PhotoDisc, Inc.*
Senior supplement producer: *Stacy A. Patch*
Media technology senior producer: *Phillip Meek*
Compositor: *Lachina Publishing Services*
Typeface: *10/12 Times Roman*
Printer: *Quebecor World Fairfield, PA*

Library of Congress Cataloging-in-Publication Data

Collins, William J. (William Joseph)
Data structures and the Java collections framework / William J. Collins—1st ed.
p. cm.
Includes index.
ISBN 0–07–236964–7
1. Java (Computer program language). 2. Data structures (Computer science). I. Title.

QA76.73.J38 C657 2002
005.7'3—dc21 2001030694
CIP

INTERNATIONAL EDITION ISBN 0–07–112184–6

To Karen, my wife of 35 years, for giving me 20 of the happiest years of my life.

W.J.C.

PREFACE

This book is intended for an object-oriented course in data structures and algorithms. The implementation language is Java, and it is assumed that students have taken an introductory course in which that language was used. That course should have covered the fundamental statements and data types, as well as arrays and the basics of file processing.

THE JAVA COLLECTIONS FRAMEWORK

One of the distinctive features of this text is its emphasis on the Java Collections Framework, part of the java.util package. Basically, the framework is a hierarchy with interfaces at each level except the lowest, and collection classes that implement those interfaces at the lowest level. The collection classes implement most of the data structures studied in a second computer science course, such as a resizable array class, a linked-list class, a balanced binary-search-tree class, and a hash-set class.

There are several advantages to using the Java Collections Framework. First, students will be working with code that has been extensively tested; they need not depend on modules created by the instructor or textbook author. Second, students will have the opportunity to study professionals' code, which is substantially more efficient—and succinct—than what they have seen before. Third, the framework is available for later courses in the curriculum, and beyond!

OTHER IMPLEMENTATIONS CONSIDERED

As important as the Java Collections Framework is, it cannot be the exclusive focus of such a fundamental course in data structures and algorithms. Approaches that differ from those in the Java Collections Framework also deserve consideration. For example, the HashSet and HashMap classes utilize chaining, so there is a separate section on open addressing, and a discussion of the trade-offs of one design over the other. Also, there is coverage of data structures (such as graphs) and algorithms (such as the heap sort) that are not yet included in the Java Collections Framework.

This text also satisfies another essential need of a data structures and algorithms course: the need for students to practice developing their own data structures. There are programming projects in which data structures are either created "from the ground up" or extended from examples in the chapters. And there are other projects to develop or extend applications that *use* the Java Collections Framework.

GRAPHICAL USER INTERFACE

Instead of console input-output, we employ a simple graphical user interface (GUI) with a single input line and any number of output lines. A brief outline of this GUI is given in Chapter 1, and a more extensive description is provided in Appendix 2. Two immediate consequences of using this GUI are that input loops are no longer applicable, and that the output is scrollable. But the main significance is that students gain a better understanding of event-driven programming: the real-world environment.

PEDAGOGICAL FEATURES

This text offers several features that may improve the teaching environment for instructors and the learning environment for students. Each chapter starts with a list of objectives, and concludes with at least one major programming assignment. Each data structure is carefully described, with a precondition and postcondition for each method. In addition, most of the methods include examples of how to call the method, and the results of that call.

The implementation details, especially of the Java Collections Framework classes, are carefully investigated in the text and reinforced in a suite of 24 labs. The organization of these labs is described later in this preface. Each chapter has a variety of exercises, and the answers to all of the exercises are available to the instructor.

SUPPORT MATERIAL

The website for all of the support material is www.mhhe.com/collins, and it has links to the following information for students:

- An overview of the labs and how to access them
- The source code for all projects developed in the text
- Applets for projects that have a strong visual component

Additionally, instructors can obtain the following from the website:

- Instructors' options with regard to the labs
- PowerPoint slides for each chapter (approximately 1500 slides)
- Answers to every exercise

SYNOPSES OF THE CHAPTERS

Chapter 1 presents those features of Java that serve as the foundation for subsequent chapters. Much of the material reflects an object orientation: inheritance, polymorphism, and exception handling. There are lab experiments to review classes, as well as on inheritance and exception handling. See the "Organization of the Labs" section in this preface for more information on labs.

Chapter 2 introduces abstract classes and interfaces, and there is a lab for both of these topics. Of special interest is the Collection interface, the root of many classes in the Java Collections Framework. As a simple implementation of the Collection interface, a primitive version of a singly linked list class is created. This LinkedCollection class serves mainly to provide a backdrop for several key features of the Java Collections Framework, such as iterators and embedded classes. A lab experiment on iterators helps to solidify a student's understanding of this vital concept.

Chapter 3, an introduction to software engineering, outlines the four stages of the software-development life cycle: analysis, design, implementation, and maintenance. The Unified Modeling Language is introduced as a design tool to depict inheritance, composition, and aggregation. Big-O notation, which pervades subsequent chapters, allows environment-independent estimates of the time requirements for methods. Both run-time validation and timing are discussed, and for each of those topics there is a follow-up lab.

Chapter 4, on recursion, represents a temporary shift in emphasis from data structures to algorithms. Backtracking is introduced, not only as a general technique for problem solving, but as another illustration of creating polymorphic references through interfaces. And the same BackTrack class is used for searching a maze; placing eight queens on a chess board, where none is under attack by another queen; and illustrating that a knight can traverse every square in a chess board without landing on any square more than once. Other applications of recursion, such as for the Towers of Hanoi game, further highlight the elegance of recursion, especially when compared to the corresponding iterative methods. This elegance is further illustrated in labs on Fibonacci numbers, the binary search, and generating permutations. Recursion is also encountered in later chapters, notably in the Java Collections Framework versions of the quick sort and merge sort. Moreover, recursion is an indispensable—even if seldom used—tool for every computing professional.

In Chapter 5, we begin our study of the Java Collections Framework with the ArrayList data structure and class. An ArrayList structure is a smart array: automatically resizable, and with methods to handle insertions and deletions at any index. The design starts with the method description—precondition, postcondition, and method heading—of the most widely used methods in the ArrayList class. There follows an outline of the implementation of the class, and further details are available in a lab. The application of the ArrayList class, high-precision arithmetic, is essential for public-key cryptography. This application is extended in a lab and in a programming project. There is another programming project to develop a Deque class.

Chapter 6 presents the LinkedList data structure and class, characterized by linear-time methods for inserting, removing, or retrieving at an arbitrary position. This property makes a compelling case for list iterators: objects that traverse a LinkedList object and have constant-time methods for inserting, removing, or retrieving at the "current" position. The Java Collections Framework's design is doubly linked and circular, but other approaches are also considered. The application is a small line-editor, for which list iterators are well suited. This application is extended in a programming project.

Queues and stacks are the subjects of Chapter 7. The Queue class is not currently included in the Java Collections Framework, but is easily and efficiently implemented with a LinkedList field. A contiguous implementation is also presented. The specific application of calculating the average waiting time at a car wash falls into the general category of *computer simulation.* The Stack class implementation, in the package java.util, predates the Java Collections Framework. There are two applications of the Stack class: the implementation of recursion, and the conversion from infix to postfix notation. This latter application is expanded in a lab, and forms the basis for a project on evaluating a condition.

Chapter 8 focuses on binary trees in general, and binary search trees in particular. The essential features of binary trees are presented; these are important for understanding later material on AVL trees, red-black trees, heaps, and decision trees. The study of binary search trees sets the stage for the subject of chapter 9, balanced binary search trees. In fact, the binary search tree class is a monochromatic version of the Java Collections Framework's implementation of red-black trees.

In Chapter 9, we look at balanced binary search trees, specifically, AVL trees and red-black trees. Rotations are introduced as the mechanism by which rebalancing is accomplished. With the help of Fibonacci trees, we establish that the height of an AVL tree is always logarithmic in the number of elements in the tree. Red-black trees are similarly well behaved. The AVLTree class is implemented, except for the remove method (Project 9.1).

Red-black trees are implemented in the Java Collections Framework in the TreeMap and TreeSet classes, the foci of Chapter 10. In a Map object, each element has a unique key part and also a value part. A TreeMap object is stored in a red-black tree, ordered by the elements' keys. There are labs to guide students through the details of restructuring after an insertion or removal. The application consists of searching a thesaurus for synonyms. A TreeSet object is implemented as a TreeMap object in which each element has the same dummy value part. The application of the TreeSet class is a simple spell-checker. There are project assignments to determine the frequency of each word in a text file, and to build a concordance.

Chapter 11 introduces the PriorityQueue interface, which is not yet part of the Java Collections Framework. A heap-based implementation allows insertions in constant average time, and removal of the highest-priority element in logarithmic worst time. The application is in the area of data compression, specifically, Huffman encodings: given a text file, generate a minimal, prefix-free encoding. The project assignment is to convert the encoding back to the original text file. The lab experiment incorporates fairness into a priority queue, so that ties for highest-priority element are always resolved in favor of the element that was on the priority queue for the longest time.

Sorting is the topic of Chapter 12. Estimates of the maximum lower bounds for comparison-based sorts are developed. The Java Collections Framework provides two sort methods: the quick sort for arrays of primitive types, and the merge sort for arrays of objects and for implementations of the List interface. Two other important sort algorithms, the tree sort and the heap sort, are also included. The chapter's lab experiment compares all of these sort algorithms on randomly generated integers. The project assignment is to sort a file of names and social security numbers.

Chapter 13 starts with a review of sequential and binary searching, and then investigates hashing. The Java Collections Framework has a HashMap class for elements that consist of key-value pairs. The HashSet class is backed by the HashMap class; that is, a HashSet object is viewed as a HashMap object in which all the elements have the same dummy value parts. Basically, the average time for insertion, removal, and searching is constant! This average speed is further explored in a lab that compares HashMap objects to TreeMap objects. There is also a comparison of chained hashing (the basis for the HashMap class) and open-address hashing. This comparison is further explored in a programming project.

The most general data structures—graphs, trees, and networks—are presented in Chapter 14. There are, initially, outlines of the essential algorithms: breadth-first traversal, depth-first traversal, connectedness, finding a minimal spanning tree, and finding a shortest path between two vertices. The only class developed is the (directed) Network class, with an adjacency-list implementation. Other classes, such as Undirected Graph and UndirectedNetwork, can be straightforwardly defined as subclasses of the Network class. The traveling salesperson problem is investigated in a lab, and there is a programming project to complete an adjacency-matrix version of the Network class. Another backtracking application is presented, with the same BackTrack class that was introduced in Chapter 4.

With each chapter, there is an associated web page that includes all programs developed in the chapter, and applets, where appropriate, to animate the concepts presented.

APPENDICES

Appendix 1 contains the background that will allow students to comprehend the mathematical aspects of the chapters. Summation notation and the rudimentary properties of logarithms are essential, and the material on mathematical induction will lead to a deeper appreciation of the analysis of binary trees and open-address hashing.

Appendix 2 is a brief tutorial on the GUI and GUIListener classes. These classes support the event model for input and output in all the programs in the text and labs. Understanding the event model and the role of separate threads is quite a bit more difficult than understanding console input and output. But virtually every application program is event driven, so the time spent on these topics is an investment in the future.

Appendix 3 presents a user's view of the Java Collections Framework. For each method, a method description is provided: precondition, postcondition, and method heading. The relationships between the six classes and the four interfaces are as follows:

The ArrayList class **implements** the List interface, which **extends** the Collection interface.

The LinkedList class **implements** the List interface, which **extends** the Collection interface.

The TreeMap class **implements** the Map interface.

The TreeSet class **implements** the Set interface, which **extends** the Collection interface.

The HashMap class **implements** the Map interface.

The HashSet class **implements** the Set interface, which **extends** the Collection interface.

ORGANIZATION OF THE LABS

There are 24 website labs associated with this text. For both students and instructors, the Uniform Resource Locator (URL) is www.mhhe.com/collins. The labs do not contain essential material, but provide reinforcement of the text material. For example, after the ArrayList and LinkedList classes have been investigated, there is a lab to perform some timing experiments on those two classes.

The labs are self-contained, so the instructor has considerable flexibility in assigning the labs:

1. They can be assigned as closed labs.
2. They can be assigned as open labs.
3. They can be assigned as ungraded homework.

In addition to the obvious benefit of promoting active learning, these labs also encourage use of the scientific method. Basically, each lab is set up as an experiment. Students *observe* some phenomenon, such as the organization of the Java Collections Framework's LinkedList class. They then formulate and submit a *hypothesis*—with their own code—about the phenomenon. After *testing* and, perhaps, revising their hypothesis, they submit the *conclusions* they drew from the experiment.

There are more labs related to earlier chapters than to later ones. This allows students to start working right from the beginning of the course, even before programming projects can be assigned.

ACKNOWLEDGMENTS

Chun Wai Liew developed the graphical user interface employed throughout the book and converted the Java classes to utilize Swing components. Joshua Bloch of Sun Microsystems provided valuable insights into the Java Collections Framework. And I am grateful to Sun Microsystems for permission to include code (mostly Joshua's!) from the Java Collections Framework.

The following reviewers made many helpful suggestions:

Joseph M. Clifton, *University of Wisconsin, Platteville*
Homer C. Gerber, *University of Central Florida*
James K. Huggins, *Kettering University*
Dean Kelley, *Minnesota State University, Mankato*
Teresa Leyk, *Texas A&M University*
Mitchell L. Neilsen, *Kansas State University*
Thaddeus F. Pawlicki, *University of Rochester*
Gregory J. E. Rawlins, *Indiana University*
Munindar P. Singh, *North Carolina State University*
Sebastian Thrun, *Carnegie Mellon University*
Kenneth R. Vollmar, *Southwest Missouri State University*
Ming Wang, *Embry-Riddle Aeronautical University*
Zhi-Li Zhang, *University of Minnesota*

I am thankful for the encouragement and support from McGraw-Hill. Emily Gray and Betsy Jones were cordially persistent in steering me away from unworkable ideas. Jane Matthews kept the production on schedule.

Several students from Lafayette College made important contributions. Eric Panchenko created all of the applets and many of the driver programs; Yi Sun constructed the initial template for the labs; Xenia Taoubina wrote many of the method descriptions and examples in Appendix 3. Finally, I am indebted to all of the students at Lafayette College who participated in the class testing of the book and endured earlier versions of the labs.

Bill Collins

BRIEF CONTENTS

CONTENTS