



Data Structures and the Java Collections Framework

```
private void ...  
if (month == 1) ...  
return 30;  
else if (month == 2) ...  
month == 8 ...  
return 31; // January ...  
else if (year % 4 == 0) ...  
return ...  
return 29;
```

William J. Collins

Data Structures and the Java Collections Framework

William J. Collins
Lafayette College



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

Published by McGraw-Hill, a business unit of The McGraw-Hill Companies, Inc., 1221 Avenue of the Americas, New York, NY 10020. Copyright © 2002 by The McGraw-Hill Companies, Inc. All rights reserved. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written consent of The McGraw-Hill Companies, Inc., including, but not limited to, in any network or other electronic storage or transmission, or broadcast for distance learning.

Some ancillaries, including electronic and print components, may not be available to customers outside the United States.

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 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
005.73—dc21 2001030694
CIP

INTERNATIONAL EDITION ISBN 0-07-112184-6

Copyright © 2002. Exclusive rights by The McGraw-Hill Companies, Inc., for manufacture and export. This book cannot be re-exported from the country to which it is sold by McGraw-Hill. The International Edition is not available in North America.

www.mhhe.com

**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

SYNOPSIS 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 `LinkedList` 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 `UndirectedGraph` 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

Preface xiv

CHAPTER 1	
Important Features of Java	1
CHAPTER 2	
Interfaces and Collection Classes	39
CHAPTER 3	
Introduction to Software Engineering	65
CHAPTER 4	
Recursion	93
CHAPTER 5	
Array Lists	149
CHAPTER 6	
Linked Lists	185
CHAPTER 7	
Queues and Stacks	233
CHAPTER 8	
Binary Trees and Binary Search Trees	277
CHAPTER 9	
Balanced Binary Search Trees	323
CHAPTER 10	
Tree Maps and Tree Sets	361
CHAPTER 11	
Priority Queues	415
CHAPTER 12	
Sorting	453

CHAPTER 13	
Searching and the Hash Classes	495
<hr/>	
CHAPTER 14	
Graphs, Trees, and Networks	539
<hr/>	
APPENDIX 1	
Mathematical Background	593
<hr/>	
APPENDIX 2	
The GUI and GUIListener Classes	607
<hr/>	
APPENDIX 3	
The Java Collections Framework	619
<hr/>	

Bibliography 709

Index 711

CONTENTS

Preface xiv

CHAPTER 1 Important Features of Java 1

Chapter Objectives 1

1.1 Classes 2

- 1.1.1 Method Descriptions 2
- 1.1.2 Data Abstraction 5
- 1.1.3 An Employee Class 7
- 1.1.4 Local Variables and Fields 10
- 1.1.5 Constructors 10
- 1.1.6 Instance Variables and Static Variables 11
- 1.1.7 Visibility Modifiers 12
- 1.1.8 Graphical User Interfaces 13
- 1.1.9 The Company Class 14
- Lab 1: The CompanyMain Project 14**
- 1.1.10 Inheritance 14
- 1.1.11 The protected Visibility Modifier 16
- 1.1.12 Inheritance and Constructors 18
- Lab 2: The SalariedEmployee Class 22**
- 1.1.13 Polymorphism 23
- 1.1.14 Information Hiding 27
- 1.1.15 Exception Handling 27
- 1.1.16 Propagating Exceptions 30
- Lab 3: An Example of Exception Handling 31**

Summary 33

Exercises 33

Programming Project 1.1: Developing and Using a Sequence Class 37

CHAPTER 2 Interfaces and Collection Classes 39

Chapter Objectives 39

2.1 Abstract Methods and Abstract Classes 40

Lab 4: A Class for Regular Polygons 41

2.2 Interfaces 41

2.3 Arrays 45

2.4 Collection Classes 46

2.5 Storage Structures for Collection Classes 48

Lab 5: The ArrayList Class's Implementation of the Collection Interface 49

2.5.1 Linked Structures 49

2.5.2 The LinkedList Class 49

2.5.3 Fields and Method Definitions in the LinkedList Class 51

2.5.4 Iterators 55

Lab 6: Expanding the LinkedList Class 59

2.5.5 Data Structures and the Java Collections Framework 59

Summary 59

Exercises 60

Programming Project 2.1: Expanding the LinkedList Class 62

CHAPTER 3 Introduction to Software Engineering 65

Chapter Objectives 65

3.1 The Software Development Life Cycle 66

3.2 Problem Analysis 66

3.2.1 System Tests 68

3.3 Program Design 69

3.3.1 Method Descriptions and Fields 69

3.3.2 Dependency Diagrams 70

3.4 Program Implementation 73

3.4.1 Method Validation 73

Lab 7: Drivers 74

3.4.2 Is Correctness Feasible? 74

3.4.3 Estimating the Efficiency of Methods 75

3.4.4 Big-O Notation 76

- 3.4.5 *Getting Big-O Estimates Quickly* 78
- 3.4.6 *Trade-Offs* 82
- 3.4.7 *Run-Time Analysis* 84
- 3.4.8 *Overview of the Random Class* 85
- Lab 8: Randomness and Timing** 86

3.5 Program Maintenance 86

Summary 87

Exercises 88

Programming Project 3.1: Further Expansion of the
LinkedCollection Class 91

CHAPTER 4 Recursion 93

Chapter Objectives 93

4.1 Introduction 94

4.2 Factorials 94

4.2.1 *Execution Frames* 96

4.3 Decimal to Binary 98

Lab 9: Fibonacci Numbers 101

4.4 Towers of Hanoi 102

4.4.1 *A Recurrence Relation* 110

4.5 Backtracking 111

4.5.1 *An A-maze-ing Application* 114

4.6 Binary Search 120

Lab 10: Iterative Binary Search 131

Lab 11: Generating Permutations 131

4.7 Indirect Recursion 131

4.8 The Cost of Recursion 132

Summary 133

Exercises 134

Programming Project 4.1: Iterative Version of
Towers of Hanoi 142

Programming Project 4.2: Eight Queens 144

Programming Project 4.3: A Knight's Tour 146

CHAPTER 5 Array Lists 149

Chapter Objectives 149

5.1 The List Interface 150

5.2 The ArrayList Class 151

5.2.1 *Method Descriptions for the ArrayList
Class* 152

5.2.2 *ArrayList Class Heading* 158

5.2.3 *Fields in the ArrayList Class* 159

5.2.4 *ArrayList Objects Are Serializable* 160

5.2.5 *ArrayList Objects Are Cloneable* 161

5.3 The ArrayList Implementation 162

5.3.1 *Definition of the add Method* 163

5.3.2 *Amortized Time* 165

5.3.3 *The clone Method and the Copy
Constructor* 166

5.3.4 *Fail-Fast Iterators* 168

Lab 12: More Details on the ArrayList Class 169

5.4 Application: High-Precision Arithmetic 169

5.4.1 *Design of the VeryLongInt Class* 170

5.4.2 *Implementation of the VeryLongInt
Class* 171

Lab 13: Extending the VeryLongInt Class 175

5.5 The Vector Class 175

Summary 175

Exercises 175

Programming Project 5.1: Extending the
VeryLongInt Class 179

Programming Project 5.2: The Deque Class 180

CHAPTER 6 Linked Lists 185

Chapter Objectives 185

6.1 The LinkedList Class 186

6.1.1 *The LinkedList Class versus the
ArrayList Class* 187

6.1.2 *LinkedList Iterators* 190

6.1.3 *Fields and Implementation of the LinkedList
Class* 195

6.1.4 *Fields and Implementation of Listltr
Class* 203

**Lab 14: More Implementation Details of the
Listltr Class** 206

**Lab 15: Timing the ArrayList and
LinkedList Classes** 207

6.1.5 *Alternative Designs and Implementations
of the LinkedList Class* 207

6.1.6	<i>Circular Linked Lists</i>	210
6.2	Application: A Line Editor	211
6.2.1	<i>Design of the Editor Class</i>	215
6.2.2	<i>Implementation of the Editor Class</i>	217
6.2.3	<i>Big-O Analysis of the Editor Class Methods</i>	221
6.2.4	<i>The EditorDriver Class</i>	221
	Summary	223
	Exercises	224
	Programming Project 6.1: Extending the Line Editor	226
	Programming Project 6.2: Alternative Design and Implementation of the LinkedList Class	231

CHAPTER 7

Queues and Stacks 233

	Chapter Objectives	233
7.1	Queues	234
7.1.1	<i>Design and Implementation of the Queue Class</i>	235
7.1.2	<i>Alternative Designs and Implementation of the Queue Class</i>	237
7.2	Computer Simulation	242
7.3	Application: A Simulated Car Wash	244
7.3.1	<i>Design of the CarWash Class</i>	245
7.3.2	<i>Implementation of the CarWash Class</i>	246
7.3.3	<i>Analysis of the CarWash Methods</i>	249
7.3.4	<i>Randomizing the Arrival Times</i>	250
	Lab 16: Randomizing the Arrival Times	251
7.4	Stacks	251
7.4.1	<i>Design and Implementation of the Stack Class</i>	252
7.4.2	<i>The Stack Class in the Java Collections Framework</i>	253
7.4.3	<i>Alternative Designs and Implementations of the Stack Class</i>	253
7.5	Application: How Compilers Implement Recursion	254
7.6	Application: Converting From Infix to Postfix	257
7.6.1	<i>Postfix Notation</i>	259
7.6.1	<i>Transition Matrix</i>	260

7.6.3	<i>Tokens</i>	261
	Lab 17: Converting from Infix to Postfix	263
7.6.4	<i>Prefix Notation</i>	263

Summary	267
Exercises	267
Programming Project 7.1: Extending Speedo's Car Wash	270
Programming Project 7.2: Run-Time Evaluation of a Condition	272
Programming Project 7.3: An Iterative Version of Maze-Search	276

CHAPTER 8

Binary Trees and Binary Search Trees 277

	Chapter Objectives	277
8.1	Definition and Properties of Binary Trees	278
8.1.1	<i>The Binary Tree Theorem</i>	284
8.1.2	<i>External Path Length</i>	287
8.1.3	<i>Traversals of a Binary Tree</i>	288
8.2	Binary Search Trees	294
8.2.1	<i>The BinSearchTree Class</i>	295
8.2.2	<i>Fields and Embedded Classes in the BinSearchTree Class</i>	298
8.2.3	<i>Implementation of the BinSearchTree Class</i>	298
8.2.4	<i>The remove Method</i>	304
8.2.5	<i>The Treeliterator Class</i>	313
	Lab 18: A Run-Time Estimate of the Average Height of a BinSearchTree Object	315
	Summary	316
	Exercises	317
	Programming Project 8.1: An Alternative Design and Implementation of the Binary-Search-Tree Data Structure	321

CHAPTER 9

Balanced Binary Search Trees 323

	Chapter Objectives	323
9.1	A Problem with Binary Search Trees	324

- 9.2** Rotations 324
- 9.3** AVL Trees 329
 - 9.3.1 *The Height of an AVL Tree* 330
 - 9.3.2 *The AVLTree Class* 332
 - 9.3.3 *The fixAfterInsertion Method* 335
 - 9.3.4 *Correctness of the add Method* 345
- 9.4** Red-Black Trees 348
 - 9.4.1 *The Height of a Red-Black Tree* 351
- Summary 355
- Exercises 356
- Programming Project 9.1: Defining the remove Method in the AVLTree Class 360

CHAPTER 10

Tree Maps and Tree Sets 361

Chapter Objectives 361

- 10.1** The TreeMap Class 362
 - 10.1.1 *Method Descriptions of the TreeMap Class* 362
 - 10.1.2 *The Fields in the TreeMap Class* 365
 - 10.1.3 *The Comparator and Comparable Classes* 365
 - 10.1.4 *The Entry Class* 367
 - 10.1.5 *Implementation of the TreeMap Class* 367
 - 10.1.6 *The fixAfterInsertion Method* 369
 - 10.1.7 *Three Cases of Insertion* 371
 - Lab 19:** *A Red-Black Tree Insertion with All Three Cases* 375
 - 10.1.8 *More TreeMap Methods* 375
 - 10.1.9 *The fixAfterDeletion Method* 380
 - Lab 20:** *A Red-Black Tree Removal with All Four Cases* 388
 - 10.1.10 *The entrySet Method* 389
- 10.2** Application: TreeMap Objects: A Simple Thesaurus 389
 - 10.2.1 *Design and Implementation of the Thesaurus Class* 390
 - 10.2.2 *Design and Implementation of the ThesaurusDriver Class* 392

- 10.3** The TreeSet Class 395
 - 10.3.1 *Design and Implementation of the TreeSet Class* 398
- 10.4** Application: A Simple Spell-Checker 399
 - 10.4.1 *Design and Implementation of the SpellChecker Class* 400
 - 10.4.2 *Design and Implementation of the SpellCheckerDriver Class* 402
- Summary 405
- Exercises 405
- Programming Project 10.1: Enhancing the SpellChecker Project 408
- Programming Project 10.2: Determining Word Frequencies 410
- Programming Project 10.3: Building a Concordance 412

CHAPTER 11

Priority Queues 415

Chapter Objectives 415

- 11.1** Introduction 416
- 11.2** Definition of the PriorityQueue Interface 417
- 11.3** Implementations of the PriorityQueue Interface 417
 - 11.3.1 *The Heap Class* 419
 - 11.3.2 *Fields in the Heap Class* 424
 - 11.3.3 *Implementation of the Heap Class* 424
 - 11.3.4 *The percolateUp Method* 425
 - 11.3.5 *The percolateDown Method* 430
 - Lab 21:** *Incorporating Fairness in Heaps* 432
- 11.4** Application: Huffman Codes 432
 - 11.4.1 *Huffman Trees* 435
 - 11.4.2 *Greedy Algorithms* 438
 - 11.4.3 *The Huffman Class* 439
- Summary 446
- Exercises 447
- Programming Project 11.1 Decoding a Huffman-Encoded Message 450