

LEWIS | CHASE

*java*TM

Third Edition

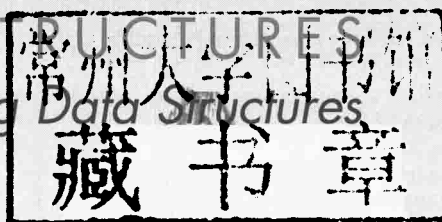
SOFTWARE STRUCTURES

Designing and Using Data Structures





*java*TM Third Edition
SOFTWARE STRUCTURES
Designing and Using Data Structures



JOHN LEWIS

Virginia Tech

JOSEPH CHASE

Radford University

Addison-Wesley

New York Boston San Francisco

London Toronto Sydney Tokyo Singapore Madrid

Mexico City Munich Paris Cape Town Hong Kong Montreal

Editor-in-Chief
Editorial Assistant
Managing Editor
Production Supervisor
Marketing Manager
Marketing Coordinator
Senior Manufacturing Buyer
Online Product Manager
Art Director
Cover Design
Project Management, Composition, and Illustrations
Project Coordinator, Nesbitt Graphics, Inc.
Project Manager, Nesbitt Graphics, Inc.
Text Design, Nesbitt Graphics, Inc.
Cover Image

Michael Hirsch
Stephanie Sellinger
Jeffrey Holcomb
Heather McNally
Erin Davis
Kathryn Ferranti
Carol Melville
Bethany Tidd
Linda Knowles
Elena Sidorova
Nesbitt Graphics, Inc.
Harry Druding
Kathy Smith
Jerilyn Bockorick, Alisha Webber
Steve Cole/Getty Images

Access the latest information about Addison-Wesley Computer Science titles from our World Wide Web site:
<http://www.pearsonhighered.com/cs>.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial caps or all caps.

The programs and applications presented in this book have been included for their instructional value. They have been tested with care, but are not guaranteed for any particular purpose. The publisher does not offer any warranties or representations, nor does it accept any liabilities with respect to the programs or applications.

Library of Congress Cataloging-in-Publication Data

Lewis, John, 1963-

Java software structures : designing and using data structures / John

Lewis, Joseph Chase. -- 3rd ed.

p. cm.

Includes bibliographical references and index.

ISBN 978-0-13-607858-6 (alk. paper)

1. Java (Computer program language)
 2. Data structures (Computer science)
 3. Computer software--Development. I. Chase, Joseph. II. Title.
- QA76.73.J38L493 2009
005.13'3--dc22

2009000302

Copyright © 2010 Pearson Education, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America.

For information on obtaining permission for use of material in this work, please submit a written request to Pearson Education, Inc., Rights and Contracts Department, 501 Boylston Street, Suite 900, Boston, MA 02116, fax your request to (617)671-3447, or e-mail at <http://www.pearsoned.com/legal/permissions.htm>.

Addison Wesley
is an imprint of

PEARSON

www.pearsonhighered.com

ISBN-13: 978-013-607858-6

ISBN-10: 0-13-607858-3

1 2 3 4 5 6 7 8 10-EB-13 12 11 10 09

*To my wife Sharon and my kids:
Justin, Kayla, Nathan, and Samantha*

-J. L.

*To my loving wife Melissa for her support and encouragement
and to our families, friends, colleagues, and students who have provided
so much support and inspiration through the years.*

-J. C.

Preface

This book is designed to serve as a text for a course on data structures and algorithms. This course is typically referred to as the CS2 course because it is often taken as the second course in a computing curriculum. We have designed this book to embrace the tenets of Computing Curricula 2001 (CC2001).

Pedagogically, this book follows the style and approach of the leading CS1 book *Java Software Solutions: Foundations of Program Design*, by John Lewis and William Loftus. Our book uses many of the highly regarded features of that book, such as the Key Concept boxes and complete code examples. Together, these two books support a solid and consistent approach to either a two-course or three-course introductory sequence for computing students. That said, this book does not assume that students have used *Java Software Solutions* in a previous course.

Material that might be presented in either course (such as recursion or sorting) is presented in this book as well. We also include strong reference material providing an overview of object-oriented concepts and how they are realized in Java.

We understand the crucial role that the data structures and algorithms course plays in a curriculum and we think this book serves the needs of that course well.

The Third Edition

We have made some key modifications in this third edition to enhance its pedagogy. The most important change is a fundamental reorganization of material that is designed to create a cleaner flow of topics. Instead of having an early, large chapter to review object-oriented concepts, we've included that material as an appendix for reference. Then we review concepts as needed and appropriate in the context of the implementation strategies discussed throughout the book and cite the appropriate reference material. This not only links the topics in a timely fashion but also demonstrates the usefulness of particular language constructs.

We've expanded the discussion of Analysis of Algorithms, and given it its own chapter. The discussion, however, stays at an appropriately moderate level. Our strategy is to motivate the concepts involved in the analysis of algorithms, laying a solid foundation, rather than get embroiled in too much formality.

Another key organizational change is that the introduction to collections uses a stack as the primary example. In previous editions of this book we went out of

our way to introduce collections in an abstract way that separated it from the core data structures, using examples such as a bag or set collection. This new approach capitalizes on the fact that a stack is conceptually about as straightforward as it gets. Using it as a first example enhances the understanding of collections as a whole.

The previous edition of the book had several chapters that focused on larger case studies that made use of collections to solve non-trivial problems. While many instructors found these useful, they also seemed to interrupt the flow of coverage of core topics. Therefore we have taken the case study chapters out of the book and put them on the web as supplementary resources. We encourage all instructors to download and use these resources as they see fit.

Finally, for this edition we've reviewed and improved the discussions throughout the book. We've expanded the discussion of graphs and reversed the order of the graphs and hashing chapters to make a cleaner flow. And we've added a chapter that specifically covers sets and maps.

We think these modifications build upon the strong pedagogy established by previous editions and give instructors more opportunity and flexibility to cover topics as they choose.

Our Approach

Books of this type vary greatly in their overall approach. Our approach is founded on a few important principles that we fervently embraced. First, we present the various collections explored in the book in a consistent manner. Second, we emphasize the importance of sound software design techniques. Third, we organized the book to support and reinforce the big picture: the study of data structures and algorithms. Let's examine these principles further.

Consistent Presentation

When exploring a particular type of collection, we carefully address each of the following issues in order:

1. **Concept:** We discuss the collection conceptually, establishing the services it provides (its interface).
2. **Use:** We explore examples that illustrate how the particular nature of the collection, no matter how it's implemented, can be useful when solving problems.
3. **Implementation:** We explore various implementation options for the collection.
4. **Analysis:** We compare and contrast the implementations.

The Java Collections API is included in the discussion as appropriate. If there is support for a particular collection type in the API, we discuss it and its implementation. Thus we embrace the API, but are not completely tied to it. And we are not hesitant to point out its shortcomings.

The analysis is kept at a high level. We establish the concept of Big-Oh notation in Chapter 2 and use it throughout the book, but the analysis is more intuitive than it is mathematical.

Sound Program Design

Throughout the book, we keep sound software engineering practices a high priority. Our design of collection implementations and the programs that use them follow consistent and appropriate standards.

Of primary importance is the separation of a collection's interface from its underlying implementation. The services that a collection provides are always formally defined in a Java interface. The interface name is used as the type designation of the collection whenever appropriate to reinforce the collection as an abstraction.

In addition to practicing solid design principles, we stress them in the discussion throughout the text. We attempt to teach both by example and by continual reinforcement.

Clean Organization

The contents of the book have been carefully organized to minimize distracting tangents and to reinforce the overall purpose of the book. The organization supports the book in its role as a pedagogical exploration of data structures and algorithms as well as its role as a valuable reference.

The book can be divided into numerous parts: Part I consists of the first two chapters and provides an introduction to the concept of a collection and analysis of algorithms. Part II includes the next four chapters, which cover introductory and underlying issues that affect all aspects of data structures and algorithms as well as linear collections (stacks, queues, and lists). Part III covers the concepts of recursion, sorting, and searching. Part IV covers the nonlinear collections (trees, heaps, hashing, and graphs). Each type of collection, with the exception of trees, is covered in its own chapter. Trees are covered in a series of chapters that explore their various aspects and purposes.

Chapter Breakdown

Chapter 1 (Introduction) discusses various aspects of software quality and provides an overview of software development issues. It is designed to establish the

appropriate mindset before embarking on the details of data structure and algorithm design.

Chapter 2 (Analysis of Algorithms) lays the foundation for determining the efficiency of an algorithm and explains the important criteria that allow a developer to compare one algorithm to another in proper ways. Our emphasis in this chapter is understanding the important concepts more than getting mired in heavy math or formality.

Chapter 3 (Collections) establishes the concept of a collection, stressing the need to separate the interface from the implementation. It also conceptually introduces a stack, then explores an array-based implementation of a stack.

Chapter 4 (Linked Structures) discusses the use of references to create linked data structures. It explores the basic issues regarding the management of linked lists, and then defines an alternative implementation of a stack (introduced in Chapter 3) using an underlying linked data structure.

Chapter 5 (Queues) explores the concept and implementation of a first-in, first-out queue. Radix sort is discussed as an example of using queues effectively. The implementation options covered include an underlying linked list as well as both fixed and circular arrays.

Chapter 6 (Lists) covers three types of lists: ordered, unordered, and indexed. These three types of lists are compared and contrasted, with discussion of the operations that they share and those that are unique to each type. Inheritance is used appropriately in the design of the various types of lists, which are implemented using both array-based and linked representations.

Chapter 7 (Recursion) is a general introduction to the concept of recursion and how recursive solutions can be elegant. It explores the implementation details of recursion and discusses the basic idea of analyzing recursive algorithms.

Chapter 8 (Sorting and Searching) discusses the linear and binary search algorithms, as well as the algorithms for several sorts: selection sort, insertion sort, bubble sort, quick sort, and merge sort. Programming issues related to searching and sorting, such as using the Comparable interface as the basis of comparing objects, are stressed in this chapter. Searching and sorting that are based in particular data structures (such as heap sort) are covered in the appropriate chapter later in the book.

Chapter 9 (Trees) provides an overview of trees, establishing key terminology and concepts. It discusses various implementation approaches and uses a binary tree to represent and evaluate an arithmetic expression.

Chapter 10 (Binary Search Trees) builds off of the basic concepts established in Chapter 9 to define a classic binary search tree. A linked implementation of a binary search tree is examined, followed by a discussion of how the balance in the

tree nodes is key to its performance. That leads to exploring AVL and red/black implementations of binary search trees.

Chapter 11 (Priority Queues and Heaps) explores the concept, use, and implementations of heaps and specifically their relationship to priority queues. A heap sort is used as an example of its usefulness as well. Both linked and array-based implementations are explored.

Chapter 12 (Multi-way Search Trees) is a natural extension of the discussion of the previous chapters. The concepts of 2-3 trees, 2-4 trees, and general B-trees are examined and implementation options are discussed.

Chapter 13 (Graphs) explores the concept of undirected and directed graphs and establishes important terminology. It examines several common graph algorithms and discusses implementation options, including adjacency matrices.

Chapter 14 (Hashing) covers the concept of hashing and related issues, such as hash functions and collisions. Various Java Collections API options for hashing are discussed.

Chapter 15 (Sets and Maps) explores these two types of collections and their importance to the Java Collections API.

Appendix A (UML) provides an introduction to the Unified Modeling Language as a reference. UML is the de facto standard notation for representing object-oriented systems.

Appendix B (Object-Oriented Design) is a reference for anyone needing a review of fundamental object-oriented concepts and how they are accomplished in Java. Included are the concepts of abstraction, classes, encapsulation, inheritance, and polymorphism, as well as many related Java language constructs such as interfaces.

Supplements

The following supplements are available to all readers of this book at www.aw.com/cssupport.

- **Source Code** for all programs presented in the book
- **Full case studies** of programs that illustrate concepts from the text, including a Black Jack Game, a Calculator, a Family Tree Program, and a Web Crawler

The following instructor supplements are only available to qualified instructors at Pearson Education's Instructor Resource Center, <http://www.pearsonhighered.com/irc>. Please visit the Web site, contact your local Pearson Education Sales Representative, or send an e-mail to computing@pearson.com, for information about how to access them.

- **Solutions** for selected exercises and programming projects in the book
- **Test Bank**, containing questions that can be used for exams
- **PowerPoint® Slides** for the presentation of the book content

Acknowledgements

First and most importantly we want to thank our students for whom this book is written and without whom it never could have been. Your feedback helps us become better educators and writers. Please continue to keep us on our toes.

We would like to thank all of the reviewers listed below who took the time to share their insight on the content and presentation of the material in this book and its previous editions. Your input was invaluable.

Mary P. Boelk,	Marquette University
Robert Burton,	Brigham Young University
Gerald Cohen,	St. Joseph's College
Robert Cohen,	University of Massachusetts–Boston
Jack Davis,	Radford University
Bob Holloway,	University of Wisconsin–Madison
Nisar Hundewale,	Georgia State University
Chung Lee,	California State Polytechnic University
Mark C. Lewis,	Trinity University
Mark J. Llewellyn,	University of Central Florida
Ronald Marsh,	University of North Dakota
Eli C. Minkoff,	Bates College; University of Maine–Augusta
Ned Okie,	Radford University
Manuel A. Perez-Quinones,	Virginia Tech
Moshe Rosenfeld	University of Washington
Salam Salloum,	California State Polytechnic University–Pomona
Don Slater,	Carnegie Mellon University
Ashish Soni,	University of Southern California
Carola Wenk,	University of Texas–San Antonio

The folks at Addison-Wesley have gone to great lengths to support and develop this book along with us. It is a true team effort. Editor-in-Chief Michael Hirsch and his assistant Stephanie Sellinger have always been there to help. Marketing Manager Erin Davis, her assistant Kathryn Ferranti, and the entire Addison-Wesley sales force work tirelessly to make sure that instructors understand the goals and benefits of the book. Heather McNally flawlessly handled the production of the book, and Elena Sidorova is to be credited for the wonderful cover design. They are supported by Kathy Smith and Harry Druding at Nesbitt Graphics. Carol Melville always finds a way to get us time on press so

that our book makes it into your hands in time to use it in class. Thank you all very much for all your hard work and dedication to this book.

We'd be remiss if we didn't acknowledge the wonderful contributions of the ACM Special Interest Group on Computer Science Education. Its publications and conferences are crucial to anyone who takes the pedagogy of computing seriously. If you're not part of this group, you're missing out.

Finally, we want to thank our families, who support and encourage us in whatever projects we find ourselves diving into. Ultimately, you are the reason we do what we do.

Contents

Preface

Chapter 1 Introduction 1

1.1	Software Quality	2
	Correctness	3
	Reliability	3
	Robustness	4
	Usability	4
	Maintainability	5
	Reusability	5
	Portability	6
	Efficiency	6
	Quality Issues	6
1.2	Data Structures	7
	A Physical Example	7
	Containers as Objects	10

Chapter 2 Analysis of Algorithms 13

2.1	Algorithm Efficiency	14
2.2	Growth Functions and Big-OH Notation	15
2.3	Comparing Growth Functions	17
2.4	Determining Time Complexity	19
	Analyzing Loop Execution	19
	Nested Loops	20
	Method Calls	21

Chapter 3 Collections 27

3.1	Introduction to Collections	28
	Abstract Data Types	29
	The Java Collections API	31

3.2	A Stack Collection	31
3.3	Crucial OO Concepts	33
	Inheritance	34
	Class Hierarchies	36
	The Object Class	37
	Polymorphism	38
	References and Class Hierarchies	38
	Generics	40
3.4	A Stack ADT	41
	Interfaces	41
3.5	Using Stacks: Evaluating Postfix Expressions	44
3.6	Exceptions	51
	Exception Messages	52
	The try Statement	53
	Exception Propagation	54
3.7	Implementing a Stack: With Arrays	55
	Managing Capacity	56
3.8	The ArrayStack Class	57
	The Constructors	58
	The push Operation	59
	The pop Operation	61
	The peek Operation	62
	Other Operations	63

Chapter 4 Linked Structures **71**

4.1	References as Links	72
4.2	Managing Linked Lists	74
	Accessing Elements	74
	Inserting Nodes	75
	Deleting Nodes	76
	Sentinel Nodes	77
4.3	Elements Without Links	78
	Doubly Linked Lists	78
4.4	Implementing a Stack: With Links	79
	The LinkedStack Class	79

	The push Operation	83
	The pop Operation	85
	Other Operations	86
4.5	Using Stacks: Traversing a Maze	86
4.6	Implementing Stacks:	
	The <code>java.util.Stack</code> Class	93
	Unique Operations	93
	Inheritance and Implementation	94

Chapter 5 Queues 99

5.1	A Queue ADT	100
5.2	Using Queues: Code Keys	103
5.3	Using Queues: Ticket Counter Simulation	107
5.4	Implementing Queues: With Links	112
	The enqueue Operation	114
	The dequeue Operation	115
	Other Operations	117
5.5	Implementing Queues: With Arrays	117
	The enqueue Operation	123
	The dequeue Operation	124
	Other Operations	125

Chapter 6 Lists 131

6.1	A List ADT	132
	Iterators	134
	Adding Elements to a List	135
	Interfaces and Polymorphism	137
6.2	Using Ordered Lists: Tournament Maker	140
6.3	Using Indexed Lists: The Josephus Problem	150
6.4	Implementing Lists: With Arrays	152
	The remove Operation	155
	The contains Operation	157
	The iterator Operation	158
	The add Operation for an Ordered List	158

	Operations Particular to Unordered Lists	161
	The <code>addAfter</code> Operation for an Unordered List	162
6.5	Implementing Lists: With Links	163
	The <code>remove</code> Operation	163
	Doubly Linked Lists	165
	The <code>iterator</code> Operation	168
6.6	Lists in the Java Collections API	171
	<code>Cloneable</code>	172
	<code>Serializable</code>	172
	<code>RandomAccess</code>	172
	<code>Java.util.Vector</code>	173
	<code>Java.util.ArrayList</code>	173
	<code>Java.util.LinkedList</code>	176
Chapter 7	Recursion	185
7.1	Recursive Thinking	186
	Infinite Recursion	186
	Recursion in Math	187
7.2	Recursive Programming	188
	Recursion versus Iteration	190
	Direct versus Indirect Recursion	191
7.3	Using Recursion	192
	Traversing a Maze	192
	The Towers of Hanoi	197
7.4	Analyzing Recursive Algorithms	201
Chapter 8	Sorting and Searching	209
8.1	Searching	210
	Static Methods	211
	Generic Methods	211
	Linear Search	212
	Binary Search	213
	Comparing Search Algorithms	216
8.2	Sorting	217
	Selection Sort	220
	Insertion Sort	222

	Bubble Sort	224
	Quick Sort	226
	Merge Sort	229
8.3	Radix Sort	231

Chapter 9 Trees 241

9.1	Trees	242
	Tree Classifications	243
9.2	Strategies for Implementing Trees	245
	Computational Strategy for	
	Array Implementation of Trees	245
	Simulated Link Strategy for	
	Array Implementation of Trees	246
	Analysis of Trees	247
9.3	Tree Traversals	248
	Preorder Traversal	248
	Inorder Traversal	249
	Postorder Traversal	249
	Level-Order Traversal	250
9.4	A Binary Tree ADT	251
9.5	Using Binary Trees: Expression Trees	255
9.6	Implementing Binary Trees with Links	262
	The find Method	269
	The iteratorInOrder Method	270
9.7	Implementing Binary Trees with Arrays	271
	The find Method	273
	The iteratorInOrder Method	274

Chapter 10 Binary Search Trees 281

10.1	A Binary Search Tree	282
10.2	Implementing Binary Search Trees:	
	With Links	284
	The addElement Operation	286
	The removeElement Operation	288

	The removeAllOccurrences Operation	291
	The removeMin Operation	292
10.3	Implementing Binary Search Trees: With Arrays	294
	The addElement Operation	295
	The removeElement Operation	296
	The removeAllOccurrences Operation	302
	The removeMin Operation	303
10.4	Using Binary Search Trees: Implementing Ordered Lists	304
	Analysis of the BinarySearchTreeList Implementation	308
10.5	Balanced Binary Search Trees	309
	Right Rotation	310
	Left Rotation	310
	Rightleft Rotation	311
	Leftright Rotation	311
10.6	Implementing Binary Search Trees: AVL Trees	312
	Right Rotation in an AVL Tree	313
	Left Rotation in an AVL Tree	315
	Rightleft Rotation in an AVL Tree	315
	Leftright Rotation in an AVL Tree	315
10.7	Implementing Binary Search Trees: Red/Black Trees	315
	Insertion into a Red/Black Tree	316
	Element Removal from a Red/Black Tree	319
10.8	Implementing Binary Search Trees: The Java Collections API	321
10.9	A Philosophical Quandary	325

Chapter 11 Priority Queues and Heaps 333

11.1	A Heap	334
	The addElement Operation	334
	The removeMin Operation	337
	The findMin Operation	338
11.2	Using Heaps: Priority Queues	339