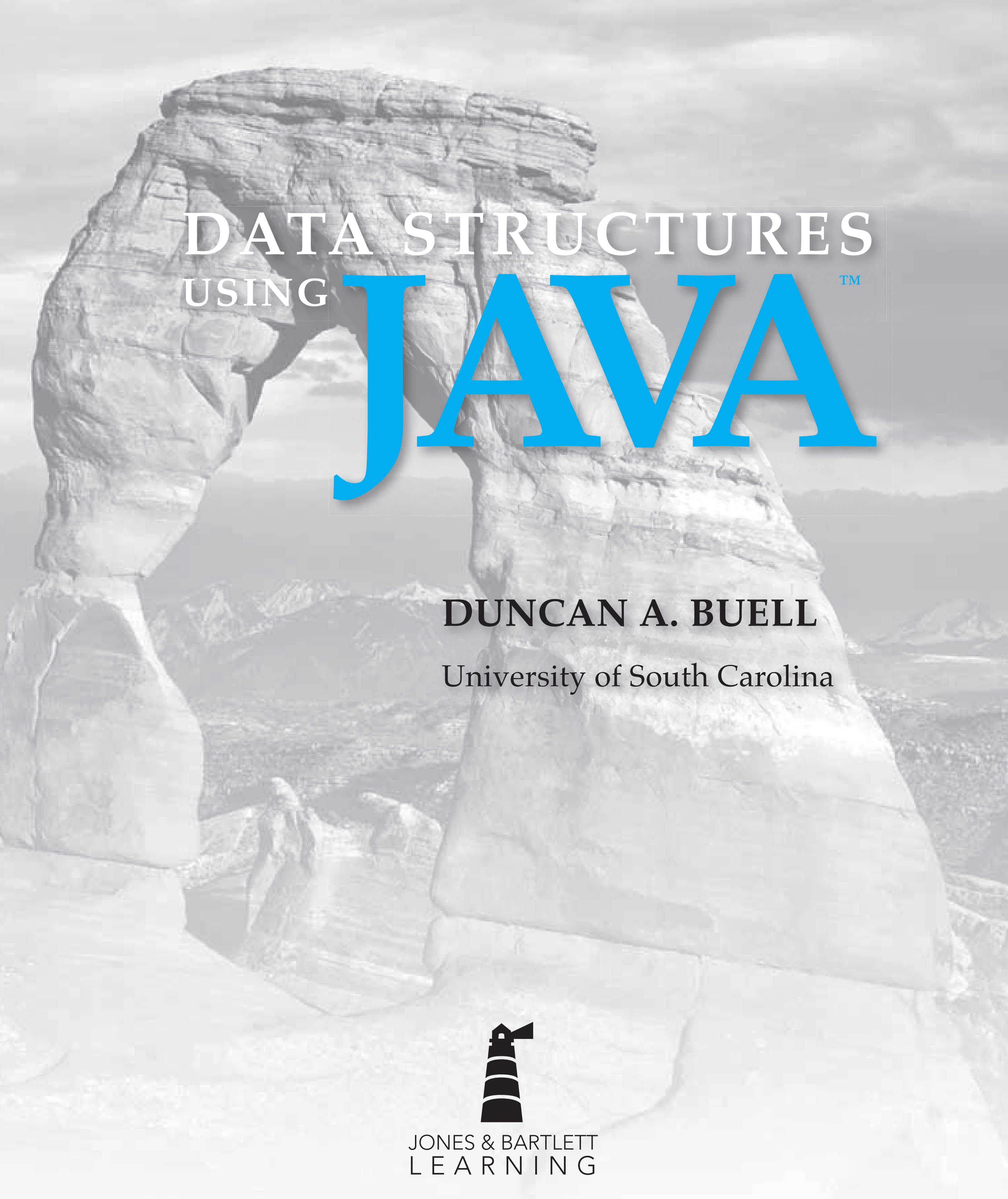

DATA STRUCTURES USING JAVA™

DUNCAN A. BUELL

University of South Carolina

JONES & BARTLETT
LEARNING

World Headquarters
Jones & Bartlett Learning
5 Wall Street
Burlington, MA 01803
978-443-5000
info@jblearning.com
www.jblearning.com

Jones & Bartlett Learning books and products are available through most bookstores and online booksellers. To contact Jones & Bartlett Learning directly, call 800-832-0034, fax 978-443-8000, or visit our website, www.jblearning.com.

Production Credits
Publisher: Cathleen Sether
Senior Acquisitions Editor: Timothy Anderson
Managing Editor: Amy Bloom
Director of Production: Amy Rose
Marketing Manager: Lindsay White
V.P., Manufacturing and Inventory Control: Therese Connell
Composition: Laserwords
Cover Design: Scott Moden
Cover Image: © Mike Norton/ShutterStock, Inc.
Printing and Binding: Malloy, Inc.
Cover Printing: Malloy, Inc.

Library of Congress Cataloging-in-Publication Data
Buell, Duncan A.
Data structures using Java / Duncan A. Buell. – 1st ed.
p. cm.
Includes index.
ISBN-13: 978-1-4496-2807-9 (pbk.)
ISBN-10: 1-4496-2807-9 (pbk.)
1. Data structures (Computer science) 2. Java (Computer program language) I. Title.
QA76.9.D35B84 2013
005.2'762–dc23
2011033497

6048

Printed in the United States of America
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

For Mary Ann

Contents

Preface

This text is intended for a second-semester course in computer science, the course that is often referred to as "CS2" in the ABET/ACM/IEEE standard curriculum. At my own university, this course has the title "Introduction to Algorithmic Design II," and I suspect the titles are similar at many other institutions. What this course is *not intended* to be is "Introduction to Programming II," although there will be a lot of programming discussed in the book. The conceptual material here is on the design and use of algorithms and data structures. The programming vehicle happens to be Java, but it could in fact be almost any language, and this course has no doubt been taught at some college or university in the past in FORTRAN (and probably Fortran), PL/I, Pascal, Basic, C, and C++.

What is important is that *some* programming language be used so that a rigorous use of the concepts is made and that students become facile with the basic ways in which information is manipulated to create an efficient data structure. What is hard to convey to the general public, and sometimes to students, is that the goal is to develop rigorous thinking, using a computer and a programming language so as to enforce the rigor through practice. Although familiarity with the specifics of how to be rigorous in a particular language will come from that practice, this is a secondary goal.

There are two basic methodological premises behind the treatment of the material in this book. The first is that there is no substitute for the school of hard knocks when it comes to understanding why we have some of the more sophisticated features in a modern programming language. A student who writes a program with a global variable that is updated by three different functions, and then changes the updating in two but not all three functions, with a program crash as the inevitable outcome, will learn through that experience why we want to encapsulate data into objects and limit access on what amounts to a "need to know" basis. A student who has never had that experience is much less likely to appreciate fully the concept of objects. The pedagogical problem is that it is difficult, in a fifteen-week semester, in a first-year class, to create assignments that are sufficiently difficult as to create the controlled failure mode that will lead to learning. Much of what we have to say is tantamount to telling students that they ought to eat their vegetables because years later they will appreciate having done so.

In response to this first problem, I have tried to limit the discussion of metaphysical, moral, and theological aspects of Java (or any language) to those concepts that can be firmly grounded in practice in the second semester of a computer science curriculum. Yes, the compiler writers have done some very cool things. No, I don't see that this is necessary in a second-semester course.

The second premise of the book is that necessity is the mother of invention and that sophisticated algorithms and data structures have their natural place in the world. If one is never going to do anything more complicated than a program to balance one's personal checking account, it's not clear that anything more sophisticated than bubblesort or linear search will ever be needed. It is when the naive methods fail us that we implement more clever approaches, and I have tried to emphasize this point. I have always been of the opinion that a program that executes without crashing, even if it doesn't do exactly what is desired, is much more valuable than a program that has all the sophisticated logic programmed in . . . except for a few bugs that crash the program every time. This opinion is what is usually used in the "bottleneckology" of high performance computing—focus on the most time-consuming part of the code and make that part disappear. Since something else will now become the most time-consuming, this is an iterative process until the law of diminishing returns sets in. Similarly, a program that does what is needed, albeit perhaps only on data sets of test size, can be improved piecemeal, method by method, class by class, until it can handle the real job.

For the convenience of the instructor, I have tried to map the contents of this book against the learning objectives of the curricular guidelines from ACM and IEEE in *Computer Science Curriculum 2008*. We have assumed that this is a second-semester text, and thus that the PF/FundamentalConstructs and PF/DataStructures (objectives 1, 3, 4, 5, and 7) areas have already been covered. We also assume some familiarity with DS/FunctionsRelationsAndSets. For the material in Chapter 6 and the discussion of algorithm analysis in general we assume a familiarity with differential calculus. Although we do need differentiation of logarithms and exponentials on a few occasions, we do not need any trigonometry. The crucial part of calculus that is used repeatedly at least in Chapter 6 is L'Hôpital's rule. The numbers in parentheses after the topic area in the following table are our estimates of the coverage of this book compared to the number of hours estimated in *Computer Science Curriculum 2008* to be necessary for minimal coverage of the topic.

Topic Area	Objectives	Our Chapters
DS/GraphsAndTrees (4)	1–4	9, 10, 12, 13
PF/AlgorithmicProblemSolving (6)	7–11	throughout
PF/DataStructures (7)	3–9	throughout
PF/Recursion (4)	1–6	Chapter 8
PF/ObjectOriented (7)	1–4	throughout
AL/AlgorithmsAndComplexity (2)	1–3	Chapter 6
AL/AlgorithmicStrategies (4)	1–4	3, 4, 6, 10, 11, 12
AL/FundamentalAlgorithms (6)	1, 4–7	3, 4, 6, 8, 10, 11, 12
PL/ObjectOrientedProgramming (5)	1–3, 4, 6–7	throughout

On the flip side, most of the Programming Fundamentals (PF) topics are covered throughout this book. For the topics that are covered more in selected chapters but are not pervasive throughout, we also index the other way around.

Chapters	Topic Area
Chapter 3	AL/AlgorithmicStrategies AL/FundamentalAlgorithms
Chapter 4	AL/AlgorithmicStrategies AL/FundamentalAlgorithms
Chapter 5	
Chapter 6	AL/AlgorithmsAndComplexity AL/AlgorithmicStrategies AL/FundamentalAlgorithms
Chapter 8	PF/Recursion AL/FundamentalAlgorithms
Chapter 9	DS/GraphsAndTrees
Chapter 10	DS/GraphsAndTrees AL/AlgorithmicStrategies AL/FundamentalAlgorithms
Chapter 11	AL/AlgorithmicStrategies AL/FundamentalAlgorithms
Chapter 12	DS/GraphsAndTrees AL/AlgorithmicStrategies AL/FundamentalAlgorithms
Chapter 13	DS/GraphsAndTrees

I have generally found that I can teach straight through in chapter order, although the vagaries of semester scheduling sometimes make it necessary to move the position of Chapter 6 around. Chapters 7 and 8, except for Section 7.5 on priority queues, do not depend on the chapter on asymptotics, and I sometimes move that material ahead of Chapter 6 in order to do the first midterm exam only on completed chapters. A decision like this is usually based on the calendar, and I have found our fall and spring semesters to be slightly different.

Additional resources for students and instructors are provided at go.jblearning.com/Buell. Students and instructors may download source code related to the text. In addition, PowerPoint Lecture Outlines, Solutions to end-of-chapter Exercises, and a Test Bank are available for free instructor download.

For the mistakes and failings of this book I will take full responsibility, but for whatever is good about this book I must thank in part all people who have helped me become a better programmer and a better teacher. The students of the last three years who have looked at versions of this text and made comments have all contributed. My colleagues at the Institute for Defense Analyses have played a part in this text, as have the reviewers and the students who have commented, caught my errors, and helped this be better than it might have been. Thank you to Dr. Alice Armstrong, DSc, Shippensburg University; John Boyland, PhD, University of Wisconsin–Milwaukee; and Hélène Martin, Garfield High School for reviewing the manuscript. I owe a special debt to the copyeditors and proofreaders at Jones & Bartlett Learning. Their work has greatly improved the consistency of the text. We all might like to think we write well; the editors are those who make it appear that we actually do. I would also like to thank the editors at Jones & Bartlett Learning; Tim Anderson, Senior Acquisitions Editor; Amy Rose, Director of Production; and Amy Bloom, Managing Editor. Finally, I thank my wife Mary Ann for being patient with me for my many hours spent in the office staring at the screen.

Duncan Buell
Columbia, South Carolina