preface

Welcome to the third edition of *Object-Oriented Data Structures Using Java™*. This book presents the algorithmic, programming, and structuring techniques of a traditional data structures course in an object-oriented context. You'll find the familiar topics of stacks, queues, lists, trees, graphs, sorting, searching, Big-O complexity analysis, and recursion, all covered from an object-oriented point of view using Java. We stress software engineering principles throughout, including modularization, information hiding, data abstraction, stepwise refinement, the use of visual aids, the analysis of algorithms, and software verification methods.

To the Student

At this point you have completed at least one semester of computer science coursework. You know that an algorithm is a sequence of unambiguous instructions for solving a problem. You can take a problem of moderate complexity, design a small set of classes/objects that work together to solve the problem, code the method algorithms needed to make the objects work, and demonstrate the correctness of your solution.

Algorithms describe actions. These actions manipulate data. For most interesting problems that are solved using computers, the structure of the data is just as important as the structure of the algorithms used to manipulate the data. Using this textbook you will discover that the way you structure data affects how efficiently you can use the data; you will see how the nature of the problem you are attempting to solve dictates your structuring decisions; and you will learn about the data structures that computer scientists have developed over the years to help solve problems.

Abstract Data Types

Over the last 20 years the focus of the data structures course has broadened considerably. The topic of data structures now has been subsumed under the broader topic of *abstract data types* (*ADTs*)—the study of classes of objects whose logical behavior is

defined by a set of values and a set of operations. The term *abstract data type* represents a domain of values and a set of operations on those values that are specified independently of any particular implementation. The shift in emphasis is representative of the move toward more abstraction in computing education. We are interested in the abstract properties of classes of data objects in addition to how the objects might be represented in a program.

In this textbook we view our data structures from three different perspectives: their specification, their application, and their implementation. The specification describes the logical or abstract level—*what* the logical relationships among the data elements are and *what* operations can be performed on the structure. The application level, sometimes called the user level, is concerned with how the data structure is used to solve a problem—*why* the operations do what they do. The implementation level involves the coding details—*how* the structures and operations are implemented.

Object-Oriented Programming with Java

Our primary goal is to present the traditional data structure topics with an emphasis on problem solving and software design. Using the Java programming language as a vehicle for problem solutions, however, presents an opportunity for students to expand their familiarity with a modern programming language and the object-oriented paradigm. As our data structure coverage unfolds, we introduce and use the appropriate Java constructs that support our primary goals. Starting early and continuing throughout the text, we introduce and expand on the use of many Java features such as classes, objects, generics, packages, interfaces, library classes, inheritance, and exceptions. Our case studies demonstrate how to identify and filter candidate classes and how to organize modular solutions to interesting problems. We use Universal Modeling Language (UML) class diagrams throughout to help us model and visualize our classes and their interrelationships.

Second Edition Improvements Retained

The second edition of this textbook included many significant changes to the first edition. This third edition retains and builds on all of those improvements. We maintain the early introduction to the heart of the textbook material, introducing data structures and the use of references (pointers, links) as a structuring mechanism in Chapter 1. We have retained the popular Chapter 2, which was added to the second edition, where we introduce a simple yet interesting ADT (a StringLog). Our development of this ADT, including both array-based and reference-based implementations, acts as a gentle introduction to the approaches used for the more complicated ADTs throughout the rest of the text.

In the second edition we also rearranged our coverage of the classic data structures, starting with the simpler stack and queue structures and then moving to the more complicated lists, trees, and graphs. We continue to follow that approach, as well as introducing recursion much earlier (Chapter 4 rather than Chapter 7), immediately after our coverage of the related Stack ADT. This rearrangement of topics from the second edition allows our List ADT coverage in Chapter 6 to include both traditional and recursive implementations of list operations, and to include a presentation of an indexed list ADT.

In addition to the major structural changes in the second edition, we streamlined our presentation of concepts throughout, added even more exercises, and included many additional example applications. We took advantage of several of the new features of Java 5.0, including autoboxing and the Scanner class, to simplify our problem solutions. Two new appendices that rounded out our second edition improvements have been retained—one devoted to Java 5.0's new generics mechanism and one providing Application Programmer Interfaces (APIs) for the Java library classes used throughout the textbook.

New to the Third Edition

When we published the second edition we chose to omit generics for reasons outlined in Section 3.2 of that edition. We believe that was a reasonable decision at the time—but since then, Java generics have become a mature technology. Therefore, we now include their use throughout the textbook, providing the dual benefits of allowing for a type-safe use of data structures while exposing students to modern approaches.

With this edition we are pleased to be among the first data structures textbooks to address the topics of concurrency and synchronization, which are growing in importance as computer systems move to using more cores and threads to obtain additional performance with each new generation. We introduce the topic in the new Section 5.7 where we start with the basics of Java threads, continue through examples of thread interference and synchronization, and culminate in a discussion of efficiency concerns.

In addition to the two major changes described above, we have improved the book in many smaller ways. We have included more code examples, added programming exercises including several project-type exercises, rearranged the order of presentation of topics in several chapters, simplified the list architecture used in Chapter 6, and clarified many tables and figures. In the robust set of exercises at the end of each chapter, you will still find the familiar computer icon indicating an exercise involving programming—but now you will see two such icons beside those problems with a "significant" programming component, indicating exercises that might be used for major class projects.

We hope that you enjoy this updated, modern approach to the data structures course.

Prerequisite Assumptions

In this book, we assume that readers are familiar with the following Java constructs:

- Built-in simple data types and the array type
- Control structures *while*, *do*, *for*, *if*, and *switch*
- Creating and instantiating objects
- Basic user-defined classes
  - variables and methods
  - constructors, method parameters, and the *return* statement
  - visibility modifiers

Throughout the text we use several of the support classes from the Java Class Library, such as *String*, *Scanner* (new in Java 5.0), *System*, *Random*, and *Math*. Appendix E provides an introduction to these classes.

Input/Output

It is difficult to know what background the students using a data structures textbook will have in Java I/O. Some may have learned Java in an environment where the Java input/output statements were "hidden" behind a package provided with their introductory textbook. Others may have learned graphical input/output techniques, but never learned how to do file input/output. Some have learned how to create graphical interfaces using the Java AWT; others have learned Swing; others have learned neither. To allow all the students using our textbook to concentrate on the primary topic of data structures, we use the simplest I/O approach we can, namely console I/O. For input we use the *Scanner* class, a class introduced in Java 5.0 that greatly simplifies the input task. Output is accomplished using the simple *System.out.print* command.

To support those teachers and students who prefer to work with graphical user interfaces (GUIs), we provide GUIs for many of our case studies (in addition to the console-based solutions). In this way they have a code base to support instruction and additional work using GUIs. At the conclusion of each case study we discuss the GUI-based solution, include some screenshots of the program in action, and provide some related exercises.

Content and Organization

Chapter 1 is all about **Getting Organized.** It introduces ways of organizing software development and software solutions. An overview of object orientation stresses mechanisms for organizing objects and classes of objects. Our primary topic of data structures starts with a look at the classic structures and the two fundamental language constructs that are used to implement those structures: the array and the reference (link/pointer). The chapter concludes with a study of Big-O analysis—how we evaluate algorithms that provide access to, or otherwise use, our data structures.

Chapter 2 introduces **Abstract Data Types** (ADTs). We view data from three different levels: the logical, application, and implementation levels. We introduce the Java *interface* mechanism as a means of supporting this three-tiered view. As a simple example of an ADT we present a collection of strings and show how it is handled at each of the three levels. For the implementation level we include both array-based and reference-based approaches. To support the reference-based approach we introduce the linked list structure. We also address ways of verifying the correctness of our work. Finally, in a case study, we see how the use of abstraction simplifies the task of implementing a trivia game system.

Chapter 3 presents **The Stack ADT.** The stack is first considered from its abstract perspective, and the idea of recording the logical abstraction in an ADT specification as a Java *interface* is reinforced. Sub-interfacing allows us to define both bounded and unbounded stack abstractions. We investigate the kinds of elements we should store in

our collection ADTs, such as the stack, to make them generally usable. We also study ways of handling exceptional situations that might arise when using our ADTs. We show how stacks are used to determine if a set of grouping symbols is well formed and to support evaluation of mathematical expressions. We investigate the implementation of stacks using the two basic implementation approaches introduced previously in the text: arrays and references. We also investigate an approach using the Java Library class *ArrayList*.

Chapter 4 discusses **Recursion**, first providing an intuitive view of the concept, and then showing how recursion can be used to solve programming problems. Guidelines for writing recursive methods are illustrated with many examples. After demonstrating that a by-hand simulation of a recursive routine can be very tedious, a simple three-question technique is introduced for verifying the correctness of recursive methods. A more detailed discussion of how recursion works leads to an understanding of how recursion can be replaced with iteration and stacks. Our sample applications include the classic Towers of Hanoi and Blob Counting (image analysis).

Chapter 5 presents **The Queue ADT**. As with the stack, the queue ADT is first considered from its abstract perspective, followed by a formal specification, and then implemented using both array-based and reference-based approaches. We include an array-based approach to implementing an unbounded queue. Example applications for the queue involve checking for palindromes, simulating the card game War, and simulating a system of real-world queues. Finally, we look at Java's concurrency and synchronization mechanisms, explaining issues of interference and efficiency.

Chapter 6 introduces **The List ADT**. Because list management requires us to directly compare objects, the chapter begins with a review of that topic. This is followed by a general discussion of lists and then a formal specification of a list framework, supporting unsorted, sorted, and indexed lists. We use inheritance to take advantage of the commonalities among our list variations for both our array-based and reference-based implementations. Three interesting applications, involving poker, golf, and music, demonstrate how each of the list variations can be used to help solve problems. This chapter includes a study of the binary search algorithm, which is useful when searching for an element in an array-based sorted list. The chapter concludes with a section on the practical topic of storing and retrieving data structures using files.

Chapter 7 looks at **More Lists**: circular linked lists, doubly linked lists, and lists with headers and trailers. An alternative representation of a linked structure, using static allocation (an array of nodes), is designed. The case study uses a list ADT developed specifically to support the implementation of large integers.

Chapter 8 introduces **Binary Search Trees** as a way to arrange data, giving the flexibility of a linked structure with efficient insertion and deletion time. We exploit the inherent recursive nature of binary trees by presenting recursive algorithms for many of the operations. We also address the problems of balancing binary search trees and implementing them with an array. The case study discusses the process of building an index for a manuscript and implements the first phase of the process.

Chapter 9 presents a collection of other ADTs: **Priority Queues, Heaps, and Graphs**. The graph algorithms make use of stacks, queues, and priority queues, thus

both reinforcing earlier material and demonstrating the general usability of these structures.

Chapter 10 presents a number of **Sorting and Searching Algorithms**. The sorting algorithms that are illustrated, implemented, and compared include straight selection sort, two versions of bubble sort, insertion sort, quick sort, heap sort, and merge sort. The sorting algorithms are compared using Big-O notation. The discussion of algorithm analysis continues in the context of searching. Previously presented searching algorithms are reviewed and new ones are described. Hashing techniques are discussed in some detail.

Additional Features

Chapter Goals Sets of knowledge and skill goals are presented at the beginning of each chapter to help the students assess what they have learned.

Sample Programs Numerous sample programs and program segments illustrate the abstract concepts throughout the text.

Case Studies Each of the five major case studies includes a problem description, an analysis of the problem, the identification of a set of support classes to use in solving the problem, the development of the code for the support classes and the driving application, and a discussion of testing the solution. The class identification stage includes descriptions of brainstorming, filtering, and scenario analysis techniques as needed.

Chapter Summaries Each chapter concludes with a summary section that reviews the most important topics of the chapter and ties together related topics.

Chapter Exercises We average more than 40 exercises per chapter. The exercises are organized by chapter sections to make them easier for you to manage. They vary in levels of difficulty, including short and long programming problems (marked with "programming-required" icons—one icon to indicate short exercises and two icons for projects), the analysis of algorithms, and problems to test students' understanding of abstract concepts.

Appendices The appendices summarize the Java reserved word set, operator precedence, primitive data types, the ASCII subset of Unicode, and the Java library classes used in the textbook.

Website http://www.jblearning/catalog/9781449613549/

This website provides access to the textbook's source code files, presentation slides for each chapter, and a glossary of terms. Additionally, registered instructors are able to access answers to most of the textbook's exercises and a test item file. Please contact the authors if you have material related to the text that you would like to share with others.

Acknowledgments

We would like to thank the following people who took the time to review this textbook: Mark Llewellyn at the University of Central Florida, Chenglie Hu at Carroll College, Val Tannen at the University of Pennsylvania, Chris Dovolis at the University of Minnesota, Mike Coe at Plano Senior High School, Mikel Petty at University of Alabama in Huntsville, Gene Sheppard at Georgia Perimeter College, Noni Bohonak at the University of South Carolina–Lancaster, Jose Cordova at the University of Louisiana–Monroe, and Judy Gurka at the Metropolitan State College of Denver. A special thanks to Christine Shannon at Centre College, to Phil LaMastra at Fairfield University, and to Kristen Obermyer and Tara Srihara, both at Villanova University, for specific comments leading to improvements to this edition.

A virtual bouquet of roses to the people at Jones & Bartlett Learning who contributed so much, especially Tim Anderson, Amy Rose, Tiffany Sliter, Melissa Potter, and Stephanie Sguigna.

ND
DJ
CW