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 with that language. That course should have covered the fundamental statements and data types, as well as arrays. Appendix 0 has a review of that material.

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-map 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, the framework is available for later courses in the curriculum, and beyond! Third, although the primary emphasis is on *using* the Java Collections Framework, the framework classes are not treated simply as "black boxes." For each such class, the heading and fields are provided, and one method definition is dissected. This exposition takes away some of the mystery that would otherwise surround the class, and allows students to see the efficiency and succinctness of professionals' code.

The version of the Java Collections Framework we will be working with includes type parameters. Type parameters, sometimes called "generics" or "templates," were added to the Java language starting with Java 2 Standard Edition (J2SE), version 1.5. Two other features that complement type parameters are ***boxing***, whereby a primitive value is automatically converted to an object, and ***unboxing***, whereby an object from wrapper class (such as Integer) is automatically converted to the corresponding value in a primitive type (such as **int**).

Other Implementations Considered

As important as the Java Collections Framework implementations are, they cannot be the exclusive focus of such a fundamental course in data structures and algorithms. Approaches that differ from those in the framework deserve consideration. For example, the HashMap class utilizes 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 a Network class) and algorithms (such as Heap Sort) that are not yet included in the Java Collections Framework.

Sometimes, the complexity of the framework classes is mitigated by first introducing simpler versions of those classes. For example, the SinglyLinkedList class – not in the Java Collections Framework – helps to pave the way for the more powerful LinkedList class, which is in the framework. And the BinarySearchTree class prepares students to understand the framework's TreeMap class, based on red-black trees.

This text satisfies another important goal of a data structures and algorithms course: Students have the opportunity to develop 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 many other projects to develop or extend applications that *use* the Java Collections Framework.

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 most chapters conclude with several major programming assignments. Each chapter also has a variety of exercises, and the answers to all of the exercises are available to the instructor.

Each data structure is carefully described, with the specifications for each method given in javadoc notation. Also, there are examples of how to call the method, and the results of that call. To reinforce the important aspects of the material and to hone students' coding skills in preparation for programming projects, there is a suite of 23 lab experiments. The organization of these labs is described later in this preface.

Support Material

The web site for all of the support material is

www.mhhe.com/collins

That web site has links to the following information for students:

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

Additionally, instructors can obtain the following from the web site:

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

Synopses of the Chapters

Chapter 1 focuses on the fundamentals of object-oriented programming. The `String` class serves as a vehicle for some background material on constructors and references. Then encapsulation, inheritance and polymorphism are introduced. For a concrete illustration of these topics, an interface is created and implemented, and the implementation is extended. The relationship between abstract data types and interfaces is explored, as is the corresponding connection between classes and data structures. The

Universal Modeling Language provides a design tool to depict the interrelationships among interfaces, classes and subclasses.

Chapter 2 introduces some additional features of the Java language. For example, there are sections on exception handling, file input and output, and the Java Virtual Machine. There is also a section on the Object class's equals method, why that method should be overridden, and how to accomplish the overriding.

Chapter 3, *Analysis of Algorithms*, starts by defining functions to estimate a method's execution-time requirements, both in the average and worst cases. Big-O notation provides a convenient tool for estimating these estimates. Because Big-O notation yields environment-independent estimates, these results are then compared with actual run-times, which are determined with the help of the Random class and currentTimeMillis method.

Chapter 4 outlines the Java Collections Framework. We start with some preliminary material on collection classes in general, type parameters and the iterator design-pattern. The remainder of the chapter presents the major interfaces (Collection, List, Set, Map) and their implementations. Special attention is given to comparing those implementations, for example, ArrayList versus LinkedList and TreeMap versus HashMap.

Chapter 5, on recursion, represents a temporary shift in emphasis from data structures to algorithms. There is a gradual progression from simple examples (factorial and decimal-to-binary) to more powerful examples (binary search and backtracking). The mechanism for illustrating the execution of recursive methods is the execution frame. Backtracking is introduced, not only as a design pattern, but as another illustration of creating polymorphic references through interfaces. And the same BackTrack class is used for maze-searching, eight queens and knight's tour!

In Chapter 6, we study the Java Collections Framework's ArrayList class. An ArrayList object is a smart array: automatically resizable, and with methods to handle insertions and deletions at any index. The design starts with the method specifications for some of the most widely-used methods in the ArrayList class. There follows a brief outline of the implementation of the class. 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 also introduces a “theme” project: to develop an integrated web browser and search engine. This project, based on a paper by Newhall and Meeden [2002], assumes some familiarity with graphical user interfaces. The project continues through five of the remaining chapters, and clearly illustrates the practical value of understanding data structures.

Chapter 7 presents linked lists. A discussion of singly-linked lists leades to the development of a primitive SinglyLinkedList class. This serves mainly to prepare students for the framework’s LinkedList class. LinkedList objects are characterized by linear-time methods for inserting, removing or retrieving at an arbitrary position. This property makes a compelling case for *list iterator*s: objects that traverse a LinkedList object and have constant-time methods for inserting, removing or retrieving at the “current” position. The 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 8. The Java Collections Framework has a Queue interface, but that interface allows the removal of any element from a queue! Because this violates the definition of a queue, we instead create a simple PureQueue interface that corresponds to the abstract data type queue. Implementations based on a LinkedList and on an array are developed and analyzed. The specific application, calculating the average waiting time at a car wash, falls into the general category of *computer simulation*.

A PureStack interface, with several simple implementations, allows us to bypass the framework’s Stack class. The fatal flaw in the Stack class is that elements can be inserted or removed anywhere in a Stack object. There are two applications of stacks: the implementation of recursion by a compiler, and the conversion from infix to postfix. This latter application is expanded in a lab, and forms the basis for a project on evaluating a condition.

Chapter 9 focuses on binary trees in general, as a prelude to the material in Chapters 10 through 13. The essential features of binary trees are presented, including both botanical (root, branch, leaf) and familial (parent, child, sibling) terms. Binary trees are important for understanding later material on AVL trees, decision trees, red-black trees, heaps and Huffman trees.

In Chapter 10, we look at binary search trees, including a BinarySearchTree class, and explain the value of balanced binary search trees. Rotations are introduced as the mechanism by which re-balancing is accomplished, and AVL trees are offered as examples of balanced binary search trees. An AVLTree class, as a subclass of BinarySearchTree, is outlined; the crucial methods, fixAfterInsertion and fixAfterDeletion, are given as programming projects.

Sorting is the topic of Chapter 11. Estimates of the lower bounds for comparison-based sorts are determined. A few simple sorts are defined, and then we move on to two sort methods provided by the framework. Quick Sort sorts an array of a primitive type, and Merge Sort works for an array of objects and for implementations of the List interface. A lab experiment compares all of these sort algorithms on randomly-generated integers.

The central topic of Chapter 12 is how to use the TreeMap class. A ***map*** is a collection in which each element has a unique key part and also a value part. In the TreeMap implementation of the Map interface, the elements are stored in a red-black tree, ordered by the elements' keys. There are labs to guide students through the details of re-structuring after an insertion or removal. The application consists of searching a thesaurus for synonyms. The TreeSet class has a *TreeMap* field in which each element has the same, dummy value-part. The application of the TreeSet class is a simple spell-checker.

Chapter 13 introduces the PurePriorityQueue 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 smallest-valued element in logarithmic worst time. The application is in the area of data compression: Given a text file, generate a minimal, prefix-free encoding. The project assignment is to convert the encoding back to the original text file.

Chapter 14 investigates hashing. The Java Collections Framework has a HashMap class for elements that consist of unique-key/value pairs. Basically, the average time for insertion, removal and searching is constant! This average speed is exploited in an application to create a simple symbol table. The implementation, using chained hashing is compared to open-address hashing.

The most general data structures – graphs, trees and networks – are presented in Chapter 15. There are outlines of the essential algorithms: breadth-first traversal, depth-first traversal, finding a minimal spanning tree, and finding the shortest or longest 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 Network. 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 5.

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.

Appendixes

Appendix 0 has a review of Java topics assumed in the rest of the text: primitive types, the StringTokenizer class, console-oriented input, and arrays. There are also several programming exercises to reinforce the material presented.

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.

Appendix 2 has two additional features of the Java Collections Framework. Each of the collection classes in the framework is ***serializable***, that is, an instance of the class can be conveniently stored to an output stream, and the instance can later be re-constituted from an input stream (de-serialization). Framework iterators are ***fail-fast***: During an iteration through a collection, there should be no insertions into or removals from the collection except by the iterator. Otherwise, the integrity of that iterator may be compromised, so an exception will be thrown as soon as the iterator's unreliability has been established.

Organization of the Labs

There are 23 Web labs associated with this text. For both students and instructors, the initial 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:

a. they can be assigned as closed labs;
b. they can be assigned as open labs;
c. 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 creating a greedy cycle to solve the Traveling Salesperson Problem. 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.