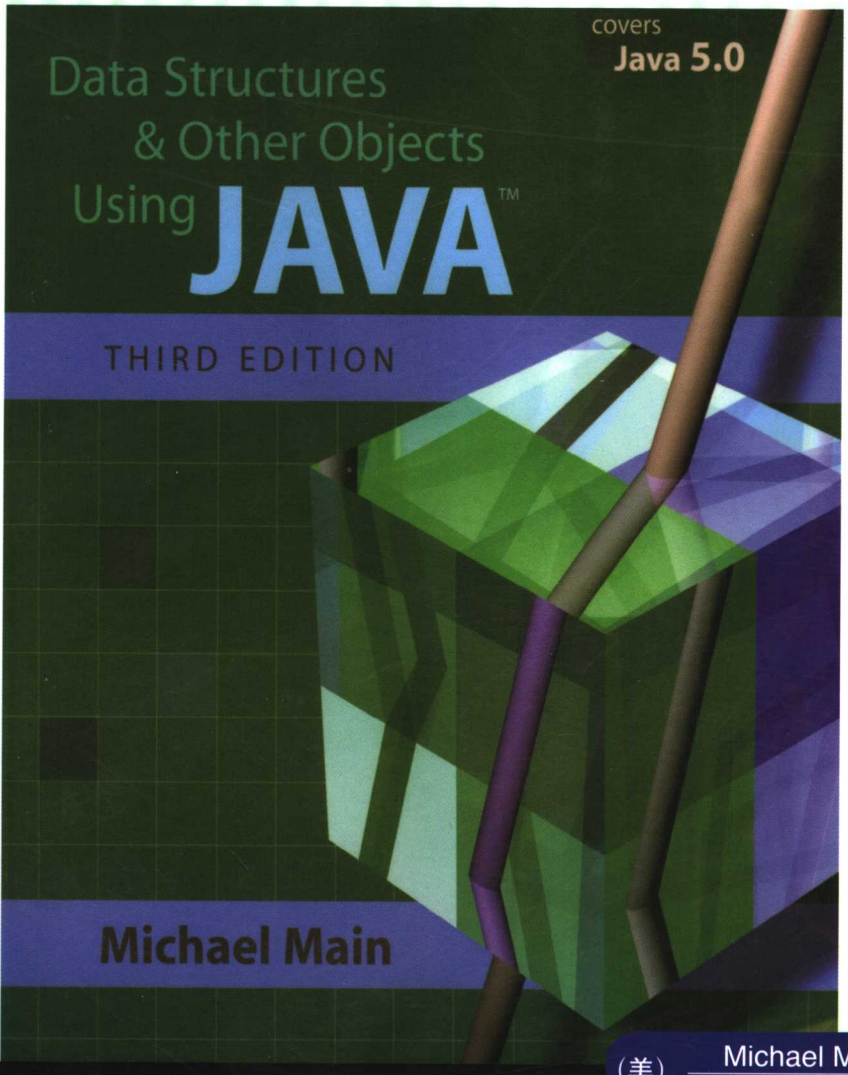
经 典 原 版 书 库

# 数据结构

## Java语言描述

（英文版 · 第3版）




（美） Michael Main 著
科罗拉多大学

机械工业出版社
China Machine Press

经 典 原 版 书 库

# 数据结构

## Java语言描述

（英文版 · 第3版）

Data Structures and Other Objects Using Java

(Third Edition)



(美) Michael Main
科罗拉多大学 著

机械工业出版社
China Machine Press

**本书版权登记号：图字：01-2006-3112**

**图书在版编目（CIP）数据**

数据结构：Java语言描述（英文版·第3版）/（美）梅因（Main, M.）著. -北京：机械工业出版社，2006.8

（经典原版书库）

书名原文：Data Structures and Other Objects Using Java, Third Edition

ISBN 7-111-19610-4

Ⅰ.数…　Ⅱ.梅…　Ⅲ.①数据结构-英文　②JAVA语言-程序设计-英文

Ⅳ.①TP311.12　②TP312

中国版本图书馆CIP数据核字（2006）第080894号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：迟振春

北京诚信伟业印刷有限公司印刷　·　新华书店北京发行所发行

2006年8月第1版第1次印刷

170mm×242mm　·　52.25印张

定价：79.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

# 出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江

大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下:

电子邮件: hzjsj@hzbook.com
联系电话: (010) 68995264
联系地址: 北京市西城区百万庄南街1号
邮政编码: 100037

# Preface

Java provides programmers with an immediately attractive forum. With Java and the World Wide Web, a programmer's work can have quick global distribution with appealing graphics and a marvelous capability for feedback, modification, software reuse, and growth. Certainly, Java's other qualities have also contributed to its rapid adoption: the capability to implement clean, object-oriented designs; the familiarity of Java's syntactic constructs; the good collection of ready-to-use features in the Java Class Libraries. But it's the winsome ways of the World Wide Web that have pulled Java to the front for both experienced programmers and the newcomer entering a first-year computer science course.

With that said, it's satisfying to see that the core of the first-year courses remain solid even as courses start using Java. The proven methods of representing and using data structures remain vital to beginning programmers in a data structures course, even as the curriculum is shifting to accommodate an object-oriented approach.

This book is written for just such an evolving data structures course, with the students' programming carried out in Java. The text's emphasis is on the *specification*, *design*, *implementation*, and *use* of the basic data types that are normally covered in a second-semester, object-oriented course. There is wide coverage of important programming techniques in recursion, searching techniques, and sorting. The text also provides coverage of big-$O$ time analysis of algorithms, an optional appendix on writing an interactive applet to test an implementation of a data structure, and using Javadoc to specify precondition–postcondition contracts.

The text assumes that the student has already had an introductory computer science and programming class, but we do include coverage of those topics (such as the Java `Object` type and a precise description of parameter passing) that are not always covered completely in a first course. The rest of this preface discusses ways that this material can be covered, starting with new material for this third edition.

## Third Edition: J2SE 5.0

The new material in this edition is primarily from Sun Microsystems latest release of the Java 2 Standard Edition 5.0 (J2SE 5.0). Many of the features in this release provide support for data structures concepts that were previously dealt with in an *ad hoc* manner. In particular, this edition of the book incorporates the following features at points where their motivation is natural:

**True Generics (introduced in Chapter 5).** The largest change in J2SE 5.0 is the support for generic methods and generic classes that depend on an unspecified underlying data type. We use generics, for example, in building a class for a collection of `E` objects, where the type `E` is left unspecified. Generics are introduced in Chapter 5, which is the location where we previously implemented a simple form of generics that used Java's `Object` class as the type of the elements. This occurs after the students have already seen two fundamental collection implementations that use an array or linked list of a specific type of element. By first showing these fundamental implementations without generics, the students learn important concepts such as building clones and manipulating linked lists without the distraction of secondary issues. After Chapter 5, generic classes are incorporated into the material in a straightforward manner.

It's interesting to note that the runtime implementation of generics in J2SE 5.0 uses the Java Object type for elements in the same way as the generic classes that programmers built before J2SE 5.0. The difference is that the extra syntax of Java generics allows new compile-time type checks to prevent the unintended mixing of types (such as inserting a string into a collection of integers).

**Input/Output (introduced in Chapter 1 and Appendix B).** Formatted output (with the `System.out.printf` method) and the input of primitive values (with the `java.util.Scanner` class) has been incorporated throughout the text. Our earlier input and output classes (EasyReader and FormatWriter) are still available online at `www.cs.colorado.edu/~main/edu/colorado/io`.

**Variable Arity Methods (introduced on page 112 in Chapter 3).** In J2SE 5.0, a method may have a variable number of arguments. The idea is shown in Chapter 3, though it doesn't play a large role in data structures.

**Enhanced For-Loops (introduced on page 108 in Chapter 3 and page 286 in Chapter 5).** J2SE 5.0 introduced a new form of the for-loop that allows the easy iteration through the elements of an array, a collection of elements that has implemented the Iterable interface, or the elements of an enum type. We use the new form throughout the text whenever it is appropriate.

**Autoboxing and Auto-Unboxing of Primitve Values (introduced on page 247 of Chapter 5).** Autoboxing allows the automatic conversion of a primitive value (such as an `int`) to a Java object (such as the `Integer` class). Auto-unboxing converts from the object back to the primitive value in most situations. At run time, the typecasts still occur, though the syntax of the program is cleaner.

**Covariant Return Types (covered on page 656 in Chapter 13).** J2SE 5.0 allows covariant return types, which means that the return type of an overridden inherited method may now be any descendant class of the original return type. We use this idea for clone methods throughout the text, thereby avoiding a type-cast with each use of a clone method and increasing type security. The larger use of the technique is postponed until the chapter on Inheritance.

**Enum Types (covered on page 687 in Chapter 13).** Enum types provide a convenient way to define a new data type whose objects may take on a small set of discrete values.

Outside of these new language features, the third edition of this text presents the same foundational data structures material as the earlier editions. In particular, the data structures curriculum emphasizes the ability to specify, design, and analyze data structures independently of any particular language, as well as the ability to implement new data structures and use existing data structures in any modern language. You'll see this approach in the first four chapters as students review Java and then go on to study and implement the most fundamental data structures (bags and sequence classes) using both arrays and linked-list techniques that are easily applied to any high-level language.

*new material on Java interfaces and the API classes*

Chapter 5 is a bit of a departure, bringing forward several Java-specific techniques that are particularly relevant to data structures: how to build generic collection classes based on Java's new generic types, using Java interfaces and the API classes, and building and using Java iterators. Although much of the subsequent material can be taught without these Java features, this is a natural time for students to learn them.

Later chapters focus on particular data structures (stacks, queues, trees, hash tables, graphs) or programming techniques (recursion, sorting techniques, inheritance). The introduction of recursion is delayed until just before the study of trees. This allows us to introduce recursion with some simple but meaningful examples that do more than just tail recursion. This avoids the mistaken first view that some students get that recursion is some kind of magic loop. After the first simple examples (including one new example for this edition), students see significant examples involving two recursive calls involving common situations with binary trees.

*new projects for trees and other areas*

Trees are particularly emphasized in this text, with Chapter 10 taking the students through two meaningful examples of data structures that use balanced trees (the heap project and the B-tree project). Additional projects for trees and other areas have also been added online at `http://www.aw.com/cssupport`.

Other new Java features have been carried over from the second edition of the text as needed: the use of assertions to check preconditions, a more complete coverage of inheritance, a new example of an abstract class to build classes that play two-play strategy games such as Othello or Connect Four, an introduction to new classes of the Java API (now in the generic form of `ArrayList`, `Vector`, `HashMap` and `HashTable`), and new features of Javadoc.

All these new bells and whistles of Java are convenient, and students need to be up-to-date—but it's the approaches to designing, specifying, documenting, implementing, using, and analyzing the data structures that will have the most enduring effect on your students.

## The Five Steps for Each Data Type

The book's core consists of the well-known data types: *sets, bags* (or *multisets*), *sequential lists*, *stacks, queues, tables,* and *graphs*. There are also additional supplemental data types such as a priority queue. Some of the types are approached in multiple ways, such as the bag class that is first implemented by storing the elements in an array and is later reimplemented using a binary search tree. Each of the data types is introduced following a pattern of five steps.

**Step 1: Understand the Data Type Abstractly.** At this level, a student gains an understanding of the data type and its operations at the level of concepts and pictures. For example, a student can visualize a stack and its operations of pushing and popping elements. Simple applications are understood and can be carried out by hand, such as using a stack to reverse the order of letters in a word.

**Step 2: Write a Specification of the Data Type as a Java Class.** In this step, the student sees and learns how to write a specification for a Java class that can implement the data type. The specification, written using the Javadoc tool, includes headings for the constructors, public methods, and sometimes other public features (such as restrictions tied to memory limitations). The heading of each method is presented along with a precondition–postcondition contract that completely specifies the behavior of the method. At this level, it's important for the students to realize that the specification is not tied to any particular choice of implementation techniques. In fact, this same specification may be used several times for several different implementations of the same data type.

**Step 3: Use the Data Type.** With the specification in place, students can write small applications or applets to show the data type in use. These applications are based solely on the data type's specification because we still have not tied down the implementation.

**Step 4: Select Appropriate Data Structures and Proceed to Design and Implement the Data Type.** With a good abstract understanding of the data type, we can select an appropriate data structure such as an array, a linked list of nodes, or a binary tree of nodes. For many of our data types, a first design and implementation will select a simple approach such as an array. Later, we will redesign and reimplement the same data type with a more complicated underlying structure.

Since we are using Java classes, an implementation of a data type will have the selected data structures (arrays, references to other objects, etc.) as private instance variables of the class. In my own teaching, I stress the necessity for a

clear understanding of the rules that relate the private instance variables to the abstract notion of the data type. I require each student to write these rules in clear English sentences that are called the *invariant of the abstract data type*. Once the invariant is written, students can proceed to implementing various methods. The invariant helps in writing correct methods because of two facts: (a) Each method (except the constructors) knows that the invariant is true when the method begins its work; and (b) Each method is responsible for ensuring that the invariant is again true when the method finishes.

**Step 5: Analyze the Implementation.** Each implementation can be analyzed for correctness, flexibility, and time analysis of the operations (using big-*O* notation). Students have a particularly strong opportunity for these analyses when the same data type has been implemented in several different ways.

## Where Will the Students Be at the End of the Course?

At the end of our course, students understand the data types inside out. They know how to use the data types and how to implement them in several ways. They know the practical effects of the different implementation choices. The students can reason about efficiency with a big-*O* analysis and can argue for the correctness of their implementations by referring to the invariant of the ADT.

*the data types in this book are cut-down versions of the Java Class Libraries*

One of the lasting effects of the course is the specification, design, and implementation experience. The improved ability to reason about programs is also important. But perhaps most important of all is the exposure to classes that are easily used in many situations. The students no longer have to write everything from scratch. We tell our students that someday they will be thinking about a problem, and they will suddenly realize that a large chunk of the work can be done with a bag, a stack, a queue, or some such. And this large chunk of work is work that they won't have to do. Instead, they will pull out the bag or stack or queue that they wrote this semester—using it with no modifications. Or, more likely, they will use the familiar data type from a library of standard data types, such as the proposed *Java Class Libraries*. In fact, the behavior of some data types in this text is a cut-down version of the JCL, so when students take the step to the real JCL, they will be on familiar ground—from the standpoint of how to use the class and also having a knowledge of the considerations that went into building the class.

## Other Foundational Topics

Throughout the course, we also lay a foundation for other aspects of "real programming," with coverage of the following topics beyond the basic data structures material.

**Object-Oriented Programming.** The foundations of object-oriented programming are laid by giving students a strong understanding of Java classes. The important aspects of classes are covered early: the notion of a method, the

separation into private and public members, the purpose of constructors, and a small exposure to cloning and testing for equality. This is primarily in Chapter 2, some of which can be skipped by students with a good exposure to Java classes in the CS1 course.

Further aspects of classes are introduced when the classes first use dynamic arrays (Chapter 3). At this point, the need for a more sophisticated clone method is explained. Teaching this OOP aspect with the first use of dynamic memory has the effect of giving the students a concrete picture of how an instance variable is used as a reference to a dynamic object such as an array.

Conceptually, the largest innovation of OOP is the software reuse that occurs via inheritance. There are certainly opportunities for introducing inheritance right from the start of a data structures course (such as implementing a set class as a descendant of a bag class). However, an early introduction may also result in juggling too many new concepts at once, resulting in a weaker understanding of the fundamental data structures. Therefore, in my own course, I introduce inheritance at the end as a vision of things to come. But the introduction to inheritance (Sections 13.1 and 13.2) could be covered as soon as classes are understood. With this in mind, some instructors may wish to cover Chapter 13 earlier, just before stacks and queues, so that stacks and queues can be derived from another class.

Another alternative is to identify students who already know the basics of classes. These students can carry out an inheritance project (such as the ecosystem of Section 13.2), while the rest of the students first learn about classes.

**Java Objects.** The Java `Object` type lies at the base of all the other Java types—or at least almost all the other types. The eight primitive types are not Java objects, and for many students, the CS1 work has been primarily with the eight primitive types. Because of this, the first few data structures are collections of primitive values, such as a bag of integers or a sequence of double numbers.

**Iterators.** Iterators are an important part of the Java Class Libraries, allowing a programmer to easily step through the elements in a collection class. The `Iterator` interface is introduced in Chapter 5. Throughout the rest of the text, iterators are not directly used, although they provide a good opportunity for programming projects such as using a stack to implement an iterator for a binary search tree (Chapter 9).

**Recursion.** First-semester courses often introduce students to recursion. But many of the first-semester examples are tail recursion, where the final act of the method is the recursive call. This may have given students a misleading impression that recursion is nothing more than a loop. Because of this, I prefer to avoid early use of tail recursion in a second-semester course.

So, in our second-semester course, we emphasize recursive solutions that use more than tail recursion. The recursion chapter provides four examples along these lines. Two of the examples—generating random fractals and traversing a maze—are big hits with the students. The fractal example runs as a graphical

applet, and although the maze example is text based, an adventurous student can convert it to a graphical applet. These recursion examples (Chapter 8) appear just before trees (Chapter 9) since it is within recursive tree algorithms that recursion becomes vital. However, instructors who desire more emphasis on recursion can move that topic forward, even before Chapter 2.

In a course that has time for advanced tree projects (Chapter 10), we analyze the recursive tree algorithms, explaining the importance of keeping the trees balanced—both to improve worst-case performance and to avoid potential execution stack overflow.

**Searching and Sorting.** Chapters 11 and 12 provide fundamental coverage of searching and sorting algorithms. The searching reviews binary search of an ordered array, which many students will have seen before. Hash tables also are introduced in the search chapter by implementing a version of the JCL hash table and also a second hash table that uses chaining instead of open addressing. The sorting chapter reviews simple quadratic sorting methods, but the majority of the chapter focuses on faster algorithms: the recursive merge sort (with worst-case time of $O(n \log n)$), Tony Hoare's recursive quicksort (with average-time $O(n \log n)$), and the tree-based heapsort (with worst-case time of $O(n \log n)$).

## Advanced Projects

The text offers good opportunities for optional projects that can be undertaken by a more advanced class or by students with a stronger background in a large class. Particular advanced projects include the following:

- Interactive applet-based test programs for any of the data structures (outlined in Appendix I).
- Implementing an `Iterator` for the sequence class (see Chapter 5 Programming Projects).
- Writing a deep clone method for a collection class (see Chapter 5 Programming Projects).
- Writing an applet version of an application program (such as the maze traversal in Section 8.1 or the ecosystem in Section 13.2).
- Using a stack to build an iterator for the binary search tree (see Chapter 9 Programming Projects).
- A priority queue implemented as an array of ordinary queues (Section 7.4) or implemented using a heap (Section 10.1).
- A set class implemented with B-trees (Section 10.2). I have made a particular effort on this project to provide information that is sufficient for students to implement the class without need of another text. Advanced students have successfully completed this project as independent work.
- An inheritance project such as the ecosystem of Section 13.2.
- A graph class and associated graph algorithms from Chapter 14. This is another case in which advanced students may do work on their own.

## Java Language Versions

All the source code of the book has been tested to work correctly with Java Version 2 Standard Edition Version 5.0, including new features such as generics. Information on all of the Java products from Sun Microsystems is available at `http://java.sun.com/products/index.html`.

## Flexibility of Topic Ordering

This book was written to give instructors latitude in reordering the material to meet the specific background of students or to add early emphasis to selected topics. The dependencies among the chapters are shown on the next page. A line joining two boxes indicates that the upper box should be covered before the lower box.

Here are some suggested orderings of the material:

**Typical Course.** Start with Chapters 1–9, skipping parts of Chapter 2 if the students have a prior background in Java classes. Most chapters can be covered in a week, but you may want more time for Chapter 4 (linked lists), Chapter 8 (recursion), or Chapter 9 (trees). Typically, I cover the material in 13 weeks, including time for exams and extra time for linked lists and trees. Remaining weeks can be spent on a tree project from Chapter 10 or on binary search (Section 11.1) and sorting (Chapter 12).

**Heavy OOP Emphasis.** If students will cover sorting and searching elsewhere, then there is time for a heavier emphasis on object-oriented programming. The first three chapters are covered in detail, and then derived classes (Section 13.1) are introduced. At this point, students can do an interesting OOP project, perhaps based on the ecosystem of Section 13.2. The basic data structures (Chapters 4–7) are then covered, with the queue implemented as a derived class (Section 13.3). Finish up with recursion (Chapter 8) and trees (Chapter 9), placing special emphasis on recursive methods.

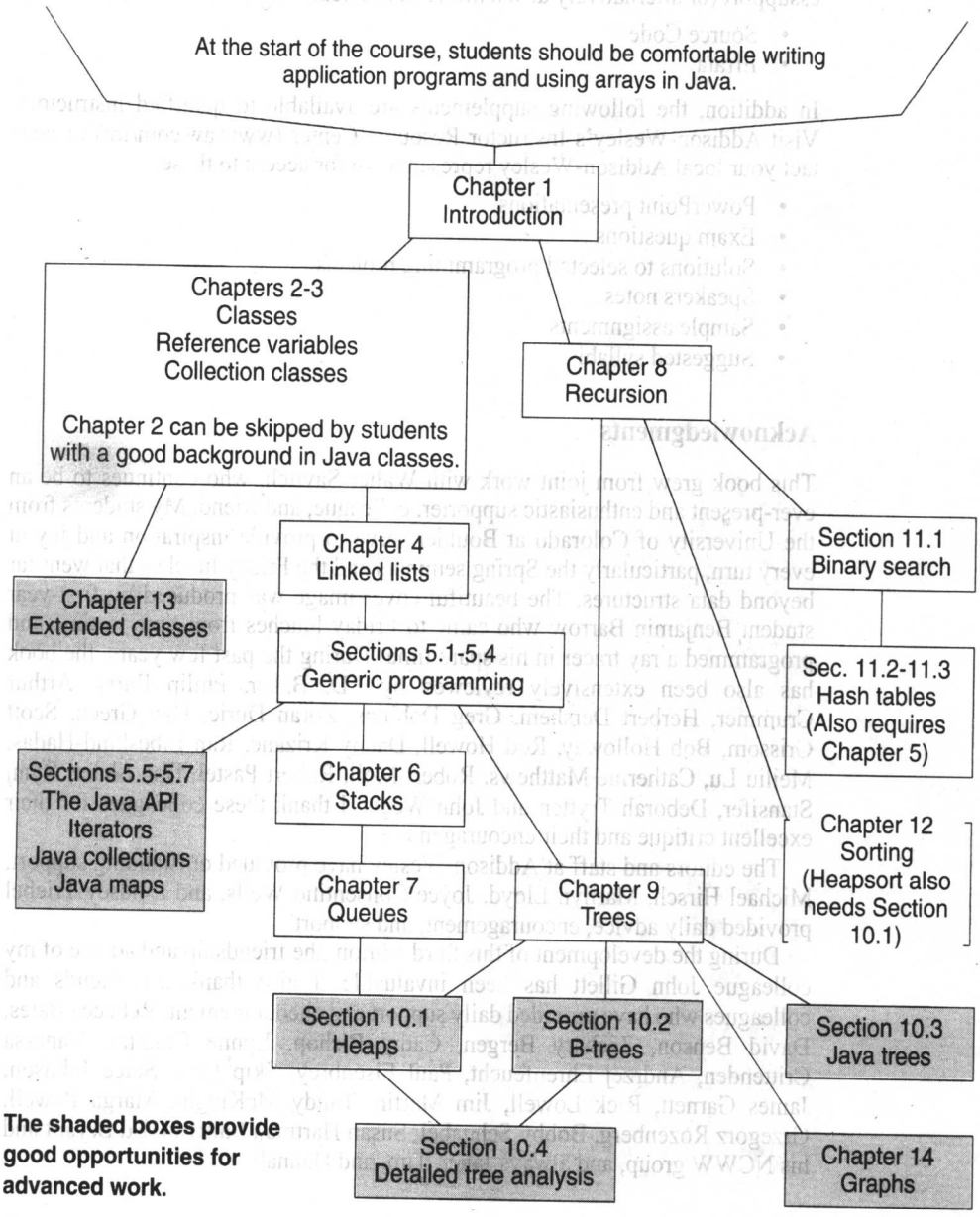
**Accelerated Course.** Assign the first three chapters as independent reading in the first week and start with Chapter 4 (linked lists). This will leave two to three extra weeks at the end of the term so that students can spend more time on searching, sorting, and the advanced topics (shaded in the chapter dependencies list).

We also have taught the course with further acceleration by spending no lecture time on stacks and queues (but assigning those chapters as reading).

**Early Recursion / Early Sorting.** One to three weeks may be spent at the start of class on recursive thinking. The first reading will then be Chapters 1 and 8, perhaps supplemented by additional recursive projects.

If the recursion is covered early, you may also proceed to cover binary search (Section 11.1) and most of the sorting algorithms (Chapter 12) before introducing collection classes.

## Chapter Dependencies




**The shaded boxes provide good opportunities for advanced work.**

## Supplements Via the Internet

The following materials are available to all readers of this text at www.aw.com/cssupport (or alternatively at www.cs.colorado.edu/~main/dsoc.html):

- Source Code
- Errata

In addition, the following supplements are available to qualified instructors. Visit Addison-Wesley's Instructor Resource Center (www.aw.com/irc) or contact your local Addison-Wesley representative for access to these:

- PowerPoint presentations
- Exam questions
- Solutions to selected programming projects
- Speakers notes
- Sample assignments
- Suggested syllabi

## Acknowledgments

This book grew from joint work with Walter Savitch, who continues to be an ever-present and enthusiastic supporter, colleague, and friend. My students from the University of Colorado at Boulder serve to provide inspiration and joy at every turn, particularly the Spring seminars and the Friday lunches that went far beyond data structures. The beautiful cover image was produced by first-year student Benjamin Barrow who came to Friday lunches from time to time and programmed a ray tracer in his spare time! During the past few years, the book has also been extensively reviewed by J.D. Baker, Philip Barry, Arthur Crummer, Herbert Dershem, Greg Dobbins, Zoran Duric, Dan Grecu, Scott Grissom, Bob Holloway, Rod Howell, Danny Krizanc, Ran Libeskind-Hadas, Meiliu Lu, Catherine Matthews, Robert Moll, Robert Pastel, Don Slater, Ryan Stansifer, Deborah Trytten and John Wegis. I thank these colleagues for their excellent critique and their encouragement.

The editors and staff at Addison-Wesley have provided outstanding support. Michael Hirsch, Marilyn Lloyd, Joyce Consentino Wells, and Lindsey Triebel provided daily advice, encouragement, and support.

During the development of this third edition, the friendship and advice of my colleague John Gillett has been invaluable. I also thank my friends and colleagues who have provided daily support and encouragement: Rebecca Bates, David Benson, Zachary Bergen, Cathy Bishop, Lynne Conklin, Vanessa Crittenden, Andrzej Ehrenfeucht, Paul Eisenbrey, Skip Ellis, Stace Johnson, James Garnett, Rick Lowell, Jim Martin, Tandy McKnight, Marga Powell, Grzegorz Rozenberg, Bobby Schnabel, Susan Hartman Sullivan, Ed Bryant and his NCWW group, and always Janet, Tim, and Hannah.

---

Michael Main (`main@colorado.edu`)
*Boulder, Colorado*

# Contents

## CHAPTER 2 Java Classes and Information Hiding 38