大学计算机教育国外著名教材系列（影印版）

PEARSON Addison Wesley

INTRODUCTION TO PROGRAMMING USING JAVA

AN OBJECT-ORIENTED APPROACH

SECOND EDITION

Java面向对象程序设计（第2版）

David M. Arnow
Scott Dexter
Gerald Weiss

著

清华大学出版社

大学计算机教育国外著名教材系列(影印版)

Introduction to Programming Using Java
An Object-Oriented Approach

Second Edition

Java 面向对象程序设计
(第2版)

David M. Arnow
Scott Dexter 著
Gerald Weiss

Brooklyn College of City University of New York

清华大学出版社
北京

北京市版权局著作权合同登记号 图字: 01-2004-5635 号

图书在版编目(CIP)数据

Java 面向对象程序设计(第 2 版) = Introduction to Programming Using Java: An Object-Oriented Approach, Second Edition/(美)阿诺(Arnow, D. M.), (美)德克斯特(Dexter, S.), (美)卫斯(Weiss, G.)著. —2 版. —影印本. —北京:清华大学出版社,2004. 10
(大学计算机教育国外著名教材系列)
ISBN 7-302-09766-6

Ⅰ. J... Ⅱ. ①阿... ②德... ③卫... Ⅲ. JAVA 语言—程序设计—高等学校—教材—英文 Ⅳ. TP312

中国版本图书馆 CIP 数据核字(2004)第 107002 号

出 版 者: 清华大学出版社 **地 址**: 北京清华大学学研大厦
http://www.tup.com.cn **邮 编**: 100084
社总机: 010-62770175 **客户服务**: 010-62776969
责任编辑: 王敏稚
印 装 者: 清华大学印刷厂
发 行 者: 新华书店总店北京发行所
开 本: 185 × 230 **印张**: 44.75
版 次: 2004 年 10 月第 1 版 2004 年 10 月第 1 次印刷
书 号: ISBN 7-302-09766-6/TP · 6743
印 数: 1 ~ 4000
定 价: 68.00 元

出 版 说 明

进入21 世纪，世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的竞争。谁拥有大量高素质的人才，谁就能在竞争中取得优势。高等教育，作为培养高素质人才的事业，必然受到高度重视。目前我国高等教育的教材更新较慢，为了加快教材的更新频率，教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从 1996 年开始，与国外著名出版公司合作，影印出版了“大学计算机教育丛书(影印版)”等一系列引进图书，受到国内读者的欢迎和支持。跨入 21 世纪，我们本着为我国高等教育教材建设服务的初衷，在已有的基础上，进一步扩大选题内容，改变图书开本尺寸，一如既往地请有关专家挑选适用于我国高等本科及研究生计算机教育的国外经典教材或著名教材，组成本套“大学计算机教育国外著名教材系列(影印版)”，以飨读者。深切期盼读者及时将使用本系列教材的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材，以利我们把“大学计算机教育国外著名教材系列(影印版)”做得更好，更适合高校师生的需要。

清华大学出版社

Preface

This book is intended as the primary text of an introductory course in programming. It assumes no programming background. The material covered is sufficient for a one- or two-semester sequence that would then be followed by a traditional CS2 (data structures) course.

To the Student

This book is an introduction to the art of computer programming in Java. It uses this popular language for a number of reasons:

- Java is an object-oriented language. Object orientation has become an essential approach of the software development community. Over the course of this text we will explain what makes a language object-oriented.
- Java is a relatively *simple* object-oriented language, at least compared to some others, such as C++. Although much of the complexity of C++ is in areas beyond the scope of our book, there are occasional pitfalls into which the beginning student can wander. Many of these *gotchas* cannot occur in Java.
- It borrows many features from other popular languages, most notably C and C++. This familiarity makes it attractive to users of those languages.
- It allows even the novice programmer to produce programs with fairly sophisticated user interfaces—that is, buttons, list boxes, scrollbars, and so on.
- It runs on many machines—PCs, Macintoshes, Sun workstations, and so on.
- It provides some fairly sophisticated facilities, including relatively easy access to networks and the Internet, making it attractive to many areas of programming.
- Programming in Java can be fun. As we pointed out above, even a relative newcomer can use the facilities provided by Java to write a program that looks nice and has sophisticated behavior.

Despite all the hoopla and fun, and despite the fact that you'll learn to use Java along the way, we have a very specific purpose: to get you to begin to think like a programmer. That means learning to analyze a problem, breaking it up into its component parts, and devising a solution. It also means practicing a lot. Programming is not learned simply from a book—you have to write lots of computer code. You won't be an expert by the end of this book, but if you pay careful attention and work at the programming exercises, you'll be on your way.

To the Instructor

In this text, we use the Java programming language to introduce students to programming. Our primary focus is on the process of developing software solutions

to problems. This process cannot be achieved in the abstract but requires a description of much of the Java language and some of its class library, as well as a discussion of a number of programming techniques and algorithms. The preface elaborates on how we achieve these goals.

A FAQ (frequently asked questions) section can be found on this book's web site, www.aw.com/cssupport. It covers many of the topics in a question-answer form.

Changes to the Second Edition

In the second edition, we have retained and sharpened our commitment to an objects-early approach. In particular, the early chapters have been significantly reworked and expanded. Chapters 2 and 3, on working with objects, incorporate additional discussion and illustration of fundamental object-oriented programming concepts; we also emphasize and exploit the benefits of viewing programming as an act of modeling. We have expanded our treatment of the class definition process (Chapters 4 and 5) to provide a more carefully paced introduction with more examples. Class design now has its own chapter (Chapter 7), also with more and richer examples. These new examples are partially supported by the earlier introduction of some imperative programming concepts (such as primitive types, assignments, and conditionals); other imperative concepts (advanced conditionals, more primitive types, and simple counting loops) are covered in Chapter 6. The discussion of traversing collections has been rewritten to use counting loops and indexing; a brief explanation of using the `Enumeration` interface as an alternative appears in Appendix D. In this way, we bring discussion of this topic into closer alignment with our coverage of loop patterns, iteration, and the `Vector` collection. We have also reorganized the later chapters somewhat, most notably by removing the chapter of examples; some parts of this chapter are now distributed across the remaining chapters.

We have rewritten much of the GUI supplement material to provide tighter connections between the supplements and the concepts introduced in the main text of the chapter. See the GUI Supplements table of contents on xxxii for a brief overview.

Finally, we have moved all section exercises to the ends of chapters and added many new exercises.

Paradigm

Any introduction to programming must take a stand on the issue of paradigm choice. Our language platform is Java, so it is not surprising that our choice is object-oriented programming (OOP). However, although it is pretty clear what procedural and functional programming entail at the CS1 level, there are a variety of competing visions of what OOP at this level signifies. We concentrate on:

- Defining and using classes
- Issues of behavior and responsibility
- Using composition, rather than inheritance

A typical problem in this text is solved by identifying a primary object in the problem, describing its behavior, and then defining a class to provide that behavior. Usually a small number (often just one) of independent subsidiary classes are defined in the solution process. The solution is completed by writing a small imperative driver, in the form of a main method, for the primary object.

In the early 1990s there was some confusion regarding the relationships between OOP, procedural programming, and imperative programming. By now, it is generally well-understood that even though OOP and procedural programming are distinct paradigms, imperative programming is equally a part of both. Certainly, as soon as assignment enters the picture, one is doing imperative programming, and sending messages that change object state may be viewed as imperative programming as well. Thus, this text teaches both OOP and imperative programming from the start. However, just as the procedures-early approach, long popular in procedural CS1 classes, introduces the mechanics and use of procedure invocation prior to imperative devices such as conditionals and loops, so here we introduce the mechanics and use of message sending and object creation before conditionals, and class definition before iteration. The rationales in both cases are identical; it is preferable to introduce the paradigm first and develop the imperative devices in that context.

In procedural programming, the way to get a task done is to find a procedure that does it and then to invoke the procedure. If no such procedure exists, the programmer has to write one. In OOP, the way to get a task done is to find or create an object of a class whose behavior includes carrying out the task and sending the object a message. If no such class exists, the programmer has to write one.

Process

Our primary focus is the process of developing software solutions to problems. To this end, we introduce informal but methodical approaches to four areas:

- Developing a class specification from a problem statement
- Implementing a class given a class specification
- Constructing loops
- Constructing recursive methods

The first two of these approaches are introduced in a rudimentary way in Chapters 4 and 5, are fleshed out in Chapter 7, and used consistently thereafter throughout the book. The other two approaches are introduced with iteration and recursion, in Chapters 10 and 14, respectively. They, too, are used consistently thereafter, wherever iteration or recursion appear.

The consistent reuse of these methodical approaches is necessary so that students realize that methodology is not just something to which one pays lip service but that it can be genuinely useful in the development of solutions to problems.

The emphasis on process means that two common fixtures of introductory texts are rarely seen in this book: dissection of code and incremental modification of

code (although the latter does appear in some extended examples and exercises). Dissection of code requires at the outset the presentation of a complete class implementation without development. It is followed by a careful analysis of the code. This approach is helpful in explaining how code works, but does not explain the process of developing code.

Many of the topics and their order in the text have been determined by our commitment to process. For example, before presenting the approach to class specification and definition, the student must be quite familiar with the idea of classes as repositories of behavior and the use of composition of classes. To that end, Chapter 3 discusses some of Java's predefined classes (including `BigInteger`, `Date`, and `GregorianCalendar`, and some of the i/o-related classes).

Language

As we elaborate on the process of program development, we introduce the features of the Java language. To prevent the discussion of the details of those features from digressing too far from the process of developing code, we often defer such discussions to special sections called Java Interludes. In these sections, we fill in the details of features introduced in the course of code development. We also use these sections to introduce language features that do not appear elsewhere but with which a CS1 student should have familiarity. Nevertheless, several features of the Java language are not covered, such as bit operations, concurrency, synchronization, and inner classes.

GUI Programming

Java's support for graphical user interface (GUI) programming is one of the reasons for its appeal in CS education—both to instructors and students. We have chosen to treat this topic in a series of GUI supplements, that is, special sections that appear at the end of each chapter. The main body of each chapter is entirely independent of these supplements. Each supplement introduces a new set of GUI tools and/or techniques in a context that reinforces the material introduced in the main body of its chapter.

The advantage of this organization is that it permits instructors to omit GUI programming altogether or introduce it at any time. It also serves to strengthen the focus of the main text on object-oriented programming rather than on language-specific features. Furthermore, it isolates the main body of the text from changes to the class library, primarily in the Abstract Window Toolkit (AWT) portion of the class hierarchy.

We have chosen to work exclusively with applets rather than applications in the GUI supplements for several reasons:

- It is easy to transform an applet into an application; the reverse is more difficult and in some cases impossible.

- The execution context of applets is more involved and therefore more worthy of a discussion in the text.
- Students love creating "cool" web pages and displaying their applets in them.

Broad coverage of the AWT is beyond the scope of this text. Our approach therefore is to address the following critical issues:

- Applet basics (Chapters 1 and 2)
- Layout—placement of components (Chapter 4)
- Event handling (Chapters 6 and 8)
- Precision in text and graphics (Chapters 5, 11, and 12)
- Threads (Chapters 11 and 14)

Along the way many useful AWT classes and methods are encountered.

Similarly, we have chosen to work almost exclusively with AWT rather than Swing (although we include a brief study of Swing as the final GUI supplement). Our reasons include the following:

- Unlike the change from JDK 1.0's event model to AWT's (JDK 1.1), the differences between AWT and Swing are primarily aesthetic, at least at the novice level.
- Swing introduces several complexities that are unnecessary for the beginner (for example the introduction of a content pane for component placement).
- The more interesting improvements only become understandable once inheritance, interfaces, and polymorphism are introduced (for example, the fact that all `JComponents` inherit from the AWT `Container` component allows placement of `JComponents`, e.g., icons, into other `JComponents`, e.g., buttons).

Flexibility

Every CS department has its own culture and its own goals for CS1. Even instructors who completely share our approach to OOP in CS1 may want to reorder or even omit some of the topics in this text. Accordingly, we have made every effort to make the introduction of many topics mutually independent. At the same time, we do want the topics in the book to build on each other. It's worth mentioning a few ways in which we resolved this tension.

Inheritance Inheritance is the subject of Chapter 12. However, an instructor who wishes to introduce this topic earlier in the course can go directly from Chapter 7 to Chapter 12 and work with the first two thirds of that chapter. The last third of the chapter addresses polymorphism and interfaces and uses material from Chapters 10 and 11.

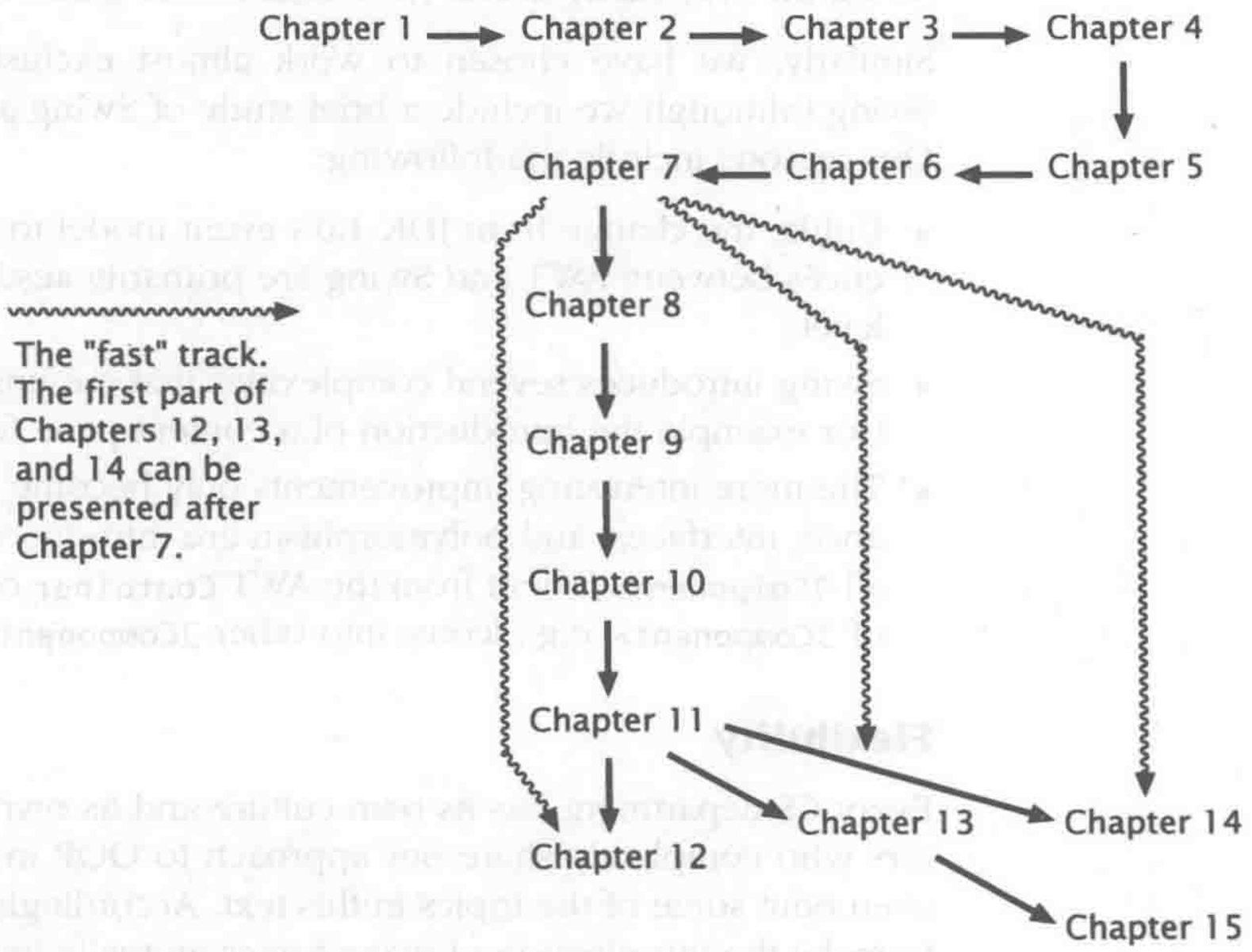
Recursion Recursion is the subject of Chapter 14. However, an instructor who wishes to introduce recursion earlier in the course can go directly from Chapter 7

to Chapter 14 and work with the first third of the chapter, which does not involve arrays and `vectors`. On the other hand, an instructor who wishes to omit the topic of recursion can do so.

Exceptions Exceptions are the subject of Chapter 13. However, the majority of the chapter is approachable directly after Chapter 7.

And, of course, the GUI supplements offer the instructors a great deal of flexibility with respect to the topic of graphic and event-driven programming. See the GUI supplement table of contents on page xxxii for a brief overview.

The dependency diagram summarizes these relationships.



Input/Output

We believe that it is important to give students thorough experience with classes, objects, constructors, composition, cascading, and the concept of a class as a model before they go about the business of writing class definitions (beginning in Chapter 4). Chapter 3 presents a rich and involved discussion of these vital topics using a variety of example classes, including some of the classes in the `java.io` package.

Chapter 3 is integral to our sequence of exposition, but for the sake of those instructors who may have alternative approaches to preparing students for Chapter 4 and

therefore wish to bypass Chapter 3, we have provided a very simple i/o package, AWIO in Appendix C, which may be used instead of Sections 3.9–3.12.

Typefaces in Code Examples

It would be useful now to identify the typefaces of the four elements that appear in code examples.

First there is the code itself:

```
class PrefaceExample {
   public static void main(String[] arg) {
         System.out.println("just an example");
   }
}
```

We often add comments to the code, notations that start with // or are surrounded by /* and */. These are intended to be notations that would actually be part of the code. The second line of the following code is an example of a comment:

```
class PrefaceExample {
   // Just print a short exemplary statement on the display.
   public static void main(String[] arg) {
      System.out.println("just an example");
   }
}
```

As we develop computer code, we will often make a notation that "holds the place" and represents code that is yet to be written. An example of such a placeholder (or pseudocode) appears in the fifth line of the following code:

```
class PrefaceExample {
   // Just print a short exemplary statement on the display.
   public static void main(String[] arg) {
      System.out.println("just an example");
      Additional output statements go here.
   }
}
```

Finally, explanatory remarks that would not normally be part of the code but serve to aid our presentation are placed in shaded screens around the code, often with arrows:

```
class PrefaceExample {
   // Just print a short exemplary statement on the display.
   public static void main(String[] arg) {
      System.out.println("just an example");
      Additional output statements go here.
   }
}
```

A comment

A class with a `main()` method

An Annotated Overview of the Chapters—The Non-GUI Parts

Chapter 1: Jumping Into Java Here we introduce the concept of programming as a means of creating models of situations. Our primary goal is to introduce the reader to ideas of classes, objects, and message passing as they are realized in Java. In addition, we present and explain an example of a program and discuss the mechanics of writing and running Java programs.

Chapter 2: Sending Messages and Performing Operations In this chapter, the focus is on the mechanics of using objects by sending them messages and all that this entails. We elaborate on the three key ideas presented in Chapter 1—classes, objects, and message passing—and introduce additional OOP and imperative essentials: methods, arguments, return values, signature, prototype, overloading, reference variables, declarations, assignment, `int`s and arithmetic, and simple conditionals. The `String` and `PrintStream` classes are used to illustrate these ideas.

Chapter 3: Working with Objects and Primitive Types The chapter begins by showing how to use a class's constructor to create objects. We continue to emphasize the theme of class as a repository for behavior. This idea is reinforced through an exploration of some of Java's predefined classes; we also explore how behaviors can be combined using cascading and composition. At the same time we introduce a few more imperative programming concepts (such as conditionals and `boolean` values) to increase the richness of our examples.

Chapter 4: Defining Classes By this time, the student is quite clear about the first principle of OOP: If anything needs to be done, find an object that can do it and send it a message. Now we show the reader how to define new classes. This definition requires quite a bit in the way of mechanics, which we begin covering in this chapter: class definition structure, method definition structure, declaration, scope and use of parameters, local variables, instance variables, and the `return` statement. Amid all these necessary language details we try to maintain a focus on the concepts of *behavior*, *interface*, and *state*. Along the way we introduce a limited version of the methodical approach to class definition that is presented in the next chapter.

Despite the embryonic character of the approach, once we get through this chapter in our courses, we breathe our first sigh of relief. At this point we can give assignments that involve the definition of new classes. We are now doing OOP.

Chapter 5: Advanced Class Definition In this chapter we cover some more mechanics of class definition: constructors, static methods, final values, and the keyword `this`.

Chapter 6: Inside the Method: Imperative Programming Now that we have discussed the major issues of class definition, we turn to the topic of writing powerful

methods. We continue with further examples of class definition, but here our focus is on imperative programming concepts (primitive data types and a wide selection of conditional statements, as well as an introduction to counting loops) that we can employ to provide the behavior our methods require.

Chapter 7: Class Design It is here that we introduce the methodical approach to class definition that we use consistently through the rest of the text. This approach starts by identifying the nouns in the problem statement. From there, a primary object is identified. This object's class is then implemented, with the implementation driving the determination of the additional classes needed in the problem. The implementation of a class entails defining its behavior and state. This straightforward "waterfall" approach is sufficient for most design problems that would be encountered by a CS1 student.

Chapter 8: Verifying Object Behavior This chapter is an overview of the need for and techniques for testing. It introduces the concept of test drivers and module testing. Additionally, it provides the reader with a starting point for the selection and construction of test cases.

Chapter 9: Working with Multiple Objects Here we introduce additional limited forms of iteration and the notions of a collection. The `while` loop is introduced and its mechanics are explained but, pending further discussion in the next chapter, its use is confined to the read/process loop pattern:

```
read
while (not eof) {
    process
    read
}
```

We also introduce a new `for` loop pattern:

```
for (i = 0; i < size of collection; i++)
    process  element number i
```

The collection we use is the `Vector` class. This provides several advantages:

- The complexity of indexing is deferred until the student can get a handle on the processing of multiple objects.
- No new syntax is required. The vector is just another object and is managed by sending messages to it.

At the conclusion of the chapter we introduce arrays and illustrate some of the similarities and differences between arrays and `Vector`s.

Despite the limitations, once we get through this chapter in our courses, we breathe a second sigh of relief. At this point we can give assignments that are much more reminiscent of "real" applications, as distinct from utility classes.

Chapter 10: Designing Iteration This is an in-depth chapter on an all-important CS1 imperative programming issue: the construction of loops. A survey of CS1 texts and courses reveals three approaches:

- The null approach—just imitate the code in the book
- Providing a set of loop patterns
- Some kind of methodical development technique, usually a watered-down formal method

In this chapter, we combine the last two of these. We present a methodical technique and then apply it to quite a few typical problems, identifying the results as loop patterns that we can refer to later.

Chapter 11: Maintaining Collections of Objects This chapter focuses on algorithms for searching and sorting. We also go beyond the usual venue of collection objects and arrays to consider searching external files. In this connection, we take the opportunity to introduce Java threads.

Chapter 12: Extending Class Behavior Our approach to inheritance is to emphasize the extending of the behavior of a superclass by a subclass. This is the way in which inheritance is most commonly used, especially by the beginning programmer—taking an existing class and adding state and behavior to produce a richer class. We do this in the context of those classes, both predefined and programmer-defined, introduced in Chapters 3–7 for the following reasons:

- The classes that we extend are already familiar to the student.
- The instructor may introduce inheritance earlier in the course, for example, immediately after Chapter 7, if it is so desired.

Extension of state, protected instance variables, overriding, and polymorphism are easily motivated in this context.

The instructor covering the GUI supplements may wish to cover at least the beginning of Chapter 12 somewhat early to give the student some appreciation of how inheritance allows even the beginner to implement complex windowed applications.

We also present an introduction to another use of inheritance: factoring out the common behavior/state of several logically related classes, producing a superclass and a class hierarchy. Again we emphasize the concept of behavior and modeling in which the various layers of the hierarchy model different abstractions of the objects.

Finally, interfaces are presented as a means of forcing a class to conform to a specified protocol.

Chapter 13: Exceptions Another theme running through the text is that of responsibility-driven programming. Classes should be responsible for as much of their behavior as possible. The other side of the coin is the idea that classes should not be responsible for behavior that is not logically theirs. We present exception handling in this light—as a way for a method to signal an exceptional, not necessarily erroneous, situation to an invoker of that method. Though some knowledge of inheritance is necessary to fully appreciate the structure of an exception hierarchy, the first part of this chapter may be covered at a relatively early stage (after Chapter 7) in order to clarify the throws clause code present in many of the methods signatures.

Chapter 14: Recursion In this treatment of recursion, we focus on its use as a programming tool. We start with extremely simple problems that some might consider inappropriately easy for recursion. These problems provide a context for developing an approach to constructing recursive solutions. We then take an obligatory detour and discuss how recursion is implemented, but we end that discussion with a stern admonition for the student to focus on how recursion is used and to ignore the implementation issue. We then move to two problems whose complexity cry out for recursion—generating permutations and the classical Towers of Hanoi problem—and end with a comparison of recursion and iteration.

When we get through this chapter in our courses we breathe a third sigh of relief. At this point, we have covered assignment, variables, expressions, numeric and logical and string types, interactive and file i/o, control structures, functions (methods), structures (classes), arrays, several algorithms, recursion, testing and debugging—all the traditional material of CS1. And of course we've done more: classes, objects, messages, plus any material from the GUI supplements that have been included. There are no more sighs of relief save the sigh when the final grades of the course are turned in to the registrar.

Chapter 15: Client/Server Computing In this chapter we provide a brief introduction to network programming using Java. We lay out some fundamental ideas about the Internet (in particular, we describe the nature of a TCP connection) and use these ideas to support the development of simple HTTP and SMTP clients.

Appendices The appendices include:

- A glossary of all the defined terms in the text (containing chapter terminology lists and terms from the GUI supplements)
- A description of how to write and run Java programs in a UNIX/Linux, Windows, or MacOS X environment
- An alternative set of classes supporting input and output that may be used in lieu of Java's predefined i/o classes.
- A brief discussion of using the `Enumeration` interface to traverse a collection.

Supplemental Materials

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

- Source code: All the completed classes and methods that appear in the text.
- JavaPlace access
- Errors: We have worked hard to avoid these and hope there are few. Here you can link to a list of errors that have been discovered.

The following instructor supplements are only available to qualified instructors. Please contact your local Addison-Wesley Sales Representative, or send e-mail to aw.csc@aw.com, for information about how to access them.

- Instructor's Manual with Solutions: This contains teaching suggestions, sample syllabi, and additional questions and problems that are suitable for homework. Also included are fully worked solutions to all exercises.
- Chapter by chapter test bank available as a Word file or in Test Gen format
- PowerPoint slides of all figures

Contact Us

We welcome questions, comments, suggestions, and corrections. Our email addresses are:

```
arnow@turingscraft.com
sdexter@brooklyn.cuny.edu
weiss@turingscraft.com
```

Acknowledgments

As everyone who reads the acknowledgments section of prefaces knows, textbooks are really the result of the collaboration and support of many people. The support we received from Addison-Wesley was first rate.

Thanks to Nathan Schultz, Lesly Hershman, and Katherine Kwack in Marketing; Patty Mahtani and Jeffrey Holcomb in Production; Joyce Cosentino Wells in Design; and Daniel Rausch and Edalin Michael at Argosy Publishing.

We are especially grateful to Galia Shokry and Maite Suarez-Rivas for their patience and encouragement.

Also essential to this effort were a host of reviewers: Michael Crowley, University of Southern California; Ralph Deters, University of Saskatchewan; Le Gruenwald, University of Oklahoma; David P. Jacobs, Clemson University; Chung Lee, California State University; Mike Litman, Western Illinois University; Yenumula B. Reddy, Grambling State University; Nan Schaller, University of Rochester; Esther Steiner,

New Mexico State University; and Shih-Ho Wang, University of California, Davis. From the first edition we thank the following reviewers: Jan Bergandy, University of Massachusetts Dartmouth; Robert H. Dependahl, Jr., Santa Barbara City College; Eileen Kraemer, Washington University in St. Louis; Ronald L. McCarty, Penn State Erie; David D. Riley, University of Wisconsin La Crosse; Jim Roberts, Carnegie Mellon University; Dale Skrien, Colby College; and Ken Slonneger, University of Iowa. These reviewers made many valuable suggestions and challenged us to refine and at times rethink our approach.

Our approach to CS1 using Java continues to mature along with that of the CS education community. Few of the ideas in this book are solely ours, but we hope we have managed to represent some of the best thinking of our fellow educators.

The customary place for acknowledgments to the family of the author is at the end. We are not ready to violate tradition, but the thanks we owe to our families for their support, encouragement, involvement, and love belongs not just at the end but at the beginning, the middle, and the end of the acknowledgments because that's where they were with us: all along, at every stage. During the course of the project, our families put up with absences, late nights, early mornings, obsessive muttering about "GUI supplements," unwashed dishes, late dinners, lots of take-out, and monopolization of the family computer, not to mention bouts of discouragement and worry. In spite of this, they gave us all the love and support one could dream of and we will never forget this. Thank you,

Barbara	Jeanne	Fern
Kera	David, Sharon, and Kathy	Yocheved
Alena	The Theoharis clan	Zvi
Joanna		Shlomo
David M. Arnow	**Scott Dexter**	**Gerald Weiss**