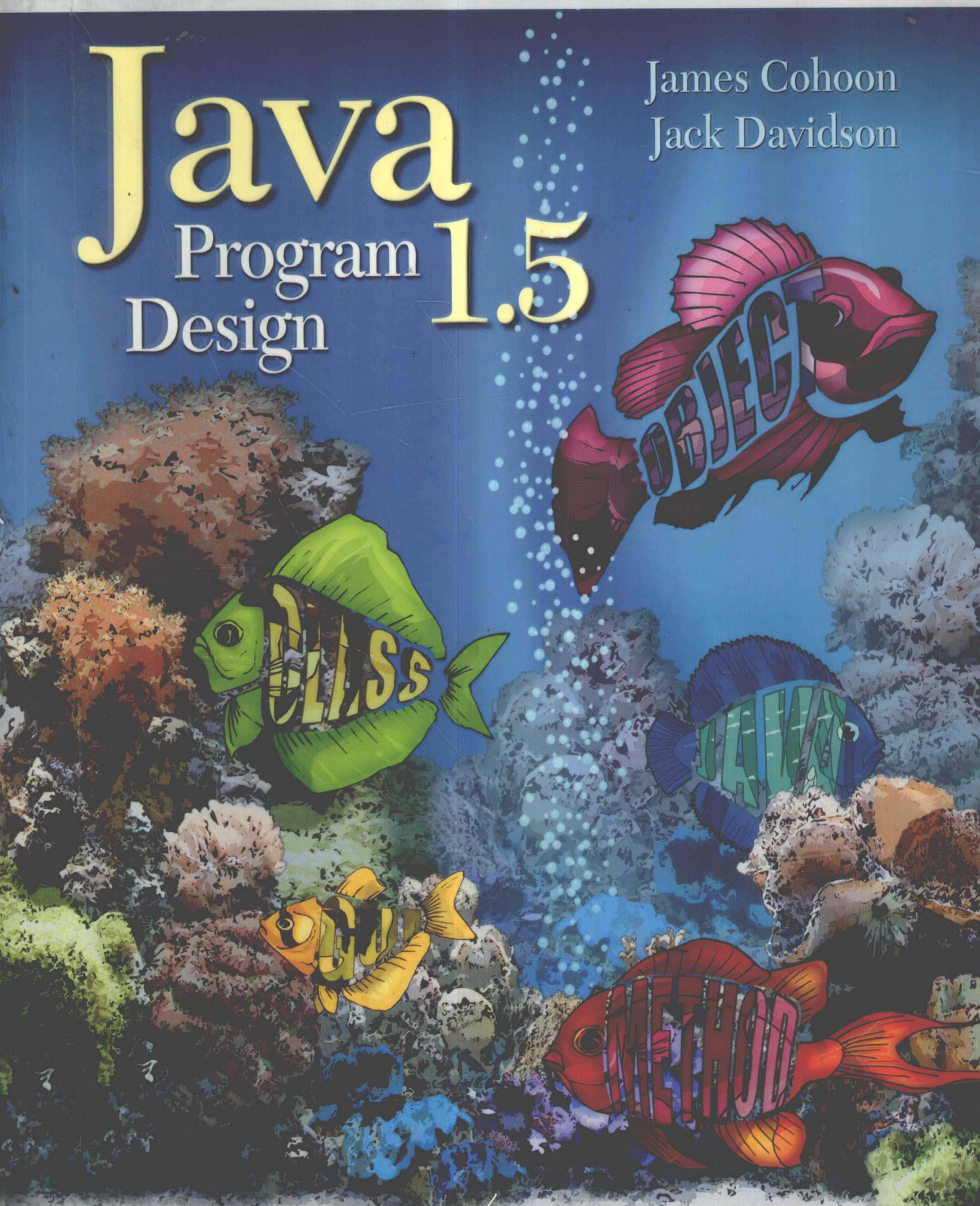
JAVA 1.5 UPDATE EDITION

Java Program Design 1.5

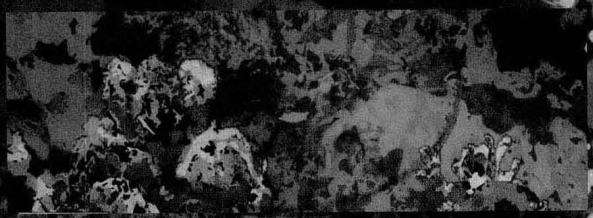
James Cohoon
Jack Davidson



Java Program Design 1.5

James P. Cohoon
University of Virginia

Jack W. Davidson



Mc Graw Hill Higher Education

Boston Burr Ridge, IL Dubuque, IA Madison, WI New York San Francisco St. Louis
Bangkok Bogotá Caracas Kuala Lumpur Lisbon London Madrid Mexico City
Milan Montreal New Delhi Santiago Seoul Singapore Sydney Taipei Toronto

The McGraw-Hill Companies

Mc Graw Hill **Higher Education**

JAVA 1.5 PROGRAM DESIGN

This book is printed on acid-free paper.

1 2 3 4 5 6 7 8 9 0 VNH/VNH 0 9 8 7 6 5 4

ISBN 0–07–304096–7

Publisher: *Elizabeth A. Jones*
Sponsoring editor: *Kelly H. Lowery*
Developmental editor: *Melinda D. Bilecki*
Marketing manager: *Dawn R. Bercier*
Senior project manager: *Kay J. Brimeyer*
Senior production supervisor: *Laura Fuller*
Lead media project manager: *Audrey A. Reiter*
Media technology producer: *Eric A. Weber*
Senior designer: *David W. Hash*
Cover/interior designer: *Rokusek Design*
Supplement producer: *Brenda A. Ernzen*
Compositor: *Interactive Composition Corporation*
Typeface: *10/13 Times Roman*
Printer: *Von Hoffmann Corporation*

Library of Congress Control Number: 2004105423

To Joanne McGrath Cohoon and Audrey Irvine

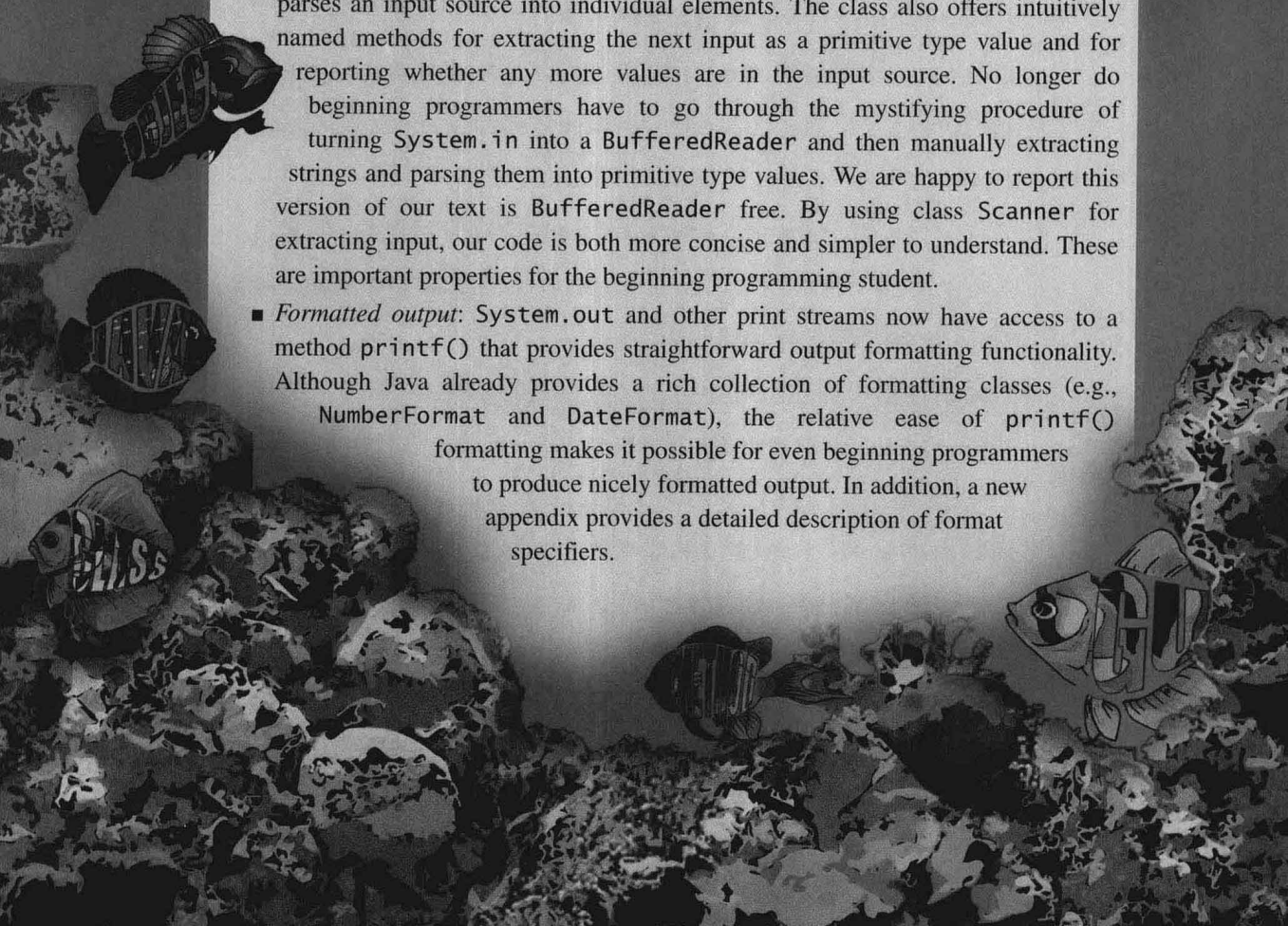
Java 1.5 edition foreword

WHAT IS DIFFERENT AND WHY

It is a great time to learn to design and develop programs in Java! The release of Java 1.5 introduced a number of components for easing development efforts. These additions are important for beginning programmers because they help make program design and development a clearer and more straightforward process.

The following list highlights some of the more important changes in Java and how our textbook makes use of them. Other changes are noted throughout the text.

- *Formatted input*: Class `Scanner` is now available. This straightforward class operates on a variety of input text and stream sources. The class automatically parses an input source into individual elements. The class also offers intuitively named methods for extracting the next input as a primitive type value and for reporting whether any more values are in the input source. No longer do beginning programmers have to go through the mystifying procedure of turning `System.in` into a `BufferedReader` and then manually extracting strings and parsing them into primitive type values. We are happy to report this version of our text is `BufferedReader` free. By using class `Scanner` for extracting input, our code is both more concise and simpler to understand. These are important properties for the beginning programming student.
- *Formatted output*: `System.out` and other print streams now have access to a method `printf()` that provides straightforward output formatting functionality. Although Java already provides a rich collection of formatting classes (e.g., `NumberFormat` and `DateFormat`), the relative ease of `printf()` formatting makes it possible for even beginning programmers to produce nicely formatted output. In addition, a new appendix provides a detailed description of format specifiers.



- *Automatic boxing and unboxing*: Java now provides automatic boxing and unboxing conversions between the primitive types and the corresponding class representations (e.g., **int** and `Integer`). Because these automatic conversions make code easier to follow, our examples make full use of them. The accompanying discussions describe the conversion process so that the reader can develop an accurate model of the translation and execution process.

- *Iterator* **for** *loop*: Java now provides an enhanced **for** loop for arrays and collections. The iterator **for** loop offers a simple syntax for the sequential access of the elements in a list. Because the iterator **for** loop ensures only valid elements are considered, experts believe it will become the preferred looping construct for list manipulation. However, programmers will also deal with existing code bases using the traditional **for** loop for list processing. Therefore, while including several demonstrations of the new loop form, our array discussion makes primary use of the traditional **for** loop. The subsequent discussion of collections uses the **for** loop form appropriate for the task at hand.

- *Generic types*: We introduce the new available generic types and their role in developing common functionality for different types of data. More importantly, the textbook introduces the *Java Collection Framework* with particular emphasis on the list data structure `ArrayList<T>` and the generic algorithms of class `Collections`. Our demonstration programs succinctly show their value to the beginning programmer.

- *Variable arity and varargs*: Print stream formatting method `printf()` makes use of the new Java capability of a method taking a variable number of parameters. Our array discussion has been updated with the development of a print method for displaying a variable number of values.

- *User interface composition*: Graphical user interface-based programming is now the dominant program form. This textbook includes two *interludes*—optional chapter-length sections—that introduce the `swing` library and event-based programming. With this new version of Java, the complexity of dealing with a graphical user interface becomes simpler. For example, the explicit acquisition of the content pane for a window is no longer necessary. The removal of such hurdles makes it easier for beginning programmers to develop their own graphical user interfaces.

Besides introducing Java 1.5, the text has undergone many other changes. Many explanations and figures have been improved and additional ones included. The appendices and index have been reorganized to make them more comprehensive and easier to use.

J. P. C

J. W. D

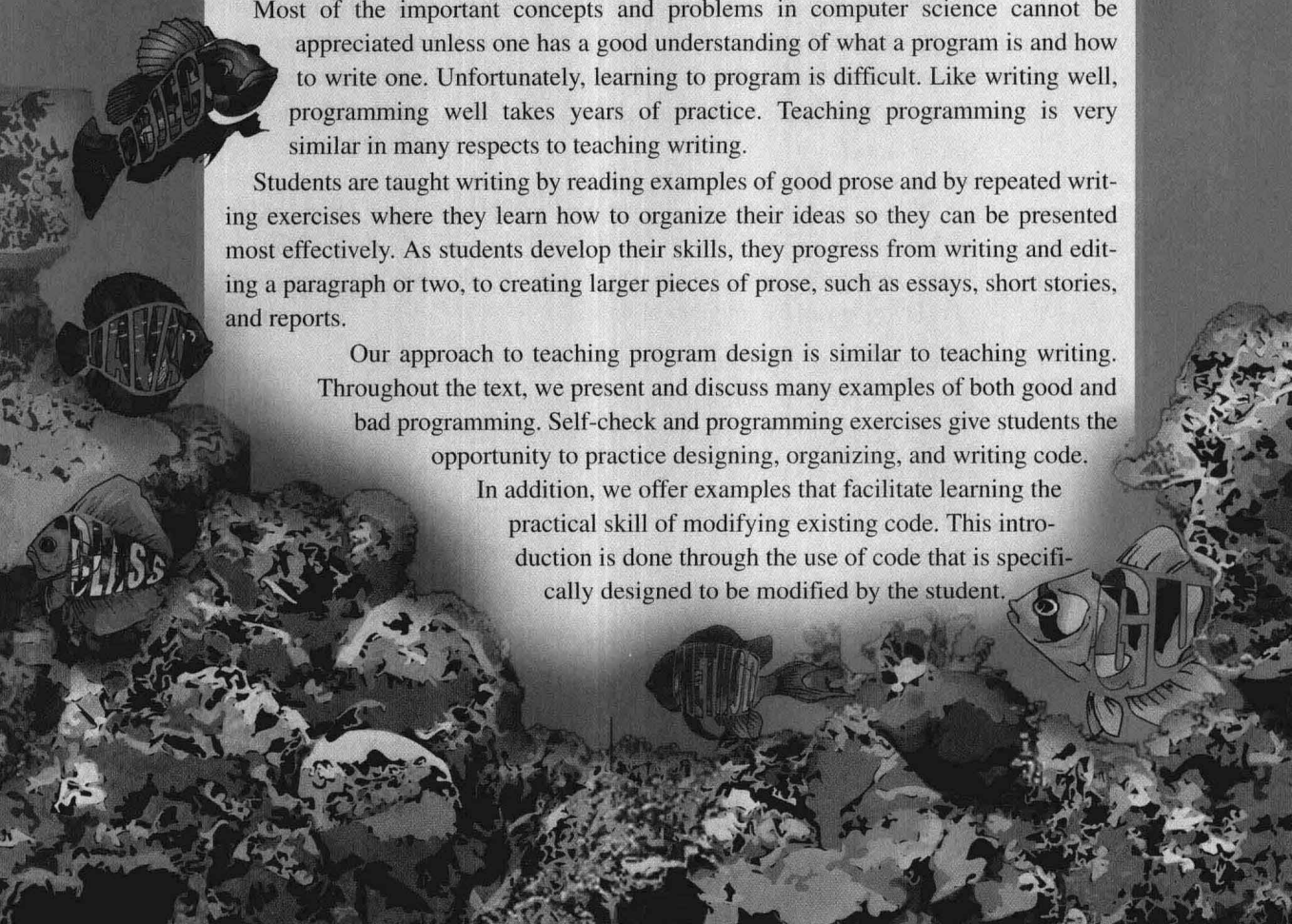
Preface

Java is now a very appropriate choice for introductory programming courses. The reasons are many. The use of the Internet continues its explosive growth. Web-ready application programs are becoming the dominant software model, and Java is *the* programming language for the Internet. Java also offers maturing software development tools, numerous packages for application programming of all types including multithreaded, advanced graphical user interfaces, and portability with its architecture-neutral design. The importance of security and robustness has taken on new meaning in recent years, and Java's support of these concerns is integral throughout its design. Being object-oriented, Java is a good pedagogical vehicle for modern software engineering and programming concepts.

Most of the important concepts and problems in computer science cannot be appreciated unless one has a good understanding of what a program is and how to write one. Unfortunately, learning to program is difficult. Like writing well, programming well takes years of practice. Teaching programming is very similar in many respects to teaching writing.

Students are taught writing by reading examples of good prose and by repeated writing exercises where they learn how to organize their ideas so they can be presented most effectively. As students develop their skills, they progress from writing and editing a paragraph or two, to creating larger pieces of prose, such as essays, short stories, and reports.

Our approach to teaching program design is similar to teaching writing. Throughout the text, we present and discuss many examples of both good and bad programming. Self-check and programming exercises give students the opportunity to practice designing, organizing, and writing code. In addition, we offer examples that facilitate learning the practical skill of modifying existing code. This introduction is done through the use of code that is specifically designed to be modified by the student.



We have found this approach to be effective because it compels students to be active participants—they must read and understand the provided code. To support this effort, the code used in this text is available electronically at our website.

GOALS OF THE TEXT

This book is targeted for a first programming course, and it has been designed to be appropriate for people from all disciplines. We assume no prior programming skills and use mathematics and science at a level appropriate to first-year college students.

The primary goals of the text are to

- Introduce students to the Java programming language;
- Present and encourage the use of the object-oriented paradigm;
- Demonstrate effective problem-solving techniques;
- Engage the student with real-world examples;
- Teach students software-engineering design concepts;
- Introduce students to Java's core and graphical libraries;
- Give students practice organizing and writing code;
- Teach students the practical skill of modifying existing code;
- Offer instructive examples of good and bad programming;
- Provide effective coverage of testing and debugging.

WHAT IS DIFFERENT AND WHY

The text provides in-depth coverage of all materials that an introductory course would need, introduces much of the remaining material generally covered in follow-on courses, and gives pointers to the rest. The breadth and the arrangement of chapters provides flexibility for the instructor in what and when topics are introduced. The chapter coverage and extensive appendices enable advanced learners to go further, and makes the book valuable as a reference source.

Some of the things that distinguish our book include the following:

- *Gentle introduction to objects*: The book implements what we call the "objects right" approach. Teaching the object-oriented paradigm in introductory courses for the last ten-plus years has shown us that Java can be successfully introduced to beginning programmers. We know that delaying user-defined classes to the end of a course inhibits the ability of students to integrate the central pillar of the object-oriented programming paradigm and forces superficial coverage of other important object-oriented programming principles. Therefore, our presentation introduces objects early—or as we prefer, right. Students begin using objects from standard packages right from the beginning. They quickly develop meaningful programs for interesting problems. Using this solid introduction, we then present the *basics* of class and object-oriented

design. After exploring control structures, we present a deeper look at methods, classes, and object-oriented design.

- *Focus on problem solving*: One of the biggest obstacles faced by many beginning students is not knowing basic problem-solving techniques. The text addresses this issue by introducing basic problem-solving skills in Chapter 1 and then applying the new concepts in each chapter to problems selected for their appeal to a variety of audiences. Students are first walked through examples that illustrate effective problem solving, and then they are given a chance to tackle similar problems on their own.
- *Introduction to software-engineering design concepts*: Software-engineering design concepts are introduced via problem studies and software projects. Besides numerous small examples, each chapter considers one or more problems in detail. As appropriate, there are object-oriented analysis and design, and algorithm development to realize the design.
- *Coverage of testing and debugging*: An important skill for programmers is how to test and debug the programs they design and implement. Chapter 13 introduces important software engineering concepts and practices with regard to testing and debugging. The chapter discusses testing techniques such as unit testing, integration testing, and code inspections. The sections on debugging focus on teaching students how to use the scientific method to find errors. The chapter also discusses common errors of beginning programmers and how to recognize them. After control structures have been introduced, this chapter material can be taught whenever an instructor deems it appropriate.
- *Engaging and inclusive examples*: Students learn from interesting situations they might encounter in real life. Diverse case studies and programming projects are drawn from topics as varied as physical fitness, spam, medical diagnosis, statistical analysis, psychological typing, data visualization, graphs, entertainment, and animation. By offering this variety of examples, the text demonstrates how programmers can participate and contribute to our daily lives.
- *Exclusive use of standard Java classes*: The text uses only standard Java classes in introducing Java programming concepts. In particular, there are no author-written classes for acquiring input. Instead standard classes and techniques are presented in a way that makes sense to beginning programmers.
- *Lab manual*: A printed lab manual accompanies the text for schools that use laboratories in their introductory courses. The lab material offers a hands-on experience that reinforces Java programming concepts and skills.
- *Programming and style tips*: Besides explaining Java and object-oriented programming, the text also provides advice on how to be a better and more knowledgeable programmer and designer. There are important tips on such topics as avoiding common programming errors, writing readable code, and following software engineering practices.
- *Self-test, exercises, and software projects*: Every chapter provides a self-test exercise section with answers to enable students to evaluate their skills on important concepts. The text also provide several hundred exercises whose solutions are available to

instructors through the publisher. Once the basics of Java are introduced in Chapter 2, that chapter and all successive chapters supply a programming project that exercises chapter concepts.

- *Reference appendices*: Appendices D and E provide nearly two hundred pages of description of the standard Java APIs. The coverage makes the text a handy reference manual well after the course completes.

CHAPTER OVERVIEW AND FEATURES

Introduction

Each chapter begins with a brief introduction designed to focus the student's attention and prepare them for the material to be covered. We emphasize both the immediate significance of the topic as well as its place in the broader programming context.

Objectives

A list of chapter objectives follows the introduction and provides a set of specific learning goals for the student. This list enables students to measure their progress as they move through chapter material and lets them evaluate their level of comprehension at the end. It also serves as a guide for instructors to use when preparing tests and quizzes.

Icons and aside boxes

Icons and shaded boxes indicate warnings, style tips, advanced material, and information pertaining to the Java language itself.

Indicates a warning about programming. Often these are tips on how to avoid common programming errors.

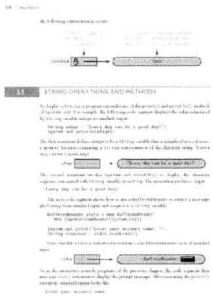
Indicates that the associated material is related to programming style.

Indicates that the associated material is concerned with the Java programming language itself.

Indicates programming tips or material that presents a more detailed discussion or a sidebar to the current topic.

Full-color design

An inviting, full-color design highlights related material, indicates section breaks, calls out special features, and makes key information easy to reference.



(a) Chapter text

(b) Case study

(c) End of chapter material

(d) Reference appendix

Code formatting

Specially formatted code listings make sections of code easy to find and reference. Each complete listing is numbered according to its place in the chapter, separated from related material by coloring and line numbering. Partial sections of code are clearly set off from surrounding text and are generously annotated with easy-to-spot author comments.

UML diagrams

The use of UML diagrams helps clarify relationships between classes while at the same time familiarizing students with this widely used system of notation.

Case studies

The chapters provide multiple case studies that are designed to teach effective problem-solving skills and to reinforce object-oriented programming and software engineering design concepts. The specific learning objective is highlighted at the beginning of each case study, and problem-solving steps are highlighted with special icons. The case studies

are colored to distinguish them from other chapter material. Coverage of the case studies is optional; they apply chapter concepts rather than introducing new concepts.

End-of-chapter reviews

Each chapter ends with a thorough, point-by-point summary of the chapter's major ideas. The end of chapter materials are color highlighted for easy access.

Self-tests

Each chapter includes a self-test with answers supplied at the end of the chapter. These self-check sections are designed to help students evaluate whether they have mastered the chapter objectives and to reinforce the key ideas of the chapter.

Programming projects

Except for Chapter 1, which provides background material, each chapter has at least one interesting programming case study presented in a manner that makes it suitable for use as a class assignment. Programming case studies include determining your exercise training zone; harvesting e-mail addresses; medical diagnosis; automobile loan calculator; and an aquarium simulation.

Exercises

An exercise section at the end of each chapter offers a variety of problems requiring a range of efforts levels.

CHAPTER SUMMARIES

- *Chapter 1: Background*—computer organization; software; software engineering principles; object-oriented software development; problem solving.
- *Chapter 2: Java basics*—program organization; method `main()`; commenting and whitespace; classes, keywords, identifiers, and naming conventions; methods; program execution; SDK; constants; variables; operations; primitive types; operators; precedence; interactive programs; `Scanner`; primitive variable assignment.
- *Chapter 3: Using objects*—`String`; reference variables; **`null`**; inserting, extracting, and concatenating strings; reference assignment; `String` methods.
- *Chapter 4: Being classy*—introduces user-defined classes; instance variables; constructors; instance methods; inspectors; mutators; facilitators; simple graphics.
- *Chapter 5: Decisions*—boolean algebra and truth tables; logical expressions; **`boolean`** type; Boolean equality and ordering operators; testing floating-point values for equality; operator precedence; short-circuit evaluation; **`if`** statement; **`if-else`** statement; string and character testing; sorting; **`switch`** statement.
- *Chapter 6: Iteration*—**`while`** statement; simple string and character processing; file processing; **`for`** statement; index variable scope; **`do-while`** statement.
- *Graphics Interlude: GUI–based programming*—graphical user interfaces; `swing`; `awt`; and event-based programming

- *Chapter 7: Programming with methods and classes*—parameter passing; invocation and flow of control; class variables; scope; local scope; name reuse; overloading; overriding; `equals()`; `toString()`; `clone()`; generics.
- *Chapter 8: Arrays and collections*—one-dimensional arrays; definitions; element access and manipulation; explicit initialization; constant arrays; members; array processing; methods; program parameters; vararg; sorting; searching; multidimensional arrays; matrices; generics; collections framework; `ArrayList<T>`; collections algorithms.
- *Chapter 9: Inheritance and polymorphism*—object-oriented design; reuse; base class; derived class; single inheritance; **`super`**; is-a, has-a, and uses-a relationships; controlling inheritance; default, **`protected`**, and **`private`** members; polymorphism; abstract base class; **`interface`** hierarchies.
- *Graphics Interlude: GUI-based programming*—case studies in the design and implementation of graphical user interfaces for personality typing and the smiley guessing game.
- *Chapter 10: Exceptions*—abnormal event; exceptions; throwing; trying; catching; exception handlers; **`finally`**; stream specialization.
- *Chapter 11: Recursive problem solving*—recursive functions, sorting, searching, visualization.
- *Chapter 12: Threads*—multiple independent flows of control; processes; threads; scheduling and repeating threads; `Timer`; `TimerTask`; `Thread`; `Date`; `Calendar`; `JOptionPane`; sleeping; animation; systems software.
- *Chapter 13: Testing and debugging*—software development; code reviews; black-box and white-box testing; inspections; test harness; statement coverage; unit, integration testing, and system testing; regression testing; boundary conditions; path coverage; debugging.
- *Appendix A: Tables and operators*—Unicode character set; reserved words; operators and precedence.
- *Appendix B: Number representation*—binary numbers; decimal numbers; two's compliment; conversions.
- *Appendix C: Formatted I/O*—`Scanner`; `printf()`.
- *Appendix D: Applets*—applet programming.
- *Appendix E: Standard Java packages*—`java.applet`; `java.awt`; `java.io`; `java.lang`; `java.math`; `java.net`; `javax.swing`; `java. text`; and `java.util`.

GRAPHICS INTERLUDES

From personal observations and from conversations and communications with colleagues, we recognize that not all introductory programming courses are able to introduce graphical user interfaces (GUIs). The time may not be available to introduce the `swing`

API and event-driven programming. Therefore, we have coalesced this material into two *Graphical Interludes*, and their coverage is optional. However, for instructors who want to stress this material, the GUI coverage can be introduced after the class concepts of Chapter 4 are presented.

We do distinguish between graphical user interfaces and creating graphical images. The Java standard APIs make it quite simple to display rectangles, lines, circles, ovals, triangles, and polygons. Their display is almost as easy as displaying text to a console window. Examples in other chapters make independent use of these Java features. These examples are also for the most part optional. However, it is our experience that students enjoy creating graphical imagery and that the concepts of object-oriented programming are easier to understand when examples have a visual nature.

USING THIS BOOK

The text continues to have more material than can be covered in a single course. The extra coverage was deliberate—it enables instructors to select their topics on programming and software development. The book has been designed for teaching flexibility. For example, if instructors desire to delay the introduction of classes, they first can cover most of the control structure materials (Sections 5.1–5.9 and Sections 6.1–6.5). Similarly, if an instructor desires to introduce arrays before classes, the fundamental array material (Sections 8.1–8.4 and Section 8.8) can proceed the discussion of classes. Also except for the example in Section 9.2, the discussion of inheritance can precede the coverage of arrays.

The testing and debugging material of Chapter 13 can be covered anytime after classes and arrays have been introduced.

We use the following layout for our introductory course.

Week	Topic	Readings
1	Computing and object-oriented design	Chapter 1
2	Programming fundamentals	Chapter 2
3	Object manipulation	Chapter 3 (Sections 3.1–3.5)
4–5	Class basics	Chapter 4
5	Conditional statements	Chapter 5 (Sections 5.1–5.7, 5.10)
6–7	Iteration statements	Chapter 6 (Sections 6.1–6.5)
8	Graphical user interfaces	Graphics Interludes: 1 and 2
9–10	Classes	Chapter 7
11-12	Arrays and lists	Chapter 8
13–14	Inheritance and polymorphism	Chapter 9

SUPPLEMENTARY MATERIALS

The publisher website at `www.javaprogramprogramdesign.com` offers the source code and data files for all listings in the text. Other materials include a complete set of slides, which are available in PowerPoint and PDF formats, and introductions to the vari-

ous Java programming IDEs. Other educational supplements are available at our class web site `http://www.cs.virginia.edu/javaprogramdesign`.

THE AUTHORS

Jim Cohoon is a professor in the computer science department at the University of Virginia and is a former member of the technical staff at AT&T Bell Laboratories. He joined the faculty after receiving his Ph.D. from the University of Minnesota. He has been nominated twice by his department for the university's best-teaching award. In 1994, Professor Cohoon was awarded a Fulbright Fellowship to Germany, where he lectured on object-oriented programming and software engineering. Professor Cohoon's research interests include algorithms, computer-aided design of electronic systems, optimization strategies, and computer science education. He is the author of more than 70 papers in these fields. He is a member of the Association of Computing Machinery (ACM), the ACM Special Interest Group on Design Automation (SIGDA), the ACM Special Interest Group on Computer Science Education (SIGCSE), the Institute of Electrical and Electronics Engineers (IEEE), and the IEEE Circuits and Systems Society. He is a member of ACM Council, SIG Governing Board Executive Committee, former member of ACM Publications Board, and is past chair of SIGDA. He can be reached at `cohoon@virginia.edu`. His Web home page is `http://www.cs.virginia.edu/cohoon`.

Jack Davidson is also a professor in the computer science department at the University of Virginia. He joined the faculty after receiving his Ph.D. from the University of Arizona. Professor Davidson has received NCR's Faculty Innovation Award for innovation in teaching. Professor Davidson's research interests include compilers, computer architecture, systems software, and computer science education. He is the author of more than 100 papers in these fields. He is a member of the ACM, the ACM Special Interest Group on Programming Languages (SIGPLAN), the ACM Special Interest Group on Computer Architecture (SIGARCH), SIGCSE, the IEEE, and the IEEE Computer Society. He served as an associate editor of *Transactions on Programming Languages and Systems*, ACM's flagship journal on programming languages and systems, from 1994 to 2000. He was chair of the 1998 Programming Language Design and Implementation Conference (PLDI '98) and program co-chair of the 2000 SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES 2000). He can be reached at `jwd@virginia.edu`. His Web home page is `http://www.cs.virginia.edu/~jwd`.

DELVING FURTHER

The following texts are primary references on the Java language.

- Ken Arnold, James Gosling, and David Holmes, *The Java Programming Language*, Third Edition, Addison-Wesley; June 2000.
- Bill Joy (Editor), Guy Steele, James Gosling, and Gilad Bracha, *The Java Language Specification,* Second Edition, Addison-Wesley, June 2000.

The following texts are good sources on the standard libraries and more-advanced object-oriented design, and program development.

- David M. Geary, *Graphic Java 1.2, Mastering the JFC: AWT,* Volume 1, Prentice Hall, September 1998.
- David M. Geary, *Graphic Java 2,* Volume 2, *Swing*, Prentice Hall, March 1999.
- Joshua Engel, *Programming for the Java Virtual Machine*, Addison-Wesley, June 1999.
- Cay S. Horstmann and Gary Cornell, *Core Java 2,* Volume I, *Fundamentals*, Prentice Hall, August 2002.
- Cay S. Horstmann and Gary Cornell, *Core Java 2:* Volume II, *Advanced Features*, Prentice Hall, December 2001.
- Matthew Robinson and Pavel A. Vorobiev, *Swing*, Manning Publications Company; February 2003.
- Stephen A. Stelting and Olav Maassen, *Applied Java Patterns*, Prentice Hall; December 2001.
- Sun Microsystems, *Java Look and Feel Design Guidelines: Advanced Topics,* Addison Wesley Professional; March 2001.
- Al Vermeulen (Editor), Scott W. Ambler, Greg Bumgardner, Eldon Metz, Alan Vermeulen, Trevor Misfeldt, Jim Shur, and Patrick Thompson, *The Elements of Java Style*, Cambridge University Press; January 2000.
- John Zukowski, *Java Collections*, APress; April 2001.

ACKNOWLEDGMENTS

We thank the University of Virginia for providing an environment that made this book possible. In particular, we thank Jack Stankovic for his tireless efforts in leading the computer science department to national prominence. We also thank Jenna Cohoon, Joanne Cohoon, John Knight, and Tom Horton for their comments. We thank Hannah Cohoon for her fish artwork and JJ Cohoon for his icon artwork.

We thank all of the people at McGraw-Hill for their efforts in making this edition a reality. In particular, we thank Betsy Jones, for her support and encouragement; Kay Brimeyer, for her behind-the-scenes product-management skills; David Hash, for leading the art and cover-design team; and Pat Steele, for copyediting. Special thanks go to Kelly Lowery, our editor, for support, direction, and focus throughout this project; Melinda Bilecki, our developmental editor, for managing and synthesizing the reviewing process; Mary Cahall for her organizational ability; and Dawn Bercier for her creative marketing ideas.

We thank the following testers, readers, and reviewers for their valuable comments and suggestions on the text and associated materials:

A. Arokiasamy, Multimedia University of Malaysia

David Aspinall, University of Edinburgh

Ivan Bajic, San Diego State University

Dwight Barnett, Virginia Tech
Vivekram Bellur, University of Virginia
David Bethelmy, Embry-Riddle Aeronautical University
Robert Biddle, Victoria University of Wellington
Elizabeth Boese, Colorado State University
Gene Boggess, Mississippi State University
Mike Buckley, University at Buffalo
Robert Burton, Brigham Young University
Judith Challinger, California State University, Chico
Errol Chopping, Charles Sturt University
Ilyas Cicekli, University of Central Florida
Charles Daly, Dublin City University
J. Greggory Dobbins, University of South Carolina
Neveen Elnahal, University of Virginia
Stephen Fickas, University of Oregon
Jeffrey Forbes, Duke University
Gerald Gordon, DePaul University
Heng Aik Koan, National University of Singapore
Michael Huhns, University of South Carolina
Norm Jacobson, University of California, Irvine
Cerian Jones, Montana Tech
Katherine Kane, University of Virginia
Cathy Key, University of Texas, San Antonio
Abigail Knight, Tandem School
Barry Lawson, University of Richmond
Susan Lindsay, University of Virginia
Evelyn Lulis, DePaul University
Lauren Malone, University of Virginia
Stephanie Kim Marvin, University of Virginia
Blayne Mayfield, Oklahoma State University
Jim McElroy, California State University, Chico
Daniel McCracken, City College of New York
Hugh McGuire, University of California, Santa Barbara
Christoph Mlinarchik, University of Virginia
Keitha Murray, Iona College
Faye Navabi, Arizona State University
Richard Pattis, Carnegie Mellon
Hal Perkins, University of Washington
Pete Petersen, Texas A&M University
Roger Priebe, University of Texas
Vera Proulx, Northeastern University
Graham Roberts, Flinders University
Roy Ruhling, University of Virginia

Celia Schahczenski, Montana Tech
Carolyn Schauble, Colorado State University
Carol Scheftic, California Polytechnic State University, San Luis Obispo
Jesse Barrack Schofield, University of Virginia
John Scott, Massachusetts Bay Community College
Eric Schwabe, DePaul University
Mike Scott, University of Texas
Barbara Ann Sherman, University of Buffalo
Barry Soroka, California Polytechnic State University, Pomona
David Vineyard, Kettering University

We thank our spouses, Audrey and Joanne, and our children for their efforts, cooperation, and sacrifices in making this book happen.

Finally, we thank the users of this book. We welcome your comments, suggestions, and ideas for improving this material. Please write in care of the publisher, McGraw-Hill, or send electronic mail to `cohoon@virginia.edu` or `jwd@virginia.edu`.

J. P. C
J. W. D