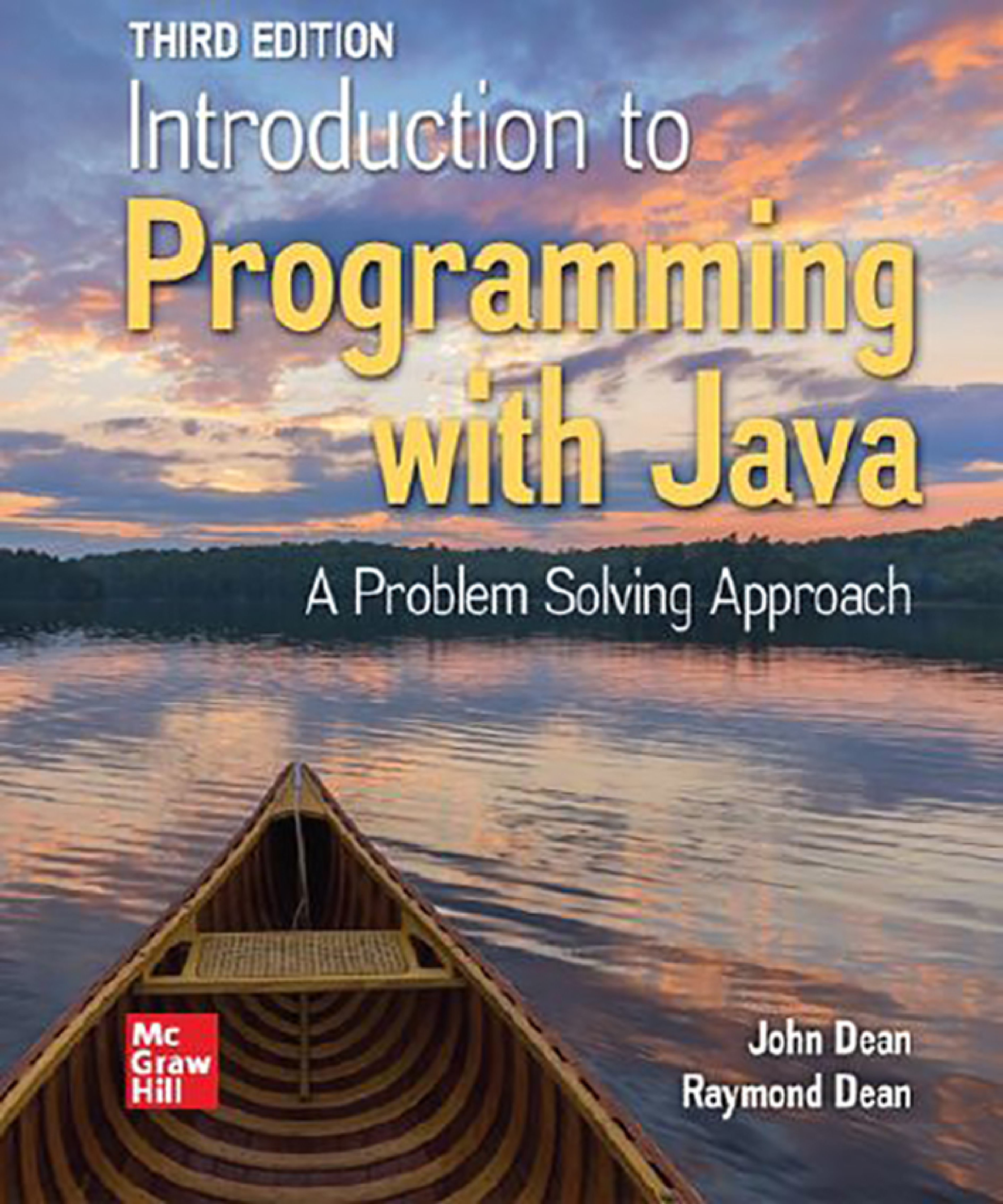

THIRD EDITION

Introduction to Programming with Java

A Problem Solving Approach

McGraw Hill

John Dean
Raymond Dean

Introduction to Programming WITH JAVA

A Problem Solving Approach

John Dean
Park University

Raymond Dean
University of Kansas

Mc
Graw
Hill

INTRODUCTION TO PROGRAMMING WITH JAVA

Published by McGraw Hill LLC, 1325 Avenue of the Americas, New York, NY 10121.

Some ancillaries, including electronic and print components, may not be available to customers outside the United States.

This book is printed on acid-free paper.

1 2 3 4 5 6 7 8 9 LCR 24 23 22 21 20

ISBN 978-1-26057524-8
MHID 1-260-57524-1

Cover Image: ©*Shutterstock/Brian Lasenby*

All credits appearing on page or at the end of the book are considered to be an extension of the copyright page.

The Internet addresses listed in the text were accurate at the time of publication. The inclusion of a website does not indicate an endorsement by the authors or McGraw Hill LLC, and McGraw Hill LLC does not guarantee the accuracy of the information presented at these sites.

mheducation.com/highered

edication

—To Stan and Kate

About the Authors

John Dean is an Associate Professor in the Computer Science and Information Systems Department at Park University. He earned a Ph.D. degree in computer science from Nova Southeastern University and an M.S. degree in computer science from the University of Kansas. He is Java certified and has worked in industry as a software engineer and project manager, specializing in Java and various web technologies—JavaScript, JavaServer Pages, and servlets. He has taught a full range of computer science courses, including Java programming and Java-based web programming. He has authored a web programming textbook with a focus on client-side technologies HTML5, CSS, and JavaScript.

Raymond Dean is a Professor Emeritus, Electrical Engineering and Computer Science, University of Kansas. He earned an M.S. degree from MIT and a Ph.D. degree from Princeton University. As a professional engineer in the HVAC industry, he wrote computer programs that design air distribution systems and analyze energy consumption and sound propagation in buildings. At the University of Kansas, he taught microprocessor programming, data structures, and other courses in electrical engineering and computer science.

Contents

CHAPTER 1

Introduction to Computers and Programming 1

CHAPTER 2

Algorithms and Design 32

CHAPTER 3

CHAPTER 4

CHAPTER 5

CHAPTER 6

Object-Oriented Programming 222

CHAPTER 7

Object-Oriented Programming— Additional Details 272

CHAPTER 8

Software Engineering 324

CHAPTER 9

Arrays 384

CHAPTER 10

ArrayLists and an Introduction to the Java Collections Framework 435

CHAPTER 11

Recursion 489

CHAPTER 12

CHAPTER 14

Inheritance and Polymorphism 637

CHAPTER 15

Exception Handling 691

CHAPTER 16

Files, Buffers, Channels, and Paths 735

CHAPTER 17

GUI Programming Basics 787

CHAPTER 18

Appendices

Additional Online Material

Mc
Graw
Hill
Education
5011111

Preface

In this book, we lead you on a journey into the fun and exciting world of computer programming. Throughout your journey, we'll provide you with lots of problem-solving practice. After all, good programmers need to be good problem solvers. We'll show you how to implement your problem solutions with Java programs. We provide a plethora of examples, some short and focused on a single concept, some longer and more "real world." We present the material in a conversational, easy-to-follow manner aimed at making your journey a pleasant one. When you're done with the book, you should be a proficient Java programmer.

Our textbook targets a wide range of readers. Primarily, it targets students in a standard college-level "Introduction to Programming" course or course sequence where no prerequisite programming experience is assumed. We have included the topics recommended by the College Board for high school students studying for advanced placement (AP) in computer science. So this text should be good for those students as well.

In addition to targeting students with no prerequisite programming experience, our textbook targets industry practitioners and college-level students who have some programming experience and want to learn Java. This second set of readers can skip the early chapters on general programming concepts and focus on the features of Java that differ from the languages that they already know. In particular, because C++ and Java are similar, readers with a C++ background should be able to cover the textbook in a single three-credit-hour course. (But we should reiterate for those of you with no programming experience: No prerequisite programming experience is required in order to use this text.)

Finally, our textbook targets those who are learning Java on their own, outside of a classroom environment. This third set of readers should read the entire textbook at a pace determined on a case-by-case basis.

What's New in This Edition?

The changes in this edition are big and small. Big changes include new chapters, reorganized chapter sections, new programming constructs, new program examples, and new exercises. Smaller changes include updating explanations and anecdotes. We've combed the entire book for opportunities to improve the book's clarity and readability. The following list highlights the more significant changes that we've made to this edition.

- **Introductory Chapter**

  To keep up with the computer industry's growth, we've made quite a few changes to

Chapter 1, such as updating the information in the computer hardware and Java history sections.

- **Switching Constructs**

  Java 12 and Java 13 made improvements to the venerable switch statement, and this edition describes those improvements. We use the new switching techniques (multiple comma-separated case constants and no break statements) for programs throughout the book. And we use switch statements versus switch expressions according to what the problem calls for. If you’re a fan of the old-style switch statement, no worries, we provide a description early on which will help you with legacy code.

- **Local Variable Type Inferencing**

  Java 10 introduced the ability to use var as a type (rather than int, double, etc.) for a local variable declaration where the declaration is part of an initialization. We describe the new syntax, but for self-documentation reasons, we stick with traditional explicit type declarations for the most part.

- **Name Change for Static Variables and Static Methods**

  The powers that be (the Oracle documentation folks) now use the terms *static variable* and *static method* for what used to be known as class variable and class method, so we’ve updated accordingly.

- **Miscellaneous Java API Library Updates**

  With the new Java releases since the second edition, there have been quite a few updates to the Java API library. We’ve updated our discussions and programs with new API method and constructor calls when appropriate. Most of our new API content can be found in our GUI coverage, but there are other API changes sprinkled throughout the book. For example, with Java’s deprecation of the wrapper class constructors, we’ve refactored our programs to rely on the wrapper classes’ valueOf methods.

- **New Section—**forEach **Method and Streams**

  We introduce the forEach method as a simple alternative to the for-each loop in the context of an ArrayList. We then use the forEach method in the context of streams, where it really shines. We describe streams in depth, with their exciting potential to take advantage of parallel processing to improve a program’s efficiency.

- **Lambda Expressions and Method References**

  Lambda expressions and method references are techniques that allow you to implement the functionality of a method so you can use it as an argument in a method call. We first present lambda expressions and method references as arguments for a forEach method call. Later, we use lambda expressions and method references extensively to help with the GUI programs.

- **Interfaces with Static Methods and Default Methods**

We've rewritten Chapter 14's section on interfaces to include a discussion of static methods and default methods. Oracle added them to interfaces because they support an interface's ability to implement multiple inheritance effectively.

- **End-of-Chapter GUI Sections**

  We've rewritten all of our end-of-chapter GUI sections to take advantage of Java's newer GUI constructs.

- **Three New Chapters—JavaFX**

  In this book's second edition, we used the AWT and Swing platforms for our two GUI chapters. This third edition moves those chapters to the book's website. We provide three new chapters in the main body of the book that describe GUI programming using the JavaFX platform. As part of that presentation, you'll learn how to format your programs using JavaFX CSS properties.

- **New Appendix—Modules**

  In Appendix 4, we introduce modules, which allow you to group together packages. Modules make it easier to organize and share classes for different programming needs. They are used to facilitate the configuration of Java software for diverse hardware and software platforms.

- **New Exercises**

  We have substantially changed most of the exercises and altered almost all of them in some way. As before, we provide exercise solutions on the password-protected instructor's portion of the book's website.

Compliant with the College Board's AP Computer Science A Curriculum

We have put a great deal of effort into ensuring that this textbook is compliant with the College Board's Advanced Placement (AP) Computer Science A curriculum content. It follows all the AP Computer Science A guidelines. As such, it appears on the College Board's approved textbook list at https://apcentral.collegeboard.org/courses/ap-computer-science-a/course-audit.

Textbook Cornerstone #1: Problem Solving

Being able to solve problems is a critical skill that all programmers must possess. We teach programmatic problem solving by emphasizing two of its key elements—algorithm development and program design.

Emphasis on Algorithm Development

In Chapter 2, we immerse readers into algorithm development by using pseudocode for the algorithm examples instead of Java. In using pseudocode, students are able to work through nontrivial problems on their own without getting bogged down in Java syntax—no need to worry about class headings, semicolons, braces, and so on.[1] Working through nontrivial problems enables students to gain an early appreciation for creativity, logic, and organization. Without that appreciation, Java students tend to learn Java syntax with a rote-memory attitude. But with that appreciation, students tend to learn Java syntax more quickly and effectively because they have a motivational basis for learning it. In addition, they are able to handle nontrivial Java homework assignments fairly early because they have prior experience with similarly nontrivial pseudocode homework assignments.

In Chapter 3 and in later chapters, we rely primarily on Java for algorithm-development examples. But for the more involved problems, we sometimes use high-level pseudocode to describe first-cut proposed solutions. Using pseudocode enables readers to bypass syntax details and focus on the algorithm portion of the solution.

Emphasis on Program Design

Problem solving is more than just developing an algorithm. It also involves figuring out the best implementation for the algorithm. That’s program design. Program design is extremely important, and that’s why we spend so much time on it. Frequently, we explain the thought processes that a person might go through when coming up with a solution. For example, we explain how to choose between different loop types, how to split up a method into multiple methods, how to decide on appropriate classes, how to choose between instance and static members, and how to determine class relationships using inheritance and composition. We challenge students to find the most elegant implementations for a particular task.

We devote a whole chapter to program design—Chapter 8, “Software Engineering.” In that chapter, we provide an in-depth look at coding-style conventions and documentation for programmers and users. We discuss design strategies like separation of concerns, modularization, and encapsulation. Also in the chapter, we describe alternative design strategies—top-down, bottom-up, case-based, and iterative enhancement.

Problem-Solving Sections

We often address problem solving (algorithm development and program design) in the natural flow of explaining concepts. But we also cover problem solving in sections that are wholly devoted to it. In each problem-solving section, we present a situation that contains an unresolved problem. In coming up with a solution for the problem, we try to mimic the real-world problem-solving experience by using an iterative design strategy. We present a first-cut solution, analyze the solution, and then discuss possible improvements to it. We use a conversational trial-and-error format (e.g., “What type of layout manager should we use? We first tried the GridLayout manager. That works OK, but not great. Let’s now try the BorderLayout manager.”). This casual tone sets the student at ease by conveying the message that it is normal, and in fact expected, that a programmer will need to work through a problem multiple times before finding the best solution.

Additional Problem-Solving Mechanisms

We include problem-solving examples and problem-solving advice throughout the text (not just in Chapter 2, Chapter 8, and the problem-solving sections). As a point of emphasis, we insert a problem-solving box, with an icon and a succinct tip, next to the text that contains the problem-solving example and/or advice.

We are strong believers in learning by example. As such, our textbook contains a multitude of complete program examples. Readers are encouraged to use our programs as recipes for solving similar programs on their own.

Textbook Cornerstone #2: Fundamentals First

Postpone Concepts That Require Complex Syntax

We feel that many introductory programming textbooks jump too quickly into concepts that require complex syntax. In using complex syntax early, students get in the habit of entering code without fully understanding it or, worse yet, copying and pasting from example code without fully understanding the example code. That can lead to less-than-ideal programs and students who are limited in their ability to solve a wide variety of problems. Thus, we prefer to postpone concepts that require complex syntax. We prefer to introduce such concepts later on, when students are better able to understand them fully.

As a prime example of that philosophy, we cover the simpler forms of GUI programming early (in an optional graphics track), but we cover the more complicated forms of GUI programming later in the book. Specifically, we postpone event-driven GUI programming until the end of the book. This is different from some other Java textbooks, which favor early full immersion into event-driven GUI programming. We feel that strategy is a mistake because proper event-driven GUI programming requires a great deal of programming maturity. When they learn it at the end of the book, our readers are better able to understand it fully.

Tracing Examples

To write code effectively, it’s imperative to understand code thoroughly. We’ve found that step-by-step tracing of program code is an effective way to ensure thorough understanding. Thus, in the earlier parts of the textbook, when we introduce a new programming structure, we often illustrate it with a meticulous trace. The detailed tracing technique we use illustrates the thought process programmers employ while debugging. It’s a printed alternative to the sequence of screen displays generated by debuggers in integrated development environment (IDE) software.

Input and Output

In the optional GUI-track sections and in the GUI chapters at the end of the book, we use GUI commands for input and output (I/O). But because of our emphasis on fundamentals, we use console commands for I/O for the rest of the book.[2] For console input, we use the Scanner class. For console output, we use the standard System.out.print, System.out.println, and System.out.printf methods.

Textbook Cornerstone #3: Real World

More often than not, today's classroom students and industry practitioners prefer to learn with a hands-on, real-world approach. To meet this need, our textbook and its associated website include:

- compiler tools
- complete program examples
- practical guidance in program design
- coding-style guidelines based on industry standards
- Unified Modeling Language (UML) notation for class relationship diagrams
- practical homework-project assignments

Compiler Tools

We do not tie the textbook to any particular compiler tool—you are free to use any compiler tool(s) that you like. If you do not have a preferred compiler in mind, then you might want to try out one or more of these:

- Java Standard Edition Development Kit (JDK), by Oracle
- TextPad, by Helios
- Eclipse, by the Eclipse Foundation
- Netbeans, backed by Oracle
- BlueJ, by the University of Kent and Deaken University

To obtain the above compilers, visit our textbook website at http://www.mhhe.com/dean3e, find the appropriate compiler link(s), and download away for free.

Complete Program Examples

In addition to providing code fragments to illustrate specific concepts, our textbook contains lots of complete program examples. With complete programs, students are able to (1) see how the analyzed code ties in with the rest of a program, and (2) test the code by running it.

Coding-Style Conventions

We include coding-style tips throughout the textbook. The coding-style tips are based on Oracle’s coding conventions (https://www.oracle.com/technetwork/java/codeconvtoc-136057.html), Google’s coding conventions (https://google.github.io/styleguide/javaguide.html), and industry practice. In Appendix 5, we provide a complete reference for the book’s coding-style conventions and an associated example program that illustrates these conventions.

UML Notation

UML has become a standard for describing the entities in large software projects. Rather than overwhelm beginning programmers with syntax for the entire UML (which is quite extensive), we present a subset of UML. Throughout the textbook, we incorporate UML notation to represent classes and class relationships pictorially. For those interested in more details, we provide additional UML notation in Appendix 7.

Homework Problems

We provide homework problems that are illustrative, practical, and clearly worded. The problems range from easy to challenging. They are grouped into three categories—review questions, exercises, and projects. We include review questions and exercises at the end of each chapter, and we provide projects on our textbook’s website.

The review questions tend to have short answers, and the answers are in the textbook. The review questions use these formats: short-answer, multiple-choice, true/false, fill-in-the-blank, tracing, debugging, and write a code fragment. Each review question is based on a relatively small part of the chapter.

The exercises tend to have short to moderate-length answers, and the answers are not in the textbook. The exercises use these formats: short-answer, tracing, debugging, and write a code fragment. Exercises are keyed to the highest prerequisite section number in the chapter, but they sometimes integrate concepts from several parts of the chapter. For this third edition, we have changed almost all of the end-of-chapter exercises, including exercises associated with unchanged material in the body of the text.

The projects consist of problem descriptions whose solutions are complete programs. Project solutions are not in the textbook. Projects require students to employ creativity and problem-solving skills and apply what they’ve learned in the chapter. These projects often include optional parts, which provide challenges for the more talented students. Projects are keyed to the highest prerequisite section number in the chapter, but they often integrate concepts from several preceding parts of the chapter. For this third edition, we have modified old projects and added new projects to make all projects conform to content in the body of the current text. Because the most substantial body-of-text changes are in the final three chapters, most of the project modifications and additions are associated with these chapters.

An important special feature of this book is the way that it specifies problems. “Sample sessions” show the precise output generated for a particular set of input values. These sample sessions include inputs that represent typical situations and sometimes also extreme or boundary situations.

Academic-Area Projects

To enhance the appeal of projects and to show how the current chapter's programming techniques might apply to different areas of interest, we take project content from several academic areas:

- computer science and numerical methods
- business and accounting
- social sciences and statistics
- math and physics
- engineering and architecture
- biology and ecology

Most of the academic-area projects do not require prerequisite knowledge in a particular area. Thus, instructors are free to assign almost any of the projects to any of their students. To provide a general reader with enough specialized knowledge to work a problem in a particular academic area, we sometimes expand the problem statement to explain a few special concepts in that academic area.

Most of the academic-area projects do not require students to have completed projects from earlier chapters; that is, most projects do not build on previous projects. Thus, for the most part, instructors are free to assign projects without worrying about prerequisite projects. In some cases, a project repeats a previous chapter's project with a different approach. The teacher may elect to take advantage of this repetition to dramatize the availability of alternatives, but this is not necessary.

Project assignments can be tailored to fit readers' needs. For example:

- For readers outside of academia—

  Readers can choose projects that match their interests.

- When a course has students from one academic area—

  Instructors can assign projects from the relevant academic area.

- When a course has students with diverse backgrounds—

  Instructors can ask students to choose projects from their own academic areas, or instructors can ignore the academic-area delineations and simply assign projects that are most appealing.

To help you decide which projects to work on, we've included a "Project Summary" section after the preface. It lists all the projects by chapter and section, and for each project, it specifies:

- prerequisite chapter and section
- academic area
- estimated difficulty
- a title and brief description

After using the “Project Summary” section to get an idea of which projects you might like to work on, see the textbook’s website for the full project descriptions.

Organization

In writing this book, we lead readers through three important programming methodologies: structured programming, OOP, and event-driven programming. For our structured programming coverage, we introduce basic concepts such as variables and operators, if statements, and loops. Then we show readers how to call prebuilt methods from Oracle’s Java API library. Many of these methods, like those in the Math class, are non-OOP methods that can be called directly. Others, like those in the String class, are OOP methods that must be called by a previously created object. After an “interlude” that gives readers a brief taste of what it’s like to write methods in a non-OOP environment, we move into OOP programming, and introduce basic OOP concepts such as classes, objects, instance variables, instance methods, and constructors. We also introduce static variables and static methods, which are useful in certain situations. However, we note that they should be used less often than instance variables and instance methods. Next, we move on to more advanced OOP concepts—arrays, collections, interfaces, and inheritance. Chapters on exception handling and files provide a transition into event-driven GUI programming. We describe and employ event-driven GUI programming in the final three chapters.

The content and sequence we promote enable students to develop their skills from a solid foundation of programming fundamentals. To foster this fundamentals-first approach, our book starts with a minimum set of concepts and details. It then gradually broadens concepts and adds detail later. We avoid overloading early chapters by deferring certain less-important details to later chapters.

Page xvii

GUI Track

Many programmers find Graphical User Interface (GUI) programming to be fun. As such, GUI programming can be a great motivational tool for keeping readers interested and engaged. That’s why we include graphics sections throughout the book, starting in Chapter 1. We call those sections our “GUI track.” Most of these end-of-chapter sections use GUI code that complements the current chapter’s previously presented non-GUI material. For readers who do not have time for the GUI track, no problem. Any or all of the GUI track sections may be skipped because the rest of the book does not depend on any of the GUI-track material.

Although the rest of the book does not depend on the GUI-track material, be aware that some of the GUI-track sections depend on some of the material in prior GUI-track sections:

- Chapter 3's GUI section introduces dialog boxes for user input, and dialog boxes are used in later GUI sections for Chapters 10, 11, 15, and 16.
- Chapters 8, 10, and 16 have GUI sections that implement a common program, with iterative enhancements in each new GUI section.
- Chapters 12 and 13 have GUI sections that implement a common GridWorld program (for readers interested in the College Board's AP Computer Science A curriculum). The GridWorld code uses AWT and Swing GUI software.

Chapter 1

In Chapter 1, we first explain basic computer terms—what are the hardware components, what is source code, what is object code, and so on. We then narrow our focus and describe the programming language we'll be using for the remainder of the book—Java. Finally, we give students a quick view of the classic bare-bones "Hello World" program. We explain how to create and run the program using minimalist software—Microsoft's Notepad text editor and Oracle's command-line JDK tools.

Chapter 2

In Chapter 2, we present problem-solving techniques with an emphasis on algorithmic design. In implementing algorithm solutions, we use generic tools—flowcharts and pseudocode—with pseudocode given greater weight. As part of our algorithm-design explanation, we describe structured programming techniques. In order to give students an appreciation for semantic details, we show how to trace algorithms.

Chapters 3–5

We present structured programming techniques using Java in Chapters 3–5. Chapter 3 describes sequential programming basics—variables, input/output, assignment statements, and simple method calls. Chapter 4 describes nonsequential program flow—if statements, switch constructs, and loops. In Chapter 5, we explain methods in more detail and show readers how to use prebuilt methods in the Java API library. In all three chapters, we teach algorithm design by solving problems and writing programs with the newly introduced Java syntax.

Interlude

This "mini-chapter" contains two programs that show how to write multiple methods without using OOP. The Interlude presents a fork in the road between two study sequences. For the standard study sequence, read the chapters in the standard order (Chapters 1 through 19). For the "objects later" study sequence, after reading Chapter 5, read the supplemental chapters S6 and S9 online before returning to Chapter 6, where you'll begin your study of OOP in earnest.

Chapters 6–7

Chapter 6 introduces the basic elements of OOP in Java. This includes implementing classes and implementing methods and variables within those classes. We use UML class diagrams and object-oriented tracing techniques to illustrate these concepts.

Chapter 7 provides additional OOP details. It explains how reference variables are assigned, tested for equality, and passed as arguments to a method. It explains overloaded methods and constructors. It also explains the use of static variables, static methods, and different types of named constants.

Chapter 8

While the art of program design and the science of computerized problem-solving are developed throughout the textbook, in Chapter 8, we focus on these aspects in the context of OOP. This chapter begins with an organized treatment of programming style. It introduces javadoc, the Java application that automatically generates documentation for user-programmers. It describes ways to communicate with users who are not programmers. It describes organizational strategies like separation of concerns, modularization, encapsulation, and provision of general-purpose utilities. Coded examples show how to implement these strategies. It describes the major programming paradigms—top-down design, bottom-up design, using pre-written software for low-level modules, and prototyping.

Chapters 9–10

Chapter 9 describes arrays, including arrays of primitives, arrays of objects, and multidimensional arrays. It illustrates array use with complete programs that sort, search, and construct histograms. Chapter 10 describes Java’s powerful array alternative, ArrayList. This provides a simple example of generic-element specification. It also introduces the Java Collections Framework, which in turn, provides natural illustrations of Java interfaces. The prewritten classes in the Java Collections Framework provide a simple introduction of sets, maps, and queues. A relatively short but complete program shows how the pre-written Java implementations of these data structures can be used to create and traverse a multiconnected random network.

Chapter 11

Chapter 11 describes another way to process a collection of data—recursion. This chapter includes a discussion of various recursive strategies. It introduces recursion with a real-life example and a familiar problem that one can solve easily with either looping or recursion. Then it moves gradually to problems that are harder to solve with looping and more easily solved with recursion. Although this chapter appears after the chapter on ArrayLists and the Java Collections Framework, it does not depend on these concepts—it uses just ordinary arrays.

Chapter 12

Early on, students need to be immersed in problem-solving activities. Covering too much syntax detail early can detract from that objective. Thus, we initially gloss over some less-important syntax details and come back to those details later in Chapter 12. This chapter provides more details on items such as these:

Page xix

- byte and short primitive types
- Unicode character set
- type promotions
- postfix versus prefix modes for the increment and decrement operators
- conditional operator
- short-circuit evaluation
- enum data type
- forEach method
- lambda expressions
- method references
- streams

The chapter ends with a friendly introduction to a relatively large public-domain program called GridWorld, which the College Board has used for many years as part of its AP Computer Science A course of study. This gives students a glimpse of how larger programs are organized.

Chapters 13–14

Chapters 13 and 14 describe class relationships in depth with numerous examples. Chapter 13 describes aggregation, composition, and inheritance. Chapter 14 describes advanced inheritance-related details such as the Object class, polymorphism, abstract classes, and the finer points of interfaces. An optional section at the end of Chapter 13 describes an extension of the GridWorld environment introduced in Chapter 12 and provides additional exposure to Java’s legacy AWT and Swing graphics. Exercises in Chapters 13 and 14 relate material in these two chapters to corresponding GridWorld features.

Chapters 15−16

Chapter 15 describes exception handling, and Chapter 16 describes files. We present exception handling before files because file-handling code requires the use of exception handling. For example, to open a file one must check for an exception. In addition to simple text I/O, our treatment of files includes buffering, random access, channeling, and memory mapping.

Chapters 17−19

As in the end-of-chapter GUI sections, Chapters 17−19 present GUI concepts using the JavaFX platform. But the programming strategies differ. What follows are the strategies used in Chapters 17–19 (which are different from the strategies used in the end-of-chapter GUI sections). For user input, the programs use the components TextField, TextArea, Button, RadioButton, CheckBox, ComboBox, ScrollPane, and Menu. For layout, the programs use the containers FlowPane, VBox, HBox, GridPane, BorderPane, TilePane, and TextFlow. For formatting, the programs use JavaFX CSS properties.

Depending on how much and what kind of GUI techniques you’re interested in, you can study one or more of the end-of-chapter GUI sections or skip all of them. That won’t affect your ability to grasp what’s in Chapters 17−19. If you’re short on time, you can omit all of the book’s GUI material without compromising your understanding of other material in the book.

Chapters S17−S18

Chapters S17 and S18 (the *S*’s stand for supplemental) are posted online. They describe the older Java GUI platforms—AWT and Swing. The trend has been for new Java programs to use JavaFX, and not AWT and Swing. However, there’s quite a bit of AWT and Swing code currently in production, and that means there’s still a need for Java programmers to understand the older techniques so they can update and improve existing code. So if you find yourself in that position, Chapters S17 and S18 are a good starting point for learning what you need to know.

Appendices

Most of the appendices cover reference material, like the ASCII character set and the operator precedence table. For this third edition, we have updated Appendix 4 by including a detailed description of Java modules.

Subject-Matter Dependencies and Sequence-Changing Opportunities

We’ve positioned the textbook’s material in a natural order for someone who wants fundamentals and also wants an early introduction to OOP. We feel that our order is the most efficient and effective one for learning how to become a proficient OOP programmer. Nonetheless, we realize that different readers have different content-ordering preferences. To accommodate those different preferences, we’ve provided some built-in flexibility. Figure 0.1 illustrates that flexibility by showing chapter dependencies and, more importantly, chapter -nondependencies. For example, the arrow between Chapter 3 and Chapter 4 means that Chapter 3 must be read prior to Chapter 4. Because there are no arrows going out of Chapters 1, 11, and 16 that point to other complete chapters, you may skip those chapters without losing prerequisite material that later chapters need. We use rectangles with rounded corners to indicate chapter sections that you may want to read in advance. If you choose that option, you'll want to return to the normal chapter sequence after completing the advanced sections.

Here are some sequence-changing opportunities revealed by Figure 0.1:

- Readers can skip Chapter 1, "Introduction to Computers and Programming."
- For an earlier introduction to OOP, readers can read the OOP overview section in Chapter 6 after reading Chapter 1.
- Readers can learn OOP syntax and semantics in Chapter 6 after finishing Java basics in Chapter 3.
- For additional looping practice, readers can learn about arrays in Chapter 9 after finishing loops in Chapter 4.
- Readers can skip Chapter 11, "Recursion," and Chapter 16, "Files."
- Readers who prefer a late objects approach can postpone reading Chapter 6, "Object-Oriented Programming," by first reading Chapter S6, "Writing Methods in a Non-Object-Oriented Environment," Sections 9.1–9.6 (which explain the basics of arrays), and Chapter S9, "Arrays in a Non-Object- Oriented Environment."
- For GUI programming, readers who prefer the Swing platform should read Chapters S17 and S18.

To support content-ordering flexibility, the book contains "hyperlinks." A hyperlink is an optional jump forward from one place in the book to another place. The jumps are legal in terms of prerequisite knowledge, meaning that the jumped-over (skipped) material is unnecessary for an understanding of the later material. We supply hyperlinks for each of the nonsequential arrows in Figure 0.1. For example, we supply hyperlinks that go from Chapter 1 to Chapter 6 and from Chapter 3 to Chapter 12. For each hyperlink tail end (in the earlier chapter), we tell the reader where they may optionally jump to. For each hyperlink target end (in the later chapter), we provide an icon at the side of the target text that helps readers find the place where they are to begin reading.