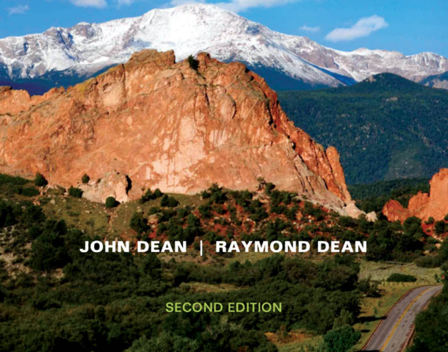
Introduction to Programming WITH JAVA

A Problem Solving Approach



JOHN DEAN | RAYMOND DEAN

SECOND EDITION

Introduction to Programming WITH JAVA

A Problem Solving Approach

John Dean
Park University

Raymond Dean
University of Kansas

Mc Graw Hill
Connect Learn Succeed™

The McGraw·Hill Companies

Mc Graw Hill Connect Learn Succeed™

INTRODUCTION TO PROGRAMMING WITH JAVA: A PROBLEM SOLVING APPROACH:
SECOND EDITION

This book is printed on acid-free paper.

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

ISBN 978–0–07–337606–6
MHID 0–07–337606–x

Vice President, General Manager: *Marty Lange*
Editorial Director: *Michael Lange*
Publisher: *Raghothaman Srinivasan*
Marketing Manager: *Curt Reynolds*
Development Editor: *Katie Neubauer*
Project Manager: *Melissa M. Leick*
Buyer: *Sandy Ludovissy*
Media Project Manager: *Prashanthi Nadipalli*
Cover Design: *Studio Montage, St. Louis, MO*
Cover image: *Chris Johnson*
Compositor: *S4Carlisle Publishing Services*
Typeface: *10/12 Times Roman*
Printer: *R. R. Donnelley Crawfordsville, IN*

Figure 1.1b: © PhotoDisc/Getty Images; Figure 1.1c: © BigStock Photos; Figure 1.1d: © PhotoDisc/Getty Images; Figure 1.1e: © BrandX/Punchstock; Figure 1.1f: © Ryan McVay/Getty Images; Figure 1.2a: © BigStock Photos; Figure 1.2b: © BigStock Photos; Figure 1.2c: © BigStock Photos; Figure 1.4a: © BigStock Photos; Figure 1.4b: © Getty Royalty Free; Figure 1.4c: © Oleksiy Mark | Dreamstime.com; Figure 1.4d: © BigStock Photos; Figure ta2.1: Courtesy of the Naval Surface Warfare Center, Dahlgren, VA, 1988. US Naval History and Heritage Command Photograph; Figure ta4.2: © Sergey Mostovoy | Dreamstime.com; Figure 5.13: Photo courtesy of the authors; Figure 5.16: Photo courtesy of the authors; Figure 11.1: Courtesy of Rafael de Sant'Anna Neri; Figure 14.5a: © Michael Pettigrew | Dreamstime.com

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

Library of Congress Cataloging-in-Publication Data

Dean, John, 1962-
Introduction to programming Java : with a problem solving approach / John S. Dean, Raymond H. Dean.
p. cm.
Includes bibliographical references and index.
ISBN 978–0–07–337606–6 (alk. paper) ISBN 0–07–337606–X (alk. paper) 1. Java (Computer program language) I. Dean, Ray, 1936- II. Title.

QA76.73.J38D4264 2012
005.2'762—dc23 2012036870

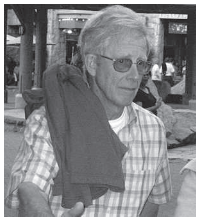
www.mhhe.com

Dedication

—To Jordan and Caiden

About the Authors

John Dean is the Department Chair of the Computer Science and Mathematics Department at Park University. He earned an M.S. degree in computer science from the University of Kansas. He is Sun 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.



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, and he is a senior member of IEEE. He has published numerous scientific papers and has 21 U.S. patents. He has industry experience in manufacturing HVAC equipment and energy-management controls, as well as in system energy analysis. At the University of Kansas, he taught a wide range of courses in electrical engineering and computer science.

Contents

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 tried to include all the topics recommended by the College Board for students studying for advanced placement (AP) in computer science. So this text should be good for these 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 let us reiterate for those of you with no programming experience: You should be fine. No prerequisite programming experience is required.)

Finally, our textbook targets high school students and readers outside of academia with no programming experience. 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, and new programs. Smaller changes include updating descriptions, anecdotes, examples, exercises, and projects. We've combed the entire book for opportunities to improve the book's clarity and readability. The following list highlights the more significant changes we've made for this edition.

- **Language Enhancements**

  We wrote the first edition when the standard Java compiler was Java 6. Back then, we described most of Java 6's language features—but not all. With this edition, we present additional language features that were previously omitted, including the `assert` keyword, enumerated types (`enum`s), and variable-length argument lists (varargs).

  We describe several Java language enhancements that were introduced in Java 7—numeric literals with underscores, binary literals, `switch` statements with strings, and type inferences with the diamond operator.

  We present closures, a language feature introduced in Java 8, as an appendix on the book's website. A *closure* is a block of code that can be passed to a method and executed later. Among other things, closures are intended to clean up the messiness of anonymous inner classes.

- **Introductory Chapter**

  To keep up with the computer industry's continued growth, we've made many changes to Chapter 1, such as updating the information in the computer hardware and Java history sections. In addition to the many updates in the preexisting sections, we've added a new short section on computer ethics.

- **Accommodation for Late Objects Advocates**

  Some of our first edition readers suggested that before launching into a full discussion of object-oriented programming (OOP) in Chapter 6, we should describe multiple-method programs in a non-OOP environment. To accommodate that suggestion, we include a short Interlude "mini-chapter" named "Multiple-Method Programs in a Non-Object-Oriented Environment" between Chapters 5 and 6. We anticipate that most readers will benefit from and be satisfied by that brief discussion. However, we realize that some readers will want more. Some of the readers of our first edition advocated for a "late objects" approach, with complete details on multiple-method programs and complete details on arrays, all before starting OOP in Chapter 6. To support that approach, we provide those details in two supplemental chapters on the book's website. The Interlude encourages late objects advocates to read those supplemental chapters before returning to Chapter 6.

- **Reduced Emphasis on Class Variables and Class Methods**

  Based on feedback from readers of the first edition, we've eliminated the "Classes with Class Members" chapter and moved a rewritten subset of that material to the end of Chapter 7.

- **Software Engineering**

  To provide more emphasis on documentation, we've moved a subset of the javadoc material forward, from the Javadoc appendix up to Chapter 8. In Chapter 8, we introduce the notion of pre- and post-conditions and include pre- and post-condition comments in code examples.

- **New Chapter—`ArrayLists` and an Introduction to the Java Collections Framework**

  We've written a new chapter that describes various classes and interfaces in the Java Collections Framework, with an emphasis on the `ArrayList` class, the `List` interface, queues, and stacks. Within this context, we describe Java generics and multiclass interfaces. The chapter provides an easily accessible glimpse of data structures and their applications.

- **New Chapter—Recursion**

  We've written a new chapter on recursion that combines the content from the first edition's "Recursion" appendix with new content. The chapter, Chapter 11, explains how to analyze and write recursive methods, and it presents a significant number of complete recursive programs, including a merge sort program and a graphical user interface (GUI) fractal program that draws an animated picture of a grove of growing trees.

- **Performance Analysis**

  In our chapters on arrays, collections, and recursion, we've added some performance analysis discussion to our presentation of various algorithms and data structures. For the sequential search, binary search, merge sort, and several other types of algorithms, we include brief discussions of how execution times depend on the number of items being processed. For Java's `ArrayList`, `LinkedList`, and `ArrayDeque` classes, we compare measured execution times for various operations. In a separate section at the end of Chapter 11, we introduce the Big O notation and use it to quantify the performance of various algorithms.

- **New Multiple-Section Case Study**

  In sections at the ends of Chapters 12 and 13, we describe and provide enhancements to the GridWorld program, a legacy case study from the College Board's Advanced Placement (AP) Computer Science curriculum. The GridWorld program implements simple animation. It shows bugs crawling around, bumping into things, changing direction as a result, eating plants or other critters, planting new flowers, changing color based on events or passage of time, and so on. We provide keyed exercises to put GridWorld features in the context of our book's presentation.

- **Rewritten Chapter—Exception Handling**

  We've rewritten about half of Chapter 15, the exception handling chapter, taking advantage of new Java 7 exception handling constructs—automatic resource management within a `try` block heading and multiple parameters within a `catch` block heading.

- **Rewritten Chapter—File Handling**

  We've rewritten almost the entire file handling chapter, Chapter 16, taking advantage of NIO.2, which is Java 7's new file system application programming interface (API). The chapter puts greater emphasis on file handling that relates to Internet communication by including alternate character sets, alternate input/output (IO) options, and random access to buffered and channeled IO. Additionally, the chapter includes memory mapping and file system traversal techniques.

  For readers who want to use files early, we've introduced in Chapter 3 an optional "quick and dirty" file input technique that appends a `throws Exception` clause to the main method.

- **Better Visuals and More GUI**

  We describe the Nimbus look and feel and explain how users can configure their computers to use it. Oracle rolled out Nimbus in a later release of Java 6 and started recommending it with the advent of Java 7. We use the Nimbus look and feel for all window screenshots throughout the book.

  Also, we have added a description of image decoration that uses semitransparent layers.

- **New Supplemental GUI Chapters Using JavaFX**

  With the release of Java 8, JavaFX replaced Swing as Oracle's preferred GUI toolkit. Thus, we've written two supplemental GUI chapters, S17 and S18, found on the book's website, that are dedicated to describing JavaFX. Because Swing will remain popular for quite some time, we explain and use Swing in the early optional GUI tracks and in Chapters 17 and 18. Our parallel descriptions of Swing and JavaFX give readers a chance to see how the old compares with the new.

- **New Appendix—Number Systems and Conversions Between Them**

  We've written a new appendix that describes how numbers are represented in different bases—decimal, binary, octal, and hexadecimal. The appendix provides algorithms for converting between the number bases.

Compliant with the College Board's AP Computer Science Curriculum

As noted by teachers using the first edition in their AP Computer Science courses, the first edition covered most of the College Board's AP Computer Science curriculum content. We have put a great deal of effort into ensuring that this second edition follows all the AP Computer Science guidelines, as put forth by the College Board.

This second edition has been reviewed by a former member of the AP Computer Science Development Committee, Judy Hromcik. She states that at the time of its printing, this edition is fully compliant with the College Board's AP Computer Science curriculum.

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 non-trivial 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 non-trivial 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 non-trivial Java homework assignments fairly early because they have prior experience with similarly non-trivial 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 class 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.

[1]Inevitably, we use a particular style for our pseudocode, but we repeatedly emphasize that other pseudocode styles are fine so long as they convey the intended meaning. Our pseudocode style is a combination of free-form description for high-level tasks and more specific commands for low-level tasks. We've chosen a pseudocode style that is intuitive, to welcome new programmers, and structured, to accommodate program logic.

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/dean, 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 (http://www.oracle.com/technetwork/java/codeconv-138413.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.

[2]We introduce GUI I/O early on with the `JOptionPane` class. That opens up an optional door for GUI fans. If readers are so inclined, they can use `JOptionPane` to implement all our programs with GUI I/O rather than console I/O. To do so, they replace all console I/O method calls with `JOptionPane` method calls.

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.

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.

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

The academic-area projects do not require prerequisite knowledge in a particular area. Thus, instructors are free to assign 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, the projects do not build on each other. Thus, 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 for each project, it specifies:

- the associated section within the chapter
- the academic area
- the length and difficulty
- a 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 class variables and class 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 event-driven GUI programming in the final two 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.

GUI Track

Many programmers find 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." For readers who do not have time for the GUI track, no problem. Any or all of the GUI track sections may be skipped as they cover material that is independent of later material.

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 non-sequential program flow—`if` statements, `switch` statements, 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 a program that shows 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 18). For the "objects later" study sequence, after reading Chapter 5, read the supplemental chapters S6 and S9 on the book's website 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 class variables and methods and 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 the other 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:

- the `byte` and `short` primitive types
- the Unicode character set
- type promotions
- postfix versus prefix modes for the increment and decrement operators
- the conditional operator
- short-circuit evaluation
- the `enum` data type

The chapter ends with a friendly introduction to a relatively large public-domain program called GridWorld, which the College Board used for many years as part of its recommended course of study for advanced placement in computer science. This gives students a glimpse of how larger programs are organized.

Chapters 13–14

We describe class relationships in Chapters 13 and 14. We spend two full chapters on class relationships because the subject matter is so important. We take the time to explain class relationship details in depth and provide numerous examples. Chapter 13 describes aggregation, composition, and inheritance. Chapter 14 describes more advanced inheritance-related details such as the `Object` class, polymorphism, `abstract` classes, and the finer points of interfaces. A section at the end of Chapter 13 extends our discussion of the GridWorld program. And numerous exercises in these two chapters relate chapter material to corresponding GridWorld features.

Chapters 15–16

We cover exception handling in Chapter 15 and files in Chapter 16. We present exception handling before files because file-handling code utilizes exception handling; for example, opening a file requires that you check for an exception. Our treatment of exception handling includes multiple-`catch` parameters and try-with-resources. In addition to simple text I/O, our treatment of files includes buffering, random access, channeling, and memory mapping.

Chapters 17–18

We describe event-driven GUI programming at the end of the book in Chapters 17 and 18. By learning event-driven GUI programming late, students are better able to grasp its inherent complexities.

Chapters S17–S18

Chapters 17 and 18 present GUI concepts using the Swing toolkit because Swing is very popular and will remain so for years to come. However, with the advent of Java 8, Oracle started supporting JavaFX as the default GUI toolkit. In Chapters S17 and S18 (the *S*'s stands for supplemental), we present GUI concepts using the JavaFX toolkit.

Appendices

Most of the appendices cover reference material, such as the ASCII character set and the operator precedence table. But the last two appendices introduce advanced Java material—multithreading and closures.

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 non-dependencies. 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.
- They can learn OOP syntax and semantics in Chapter 6 after finishing Java basics in Chapter 3.

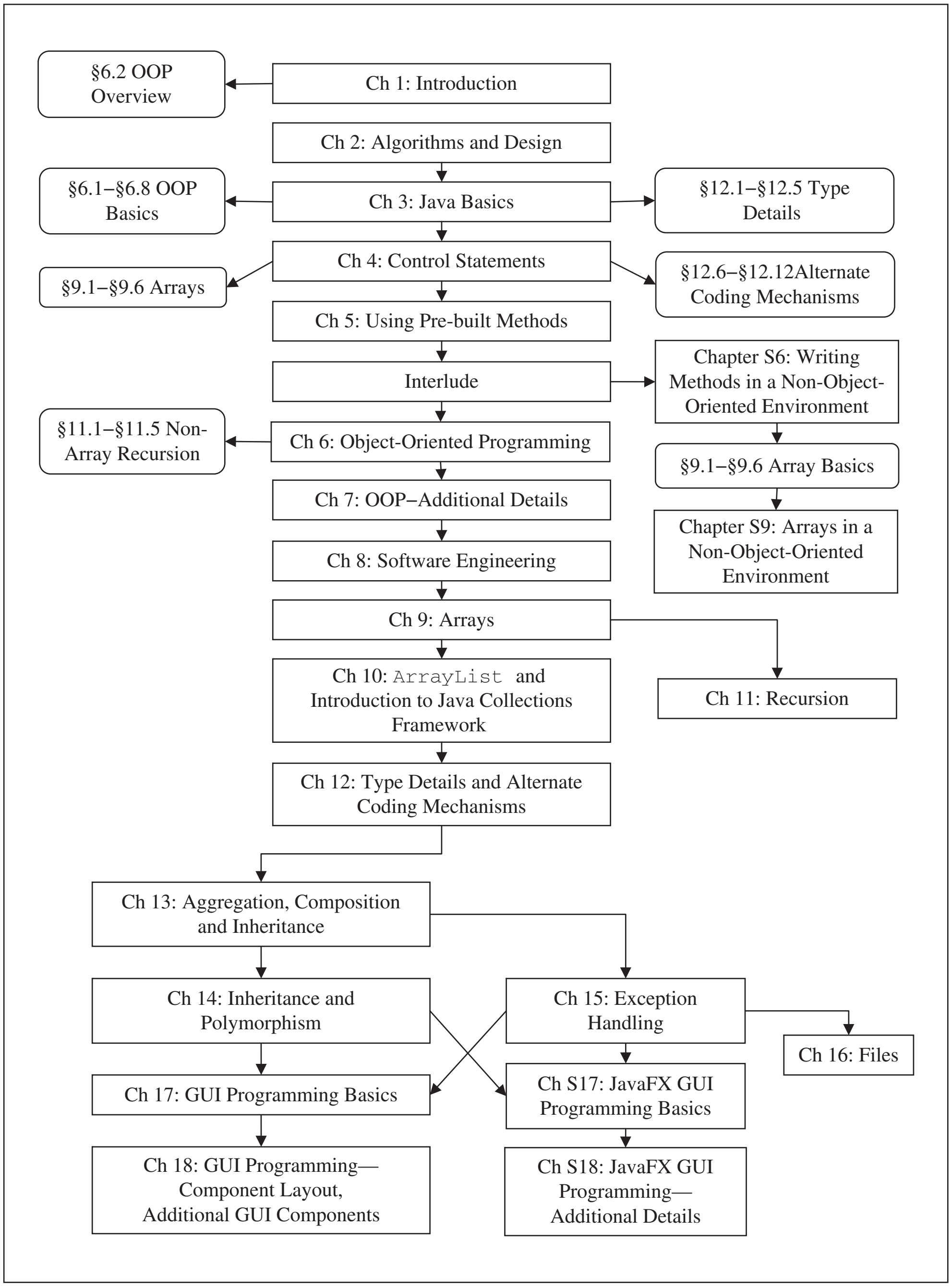

§6.2 OOP Overview
Ch 1: Introduction
Ch 2: Algorithms and Design
§6.1–§6.8 OOP Basics
Ch 3: Java Basics
§12.1–§12.5 Type Details
§9.1–§9.6 Arrays
Ch 4: Control Statements
§12.6–§12.12Alternate Coding Mechanisms
Ch 5: Using Pre-built Methods
Interlude
Chapter S6: Writing Methods in a Non-Object-Oriented Environment
§11.1–§11.5 Non-Array Recursion
Ch 6: Object-Oriented Programming
§9.1–§9.6 Array Basics
Ch 7: OOP–Additional Details
Chapter S9: Arrays in a Non-Object-Oriented Environment
Ch 8: Software Engineering
Ch 9: Arrays
Ch 10: ArrayList and Introduction to Java Collections Framework
Ch 11: Recursion
Ch 12: Type Details and Alternate Coding Mechanisms
Ch 13: Aggregation, Composition and Inheritance
Ch 14: Inheritance and Polymorphism
Ch 15: Exception Handling
Ch 16: Files
Ch 17: GUI Programming Basics
Ch S17: JavaFX GUI Programming Basics
Ch 18: GUI Programming—Component Layout, Additional GUI Components
Ch S18: JavaFX GUI Programming—Additional Details

Figure 0.1 Chapter dependencies

- 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, "Array Basics," and Chapter S9, "Arrays in a Non-Object-Oriented Environment."
- For GUI programming, readers who prefer the Swing toolkit should read Chapters 17 and 18, whereas readers who prefer the JavaFX toolkit 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 non-sequential 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.

Pedagogy

Icons

Program elegance.
Indicates that the associated text deals with a program's coding style, readability, maintainability, robustness, and scalability. Those qualities comprise a program's elegance.

Problem solving.
Indicates that the associated text deals with problem-solving issues. Comments associated with icon attempt to generalize highlighted material in the adjacent text.

Common errors.
Indicates that the associated text deals with common errors.

Hyperlink target.
Indicates the target end of a hyperlink.

Program efficiency.
Indicates that the associated text refers to program-efficiency issues.

Student Resources

At the textbook website, http://www.mhhe.com/dean2e, students (and also teachers) can view and download these resources:

- Links to compiler software—for Oracle's JDK, Helios's TextPad, Eclipse, NetBeans, and BlueJ
- TextPad tutorial

- Eclipse tutorials
- Textbook errata
- Student-version Microsoft PowerPoint lecture slides without hidden notes
  - The student-version slides are identical to the teacher-version slides except that the hidden notes, hidden slides, and quizzes are omitted.
  - Omitting the hidden notes forces the students to go to lecture to hear the sage on the stage fill in the blanks. ☺
- GridWorld code
- Project assignments
- All textbook example programs and associated resource files
- Supplemental chapters
- Supplemental appendices

Instructor Resources

At the textbook website, http://www.mhhe.com/dean2e, instructors can view and download these resources:

- Teacher-version PowerPoint lecture slides with hidden notes
  - Hidden notes provide comments that supplement the displayed text in the lecture slides.
  - For example, if the displayed text asks a question, the hidden notes provide the answer.
- Exercise solutions
- Project solutions
- Test bank materials

Acknowledgments

Anyone who has written a textbook can attest to what a large and well-orchestrated team effort it requires. Such a book can never be the work of only one person, or even a few people. We are deeply indebted to the team at McGraw-Hill Higher Education who have shown continued faith in our writing and invested generously in it.

It was a pleasure to work with Alan Apt during the original book's two-year review period. He provided excellent guidance on several large design issues. Helping us through the various stages of this edition's production was Project Manager Melissa Leick. We would also like to thank the rest of the editorial and marketing team, who helped in the final stages: Raghu Srinivasan, Publisher; Katie Neubauer, Developmental Editor; and Curt Reynolds, Marketing Manager.

All the professionals we have encountered throughout the McGraw-Hill organization have been wonderful to work with, and we sincerely appreciate their efforts.

We would like to acknowledge with appreciation the numerous and valuable comments, suggestions, and constructive criticisms and praise from the many instructors who have reviewed the book. In particular,

Robert Burton, *Brigham Young University*

William Duncan*, Louisiana State University*

Frantisek Franek, *McMaster University*

Junilda Spirollari, *New Jersey Institute of Technology*

Geoffrey Decker, *Northern Illinois University*
Patricia Roth Pierce, *Southern Polytechnic State University*
Jeffrey A. Meunier, *University of Connecticut*
Chris Johnson, *University of Wisconsin, Eau Claire*
Mark Pauley, *University of Nebraska at Omaha*
Christopher Taylor, *Milwaukee School of Engineering*

We would also like to thank Park University colleagues Guillermo Tonsmann and Ken Dewey for their help over the past several years, making suggestions for improvements to the first edition. We appreciate Daniel Green from the Oracle Corporation, who provided advice about new developments in the JavaFX platform and in the Java language itself. A special shout-out goes to Judy Hromcik, a former member of the Advanced Placement (AP) Computer Science Development Committee. In reviewing the second edition, she provided many helpful suggestions, with an eye towards making the book work well in both a college setting and also in an AP Computer Science high school setting.

Thanks to Bruce Jenkins, who worked on the JavaFX chapters. Thanks to Rafael Neri and Miki Katuwal, who created the "Woman seen yesterday" recursive picture in Chapter 11. Finally, thanks to the students who provided feedback and searched diligently for mistakes in order to earn bonus points on their homework. In particular, thank you Komi Labitoko, Nodirbek Hojimatov, Lance Delaney, Ryan Todd, Rob Liebsch, Nirendra Shakya, and Anna Luo.

Sincerely,
John and Ray

Project Summary

One of the special features of this text is the diversity of its projects. Project subject matter spans six broad academic areas, as this short table shows:

Abbreviation	Description	Easy	Moderate	Difficult	Total
CS	computer science and numerical methods	15	14	6	35
Business	business and accounting	11	12	3	26
Sociology	social sciences and statistics	6	8	5	19
Math & Phys	math and physics	10	6	3	19
Engineering	engineering and architecture	2	8	6	16
Biol & Ecol	biology and ecology	0	3	4	7
	Totals	**44**	**51**	**27**	**122**

The abbreviation in the first column above will be used in a larger table below as a brief identification of a particular academic area. The four right-side columns in the above table indicate the number of projects in various categories. Of course, the highest number of projects (35) occurs in the area of computer science and numerical methods. The 29 easy and moderate CS projects are typical CS introductory programming problems. The 6 difficult CS projects provide gentle introductions to some advanced topics like linked list operations, database operations, and simulated annealing.

In addition, there are 26 projects in business and accounting, which include miscellaneous financial calculations, simple bookkeeping problems, and cost-accounting applications. There are 19 projects in social sciences and statistics, which include applications in sociology and political science, as well as general experience. There are 19 projects in math and physics, which include applications in both classical and chaotic mechanics. There are 16 projects in engineering and architecture, which include applications in heating ventilating and air conditioning (HVAC), electrical engineering, and civil engineering. Finally, there are 7 projects in biology and ecology, which include realistic growth and predator-prey simulations. Although we've associated each project with one primary academic area, many of these projects can fit into other academic areas as well.

Because many of these projects apply to disciplines outside the field of computer science, we do not expect that the average reader will already know about all of these "other" topics. Therefore, in our problem statements, we usually take considerable time to explain the topic as well as the problem. And we often explain how to go about solving the problem—in layman's terms. Therefore, working many of these projects will be like implementing computer solutions for customers who are not programmers themselves but understand their subject matter and know what they want you (the programmer) to do for them. They will explain their problem and how to go about solving it. But then they will expect you to create the program that actually solves that problem.

Because our project explanations frequently take considerable printed space, instead of putting them in the book itself, we put them on our website:

http://www.mhhe.com/dean

The following table provides a summary of the projects on the book's website. This table lists all of the book's projects in a sequence that matches the book's sequence. The first column identifies the first point at which you should be able to do the project, by chapter and section, in the form: ChapterNumber.SectionNumber. The second column is a unique project number for the chapter in question. The third column identifies the project's primary academic area with an abbreviation that's explained in the shorter table above. The fourth column indicates the approximate number of pages of code that our solution contains. The fifth column indicates the difficulty relative to where you are in your study of Java. For example, you can see that what we call "easy" involves progressively more pages of code as you progress through the book. The last two columns provide a title and brief description of each project.

Project Summary

Ch./Sec	Proj.	Academic Area	Sol. Pages	Difficulty	Title	Brief Description
2.7	1	Business	0.6	Easy	Annual Bonus–(Flowchart)	Draw a flowchart for an algorithm that computes an annual bonus.
2.7	2	Business	0.3	Easy	Annual Bonus—(Pseudocode)	Write pseudocode for an algorithm that computes an annual bonus.
2.7	3	Business	0.6	Easy	Number of Stamps—(Flowchart)	Draw a flowchart for an algorithm that calculates the number of stamps needed for an envelope. Use one stamp for every five sheets of paper.
2.7	4	Business	0.3	Easy	Number of Stamps—(Pseudocode)	Write pseudocode for an algorithm that calculates the number of stamps needed for an envelope. Use one stamp for every five sheets of paper.
2.7	5	Biol & Ecol	0.5	Moderate	Five Kingdoms—(Pseudocode)	Write pseudocode for an algorithm that identifies a biological kingdom from a set of characteristics.
2.7	6	Math & Phys	0.6	Easy	Speed of Sound—(Flowchart)	Draw a flowchart for an algorithm that provides the speed of sound in a particular medium.
2.7	7	Math & Phys	0.4	Easy	Speed of Sound—(Pseudocode)	Write pseudocode for an algorithm that provides the speed of sound in a particular medium.
2.7	8	Business	0.6	Moderate	Stock Market Return—(Flowchart)	Draw a flowchart for an algorithm that prints the type of market and its probability given a particular rate of return.
2.7	9	Business	0.4	Moderate	Stock Market Return—(Pseudocode)	Write pseudocode for an algorithm that prints the type of market and its probability given a particular rate of return.
2.8	10	Business	0.3	Moderate	Bank Balance—(Pseudocode)	Write pseudocode for an algorithm that determines the number of years until a growing bank balance reaches a million dollars.
2.9	11	Engineering	1.0	Moderate	Loop Termination by User Query—(Flowchart)	Draw a flowchart for an algorithm that calculates the overall miles per gallon for a series of miles and gallons user inputs.

Project Summary

Ch./Sec	Proj.	Academic Area	Sol. Pages	Difficulty	Title	Brief Description
2.9	12	Engineering	0.5	Easy	Loop Termination by User Query—(Pseudocode)	Write pseudocode for an algorithm that calculates the overall miles per gallon for a series of miles and gallons user inputs.
2.9	13	Engineering	0.4	Moderate	Loop Termination by Sentinal Value—(Pseudocode)	Write pseudocode for an algorithm that calculates the overall miles per gallon for a series of miles and gallons user inputs.
2.9	14	Engineering	0.3	Easy	Loop Termination by Counter—(Pseudocode)	Write pseudocode for an algorithm that calculates the overall miles per gallon for a series of miles and gallons user inputs.
2.10	15	CS	0.4	Moderate	Average Weight—(Pseudocode)	Write pseudocode for an algorithm that determines average weight for a group of items.
3.2	1	CS	NA	Easy	Hello World Experimentation	Experiment with the `Hello.java` program to learn the meanings of typical compile-time and runtime error messages.
3.3	2	CS	NA	Moderate	Research	Study Oracle's Java Coding Conventions.
3.3	3	CS	NA	Moderate	Research	Study Appendix 5, "Java Coding-Style Conventions."
3.16 3.23	4	Engineering	2.5	Difficult	Truss Analysis	Given the load in the center of a bridge and the weights of all truss members, compute the compression or tension force in each truss member.
3.17	5	CS	1.0	Easy	Sequence of Commands	Trace a sequence of commands and write a program that executes those commands.
3.17 3.23	6	CS	1.7	Moderate	Computer Speed	Given a simple set of hardware and software characteristics, write a program that estimates the total time to run a computer program.
3.17 3.23	7	Engineering	2.7	Moderate	HVAC Load	Calculate the heating and cooling loads for a typical residence.
3.17 3.23	8	Sociology	3.5	Difficult	Campaign Planning	Write a program to help organize estimates of votes, money, and labor.
3.22	9	CS	1.0	Easy	String Processing	Trace a set of string processing operations and write a program that implements them.
3.23	10	CS	1.2	Easy	Swapping	Trace an algorithm that swaps the values in two variables, and write a program that implements that algorithm.

(continued)

Project Summary

Ch./Sec	Proj.	Academic Area	Sol. Pages	Difficulty	Title	Brief Description
3.23	11	Math & Phys	1.0	Easy	Circle Parameters	Write a program that generates and prints circle-related values.
3.23	12	Sociology	0.4	Easy	One-Hundredth Birthday	Write a program that prompts the user for his/her birthday month, day, and year and prints the date of the user's one-hundredth birthday.
4.3	1	Math & Phys	1.7	Easy	Stopping Distance	Write a program which determines whether a vehicle's tailgating distance is safe, given the speed of the vehicle, the vehicle's tailgating distance, and a formula that gives the distance required to stop the vehicle.
4.3 4.8	2	Engineering	1.9	Moderate	Column Safety	Write a program that determines whether a structural column is thick enough to support the column's expected load.
4.3	3	Business	1.1	Easy	Economic Policy	Write a program that reads in growth rate and inflation values and outputs a recommended economic policy.
4.8	4	Business	2.0	Moderate	Bank Balance	Write a program that determines the number of years until a growing bank balance reaches a million dollars.
4.9 4.12	5	CS	2.6	Difficult	Game of NIM	Implement the game of NIM. Start the game with a user-specified number of stones in a pile. The user and the computer take turns removing either one or two stones from the pile. The player who takes the last stone loses.
4.12	6	Math & Phys	1.0	Easy	Triangle	Write a program that generates an isosceles triangle made of asterisks, given user input for triangle size.
4.12	7	Sociology	0.8	Easy	Mayan Calendar	Implement an algorithm that determines the number of Tzolkins and the number of Haabs in one Calendar Round.
4.12	8	CS	0.9	Easy	Input Validation	Implement an algorithm that repeatedly prompts for inputs until they fall within an acceptable range and computes the average of valid inputs.
4.14	9	Business	2.6	Moderate	Tax Preparation	Write a program that calculates customers' income taxes using the following rules: • The amount of taxes owed equals the taxable income times the tax rate. • Taxable income equals gross income minus $1,000 for each exemption. • The taxable income cannot be less than zero.

Project Summary

Ch./Sec	Proj.	Academic Area	Sol. Pages	Difficulty	Title	Brief Description
4.14	10	CS	1.7	Moderate	Text Parsing	Write a program that converts words to pig Latin.
5.3	1	Math & Phys	1.2	Easy	Trigonometric Functions	Write a demonstration program that asks the user to select one of three possible inverse functions, arcsin, arccos, or arctan, and input a trigonometric ratio. It should generate appropriate output, with diagnostics.
5.3	2	Math & Phys	0.7	Easy	Combining Decibels	Determine the acoustical power level produced by the combination of two sound sources.
5.3	3	Business	1.0	Moderate	Net Present Value Calculation	Write a program that computes the net present value of a proposed investment, given a discount rate and an arbitrary set of future cash flows.
5.5	4	CS	1.5	Moderate	Variable Name Checker	Write a program that checks the correctness of a user-entered variable name, i.e., whether it is: (1) illegal, (2) legal, but poor style, or (3) good style. Assume that "good style" variable names use letters and digits only, and use a lowercase letter for the first character.
5.6	5	CS	1.0	Moderate	Phone Number Dissector	Implement a program that reads phone numbers, and for each phone number, it displays the phone number's three components—country code, area code, and local number.
5.6	6	CS	1.1	Difficult	Phone Number Dissector—robust version	Implement a more robust version of the above phone number program. Allow for shortened phone numbers—phone numbers that have just a local digit group and nothing else, and phone numbers that have just a local digit group and an area code and nothing else.
6.4	1	Biol & Ecol	1.5	Moderate	Plant Germination Observation	Write a program that (1) creates an object called `tree` from the `MapleTree` class; (2) calls a `plant` method to record the planting of the seed; (3) calls a `germinate` method to record the first observation of a seedling and record its height; and (4) calls a `dumpData` method to display the current values of all instance variables.

(continued)

Project Summary

Ch./Sec	Proj.	Academic Area	Sol. Pages	Difficulty	Title	Brief Description
6.4	2	Business	0.5	Easy	Bank Account	Given the code for a `BankAccount` class, provide a driver that tests that class by instantiating an object and calling its methods—`setCustomer`, `setAccountNum`, and `printAccountInfo`.
6.8	3	Math & Phys	1.5	Moderate	Logistic Equation	Exercise the logistic equation: nextX = presentX + r × presentX × (1 − presentX), where presentX = (present x)/(maximum x), and r is a growth factor.
6.9	4	Math & Phys	0.9	Easy	Circle	Given the code for a `CircleDriver` class, write a `Circle` class that defines a `radius` instance variable, a `setRadius` method, and a `printAndCalculateCircleData` method that uses the circle's radius to calculate and print the circle's diameter, circumference, and area.
6.10	5	Engineering	2.0	Moderate	Digital Filter	Given a formula for a "Chebyshev second-order low-pass" filter or a "Butterworth second-order low-pass" filter, with appropriate parameter values, write a program that asks the user to supply a sequence of raw input values and generates the corresponding filtered output.
6.10	6	Sociology	3.1	Difficult	Vending Machine	Write a program that mimics the operations of a vending machine. The program should read amounts of money inserted into the vending machine, ask the user to select an item, and then print the change that's returned to the user.
6.12	7	Math & Phys	1.1	Easy	Rectangle	Implement a `Rectangle` class that defines a rectangle with length and width instance variables, mutator and accessor methods, and a `boolean isSquare` method.
6.12	8	Biol & Ecol	4.0	Difficult	Predator-Prey Dynamics	Write a program that models a species that could be either predator or prey or both. Run a simulation that includes predators, prey, and limited renewable sustenance for the prey.
6.13	9	Math & Phys	2.1	Moderate	Guitar Mechanics	Write a program that simulates the motion of a plucked guitar string.

Project Summary

Ch./Sec	Proj.	Academic Area	Sol. Pages	Difficulty	Title	Brief Description
7.5 7.9	1	CS	3.5	Difficult	Linked List	Given the code for a driver, implement a `Recipe` class that creates and maintains a linked list of recipes. The problem assignment specifies all instance variables and methods in UML class diagrams.
7.7	2	CS	2.5	Easy	Automobile Description	Use method-call chaining to help display properties of automobiles.
7.7 7.9	3	Biol & Ecol	4.6	Difficult	Carbon Cycle	Given the code for a driver, write a pair of classes for a program that models the carbon cycle in an ecosystem. Use two generic classes. One class, `Entity`, defines things. The other class, `Relationship`, defines interactions.
7.8	4	CS	1.4	Easy	IP Address	Implement an `IpAddress` class that stores an Internet Protocol (IP) address as a dotted-decimal string and as four octet `ints`.
7.9	5	Math & Phys	4.5	Moderate	Fraction Handler	Given the `main` method of a driver class, write a `Fraction` class. Include the following instance methods: `add`, `multiply`, `print`, `printAsDouble`, and a separate accessor method for each instance variable.
7.10	6	Math & Phys	1.1	Easy	Rectangles	Write a class that processes rectangular objects. Include a variable that holds the total number of objects and a method that gets that number.
7.11	7	Sociology	2.7	Easy	Person Class	Define a class that simulates the creation and display of `Person` objects.
7.12	8	Sociology	2.7	Moderate	Homework Scores	Write a program that handles homework scores. Use instance variables for actual and maximum points on a particular homework, and use class variables for actual total and maximum total points of all homeworks combined.
7.11	9	Sociology	3.9	Difficult	Political Approval Rating	Write a program that determines the mean and standard deviation of statistical samples.
7.12	10	Engineering	5.7	Difficult	Solar Input for HVAC and Solar Collectors	Write a program that tracks the sun and determines how much solar energy penetrates a glass window of any orientation, at any place and time.

(*continued*)

Project Summary

Ch./Sec	Proj.	Academic Area	Sol. Pages	Difficulty	Title	Brief Description
7.13	11	Engineering	2.8	Moderate	Electric Circuit	Write branch and node classes for lumped-circuit elements. A branch carries current through a resistor in series with an inductor. A node holds voltage on a capacitor connected to a common ground. Driver code is provided in the problem assignment.
7.13	12	Business	5.1	Difficult	Cost Accounting	Write an object-oriented program that demonstrates cost accounting in a manufacturing plant.
7.13	13	Sociology	6.4	Difficult	Political Campaign	Write a program to help organize estimates of votes, money, and labor. This is an object-oriented version of Project 8 in Chapter 3.
7.13	14	Business	2.7	Moderate	Net Present Value Calculation	Write a program that computes the net present value of a proposed investment, given a discount rate and an arbitrary set of future cash flows. This is an OOP version of Project 3 in Chapter 5.
7.13	15	Math & Phys	7.0	Difficult	Three-Body Problem	Write a program to model the three-body problem in which two equally sized moons circle the Earth in different orbits. This illustrates chaotic dynamic motion.
8.5	1	CS	1.6	Easy	Input Validation	Implement an algorithm that repeatedly prompts for inputs until they fall within an acceptable range and computes the average of valid inputs. This is an object-oriented version of Project 8 in Chapter 4.
8.5	2	Engineering	4.0	Difficult	HVAC Load	Calculate the heating and cooling loads for a typical residence. This is an object-oriented version of Project 7 in Chapter 3.
8.8	3	Sociology	2.6	Moderate	Elevator Control	Write a program that mimics the operations of the inside of an elevator. The program should simulate what happens when the user chooses to go to a particular floor and when the user pulls the fire alarm.
8.11	4	CS	2.0	Easy	Prototype Restructuring	Consider the NestedLoopRectangle program in Figure 4.17 in Section 4.12 to be a prototype. Using top-down methodology, restructure it into OOP format.

Project Summary

Ch./Sec	Proj.	Academic Area	Sol. Pages	Difficulty	Title	Brief Description
9.4	1	Biol & Ecol	5.0	Difficult	Demographic Projections	Write a program that projects future world population and average individual wealth as a function of fertility rates and resource extraction rates, and includes the effects of governmental taxation and spending.
9.6	2	CS	3.3	Moderate	Dice-Throwing Simulator	Write a program that simulates the rolling of a pair of dice and prints a histogram showing the frequencies of possible results.
9.6	3	CS	5.1	Difficult	Simulated Annealing—the Traveling Salesman Problem	Write a program that uses simulated annealing to find the shortest itinerary that visits all of the world's major cities exactly one time.
9.7	4	Sociology	2.1	Easy	Party Guest List	Write a program that creates a `Party` object, adds guests to the party, and prints party information.
9.9	5	Sociology	2.7	Easy	Vowel Counter	Write a program that counts the number of uppercase and lowercase vowels in user-entered lines of text and prints a summary report of vowel counts.
9.9	6	Math & Phys	7.6	Difficult	Solution of Simultaneous Algebraic Equations	Write a program that loads a set of simultaneous algebraic equations into two-dimensional arrays and solves the equations by Lower-Upper Decomposition.
9.9	7	Math & Phys	2.5	Moderate	Linear Regression	Write a program that computes a linear regression by fitting a straight line to a series of random data.
9.10	8	Business	3.4	Moderate	Purchase Vouchers	Write a program that creates business vouchers that record purchases, displays current voucher information, and records payments for those purchases.
10.2	1	Sociology	1.1	Easy	Deck of Cards	Write a class that uses an `ArrayList` to hold a deck of cards.
10.4	2	Business	1.9	Easy	Bookstore	Write a program that models the storing and retrieving of books based on title.
10.9	3	Business	0.9	Easy	LIFO Inventory	Write a program that uses a stack to model last-in-first-out inventory costing.
10.10	4	CS	0.7	Easy	Queue Behavior	Write a program that illustrates the behavior of ordinary and priority queues.

(*continued*)

Project Summary

Ch./Sec	Proj.	Academic Area	Sol. Pages	Difficulty	Title	Brief Description
10.10	5	CS	2.1	Moderate	Survey with a two-variable key	Add hash codes to form the key of a map that relates a person and an event to an assessment of that event.
10.10	6	Engineering	4.8	Difficult	Discrete Event Simulation of Queuing Systems	Using iterative enhancement, write code for a discrete-event simulator for: (1) single-server queue with constant service time; (2) priority queue with random service time; (3) multiple servers and queue length limit.
11.5	1	Business	1.6	Easy	Loan Payments and Balances	Write a program that uses recursion to determine the level payment amount needed to pay off a loan in a given number of equal installments. Display balance after each payment.
11.5	2	Math & Phys	2.0	Moderate	Graphical Display of Hénon Map	Convert a given GUI applet into a GUI application. Then modify the application to zoom in to see the fine structure in a classical fractal. Convert from recursion to iteration and zoom in more.
11.6	3	Sociology	2.0	Moderate	Traversing a Maze	Use recursion to traverse a maze by following one wall. Modify the program to back up at dead ends, and modify again to find an interior object.
11.9	4	CS	2.0	Moderate	Enhanced Tree Simulation	Enhance the program in Section 11.9 by adding colored leaves, giving tree trunks and branches color and varying thickness, and randomizing branch lengths and branching angles.
12.3	1	CS	0.7	Easy	ASCII Table	Write a program that prints the 128-character ASCII table in eight tab-separated columns.
12.7	2	CS	0.8	Easy	Circular Queue	A given program implements a circular-array queue. Rewrite the `isFull`, `remove`, and `showQueue` methods by replacing conditional operators, embedded assignments, and embedded increment operators with simpler, more understandable code.
12.7	3	Math & Phys	4.1	Moderate	Polynomial Interpolation	Fit a polynomial to points on either side of a pair of points in an array of data and use that to estimate the value at a position between the pair of points.
12.9	4	CS	1.4	Moderate	Bitwise Operations	Use arithmetic and logical shifting to display the binary values of numbers.

Project Summary

Ch./Sec	Proj.	Academic Area	Sol. Pages	Difficulty	Title	Brief Description
12.11	5	CS	3.5	Moderate	Heap Sort	Use the heap-sort algorithm to sort data.
10.9 12.14	6	Biol & Ecol	5.5	Difficult	Game of Spawn	This "game" simulates reproduction and growth in a rectangular grid of cells. An X indicates life. A dead cell comes to life when it has exactly three living neighbor cells. A living cell remains alive only when surrounded by two or three living neighbor cells.
13.2	1	Business	1.7	Easy	Savings Accounts	Compute and display savings account balances that accumulate with compound interest.
13.4	2	Math & Phys	13.4	Difficult	Statistics Functions	Write a program that generates values for the Gamma, Incomplete Gamma, Beta, Incomplete Beta, and Binomial statistical functions.
13.5	3	Business	3.3	Easy	Car Program	Using inheritance, write a program that keeps track of information about new and used cars.
13.10	4	Sociology	16.4	Difficult	Game of Hearts	Write a program that simulates a basic game of hearts with an arbitrary number of players. Give all players an identical set of good strategies that optimize the chances of winning.
14.7	1	Business	9.0	Difficult	Grocery Store Inventory	Write an inventory program that keeps track of various kinds of food items. Use different methods in an `Inventory` class to process heterogeneous objects representing generic and branded food items. Store the objects together in a common `ArrayList`.
14.7	2	Engineering	8.7	Difficult	Electric Circuit Analysis	Write a program that calculates the steady-state currents in a two-loop electric circuit with resistors, inductors, capacitors, and voltage sources. Include methods to perform addition, subtraction, multiplication, and division of complex numbers.
14.8	3	Business	5.4	Moderate	Payroll	Use polymorphism to write an employee payroll program that calculates and prints the weekly payroll for a company with hourly, salaried, and salaried plus commission employees, where each type of employee gets paid using a different formula. Use an abstract base class.

(continued)

Project Summary

Ch./Sec	Proj.	Academic Area	Sol. Pages	Difficulty	Title	Brief Description
14.8	4	Business	2.9	Moderate	Bank Accounts	Write a bank account program that handles bank account balances for an array of bank accounts. Use two types of bank accounts, checking and savings, derived from an abstract class named `BankAccount`.
15.4	1	Sociology	4.0	Moderate	Body Mass Index	Write a program that prompts the user for height and weight values and displays the associated body mass index (BMI).
15.5	2	CS	6.4	Difficult	Storage and Retrieval of Objects in an Array	Search for a match with the key value in a relational table, using two different search algorithms, a sequential search, and a hashed search.
15.8	3	Biol & Ecol	3.8	Moderate	Whale Watching	Estimate whale length from observed fluke span. Use exceptions to help user correct input format errors.
15.9	4	CS	2.5	Moderate	Date Formatting	Create a class named `Date` that stores date values and prints out the date in either a numeric format or an alphabetic format.
15.9	5	CS	5.5	Difficult	Input Utility	Write a utility class that reads and parses keyboard inputs for the following: `String`, `char`, `double`, `float`, `long`, `int`.
16.2	1	Engineering	3.7	Moderate	Road Use Survey	Model traffic flowing on a highway past a particular place, store observations, and read file later for analysis.
16.2	2	Business	2.9	Moderate	Mail Merge	Write a program that reads a form letter from a text file and modifies custom fields.
16.4	3	CS	1.5	Easy	Appending Data to an Object File	Implement code needed to append data to an object file.
16.2 16.9	4	CS	5.0	Moderate	File Converter	Write a program that changes whitespace in text files.
17.12	1	Engineering	4.1	Moderate	Animated Garage Door	Write a program that simulates the operation of an automatic garage door and its controls and visually display its position as it operates.
17.14	2	Sociology	3.0	Moderate	Color Memorization	Write a program that tests the user's ability to memorize a sequence of colors.
17.14	3	Business	8.7	Difficult	Grocery Inventory GUI	Write a GUI version of the Grocery Store Inventory project in Chapter 14.
17.15	4	Sociology	4.2	Moderate	Word Order Game	Create a simple interactive game that helps kids practice their alphabetic skills.

Project Summary

Ch./Sec	Proj.	Academic Area	Sol. Pages	Difficulty	Title	Brief Description
17.16	5	Business	3.8	Moderate	Airline Reservations	Write a GUI program that assigns seats on airline flights.
18.3	1	CS	1.7	Easy	Changing Color and Alignment	Write an interactive program that modifies the color and position of buttons in a GUI window.
18.7	2	CS	1.9	Easy	Click Tracker	Write an interactive program that modifies the borders and labels of buttons in a GUI window.
18.11	3	Sociology	3.4	Moderate	Tic-Tac-Toe	Create an interactive Tic-Tac-Toe game.
18.11	4	Sociology	4.3	Moderate	Word Order Game, revisited	Modify Chapter 17's Word Order Game program so that it uses embedded layout managers.
18.11	5	Engineering	7.5	Difficult	Thermal Diffusion in a Ground-Source Heat Pump's Well	Write a program to compute and display temperature around a ground-source heat pump well as a function of distance from the well center and the time of year.

This page intentionally left blank

CHAPTER 1

Introduction to Computers and Programming

Objectives

- Describe the various components that make up a computer.
- List the steps involved in program development.
- Know what it means to write algorithms using pseudocode.
- Know what it means to write programs with programming language code.
- Understand source code, object code, and the compilation process.
- Describe how bytecode makes Java portable.
- Become familiar with Java's history—why it was initially developed, how it got its name, and so forth.
- Enter, compile, and run a simple Java program.

Outline

1.1 Introduction

This book is about problem solving. Specifically, it is about creating solutions to problems through a set of precisely stated instructions. We call such a set of instructions (when in a format that can be entered into and executed on a computer) a *program*. To understand what a program is, think about the following situation.

Suppose you manage a department store, and you don't know when to restock the shelves because you have difficulty keeping track of inventory. The solution to the problem is to write a set of instructions that keeps track of items as they arrive at your store and as they are purchased. If the instructions are correct and in a format that is understood by a computer, you can enter the instructions as a program, run the program, and enter item-arrival and item-purchase data as they occur. You can then retrieve inventory information from the computer any time you need it. That accurate and easily accessible knowledge enables you to restock your shelves effectively, and you are more likely to turn a profit.

The first step to learning how to write programs is to learn the background concepts. In this chapter, we teach background concepts. In subsequent chapters, we use these background concepts while explaining the really good stuff—how to program.

We start this chapter by describing the various parts of a computer. We then describe the steps involved in writing a program and in running a program. Next, we narrow our focus and describe the programming language we'll be using for the remainder of the book—Java. We present step-by-step instructions on how to enter and run a real Java program, so that you'll be able to gain some hands-on experience early on. We finish the chapter with an optional GUI-track section that describes how to enter and run a graphical user interface (GUI) program.

1.2 Hardware Terminology

A *computer system* is made up of all the components that are necessary for a computer to operate and the connections between those components. There are two basic categories of components—*hardware* and *software*. Hardware consists of the physical components associated with a computer. Software consists of the programs that tell a computer what to do. For now, let's focus on hardware.

Our description of a computer's hardware provides you with the information you'll need as a beginning programmer. (A *programmer* is a person who writes programs.) After you master the material here, if you decide you want more, go to Webopedia's hardware web page at http://www.webopedia.com/hardware.

The Big Picture

Figure 1.1 shows the basic hardware components in a computer system. It shows input devices at the left (keyboard, mouse, and scanner), output devices at the right (monitor and printer), storage devices at the bottom, and the central processing unit (CPU) and main memory in the center. The arrows in Figure 1.1 represent connections between the components. For example, the arrow from the keyboard to the CPU-main memory represents a cable (a connecting wire) that transmits information from the keyboard to the CPU and main memory. Throughout this section, we explain the CPU, main memory, and all the devices in Figure 1.1.

Input and Output Devices

Input and output devices are collectively called *I/O devices*. There are different definitions of an *input device,* but usually the term refers to a device that transfers information into a computer. Remember—information going <u>in</u>to a computer is input. For example, a keyboard is an <u>in</u>put device because when a person presses a key, the keyboard sends information into the computer (it tells the computer which key was pressed).

There are different definitions of an *output device,* but usually the term refers to a device that transfers information out of a computer. Remember—information going <u>out</u> of a computer is <u>out</u>put. For example, a *monitor* (also called a *display* or a *screen*) is an output device because it displays information going out of the computer.

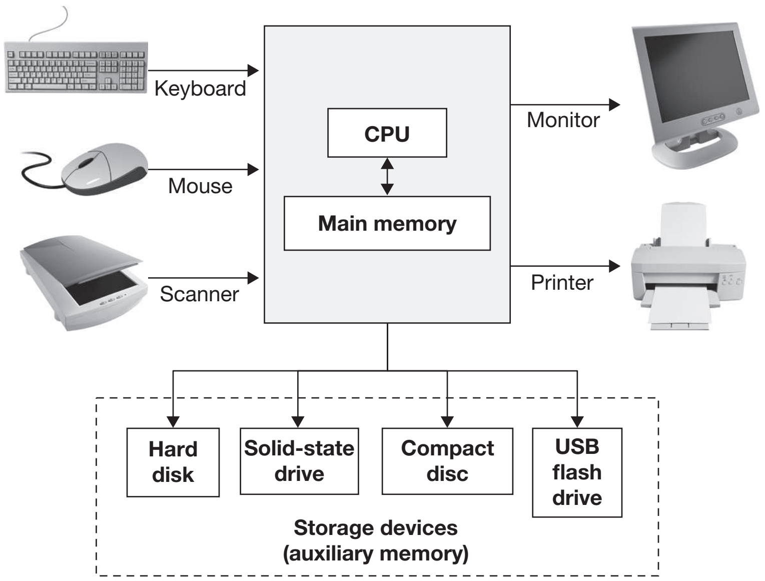


Figure 1.1 A simplified view of a computer

Central Processing Unit

The *central processing unit,* often referred to as the *processor* or *microprocessor,* can be considered the computer's brain. As with a biological brain, the CPU splits its time between two basic activities— thinking and managing the rest of its system. The "thinking" activities occur when the CPU reads a program's instructions and executes them. The "managing its system" activities occur when the CPU transfers information to and from the computer system's other devices.

Here's an example of a CPU's thinking activities. Suppose you have a program that keeps track of a satellite's position in its orbit around the Earth. Such a program contains quite a few mathematical calculations. The CPU performs those mathematical calculations.

Here's an example of a CPU's managing-its-system activities. Suppose you have a job application program. The program displays boxes in which a person enters his/her name, phone number, and so on. After entering information, the person uses his/her mouse and clicks a Done button. For such a program, the CPU manages its system as follows. To display the initial job application form, the CPU sends information to the monitor. To gather the person's data, the CPU reads information from the keyboard and mouse.

If you're thinking about buying a computer, you'll need to judge the quality of its components. To judge the quality of its components, you need to know certain component details. For CPUs, you should know the popular CPUs and the range of typical CPU speeds. We present the following CPUs and CPU speeds with hesitation because such things change in the computer world at a precipitous rate. By presenting such details, we're dating our book mercilessly. Nonetheless, we forge ahead. . . .

At the time of this second edition's writing:

- Popular CPUs—Core i7 (manufactured by Intel), Phenom II (manufactured by AMD).
- Current CPU speeds—anywhere from 2.5 GHz up to 3.8 GHz.

What is *GHz,* you ask? GHz stands for *gigahertz* and is pronounced with hard *g*'s, as in *giggle*. *Giga* means billion, and *hertz* is a unit of measure that deals with the number of times that something occurs per second. A 2.5 GHz CPU uses a clock that ticks 2.5 billion times per second. That's fast, but a 3.8 GHz CPU is even faster—it uses a clock that ticks 3.8 billion times per second. A CPU's clock speed provides a rough measure for how fast the CPU gets things done. Clock ticks are the initiators for computer tasks. With more clock ticks per second, there are more opportunities for getting tasks done.

Main Memory

When a computer executes instructions, it often needs to save intermediate results. For example, in calculating the average speed for 100 speed measurements, the CPU needs to calculate the sum of all the speed values prior to dividing by the number of measurements. The CPU calculates the sum by creating a storage area for it. For each speed value, the CPU adds the value to the sum storage area. Think of memory as a collection of storage boxes. The sum is stored in one of memory's storage boxes.

There are two categories of memory— *main memory* and *auxiliary memory.* The CPU works more closely with main memory. Think of main memory as a storage room next to the boss's office. The boss is the CPU, and he/she stores things in the storage room's storage boxes whenever the need arises. Think of auxiliary memory as a warehouse that's across the street from the boss's building. The boss uses the warehouse to store things, but doesn't go there all that often. Since auxiliary memory is considered secondary to main memory, auxiliary memory is sometimes referred to as *secondary storage*. We'll consider auxiliary memory details in the next subsection. For now, we'll focus on main memory details.

The CPU relies on main memory a lot. It's constantly storing data in main memory and reading data from main memory. With this constant interaction, it's important that the CPU and main memory are able to communicate quickly. To ensure quick communication, the CPU and main memory are physically close together. They are both constructed on *chips,* and they both plug into the computer's main circuit board, the *motherboard.* See Figure 1.2 for a picture of a motherboard, a CPU chip, and main memory chips.

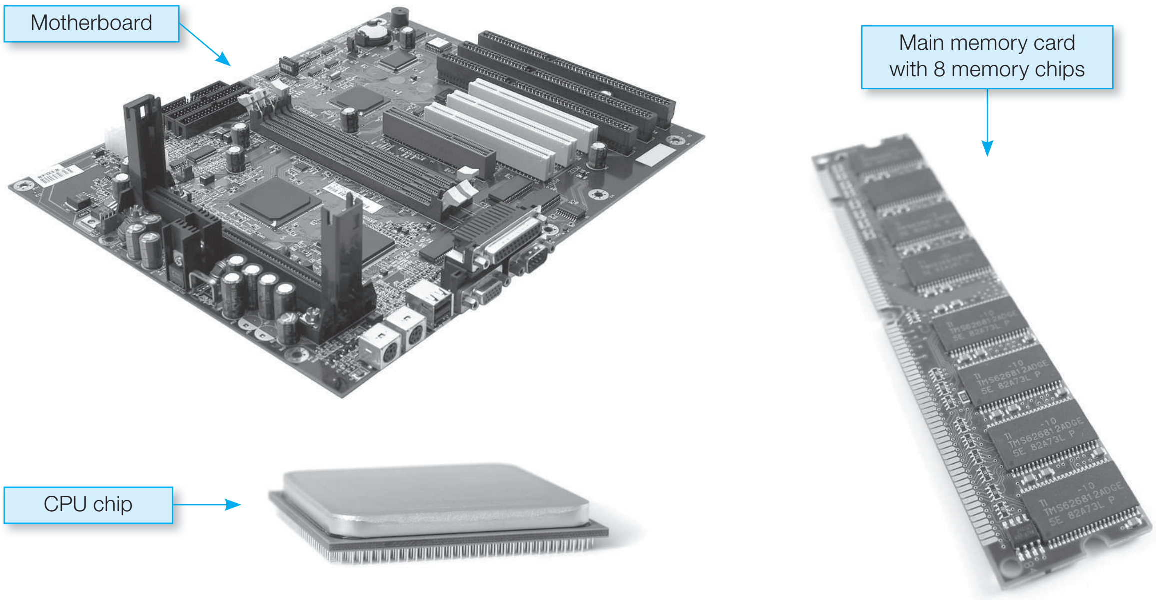


Figure 1.2 Motherboard, CPU chip, and main memory chips

Main memory contains storage boxes, and each storage box contains a piece of information. For example, if a program stores our last name, Dean, it uses eight storage boxes: one for the first half of D, one for the second half of D, one for the first half of e, one for the second half of e, and so on. After storing the four letters, the program will probably need to retrieve them at some point later on. For information to be retrievable, it must have an address. An *address* is a specifiable location. A postal address uses street, city, and ZIP code values to specify a location. A computer address uses the information's position within main memory to specify a location. Main memory's first storage box is at the zero position, so we say it's at address 0. The second storage box is at the one position, so we say it's at address 1. See Figure 1.3. It shows Dean stored in memory starting at address 50,000.

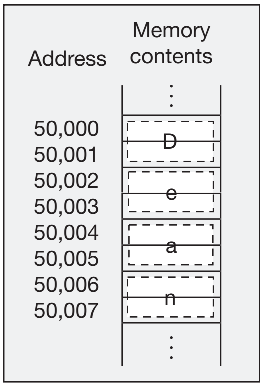


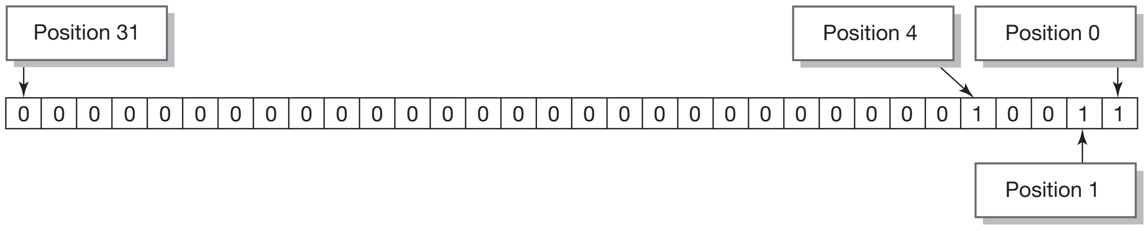
Figure 1.3 The characters D, e, a, n stored in memory starting at address 50,000

It's important to understand the formal terminology when talking about the size of main memory. Suppose you're buying a computer and you want to know how big the computer's main memory is. If you ask a salesperson how many "storage boxes" it contains, you'll probably get a perplexed look. What you need to do is ask about its *capacity*—that's the formal term for its size. If you ask for the main memory's capacity, the salesperson will say something like, "It's 1 *gigabyte*." You already know that *giga* means billion. A *byte* refers to the size of one storage box. So a 1 gigabyte capacity main memory holds 1 billion storage boxes.

Let's describe storage boxes in more detail. You know that storage boxes can hold characters, like the letter D. But computers aren't very smart—they don't understand the alphabet. They only understand 0's and 1's. So computers map each alphabet character to a series of sixteen 0's and 1's. For example, the letter D is 00000000 01000100. So in storing the letter D, main memory actually stores 00000000 01000100. Each of the 0's and 1's is called a *bit*. And each of the eight-bit groupings is called a *byte*.

Are you wondering why computers use 0's and 1's? Computers understand only high-energy signals versus low-energy signals. When a computer generates a low-energy signal, that's a 0. When a computer generates a high-energy signal, that's a 1.

You know that computers store characters as 0's and 1's, but did you know that computers also store numbers as 0's and 1's? Formally, we say that computers use the *binary number system*. The binary number system uses just two digits, 0 and 1, to represent all numbers. For example, computers store the number 19 as 32 bits, 00000000 00000000 00000000 00010011. The reason those 32 bits represent 19 is that each 1-value bit represents a power of 2. Note that there are three 1-value bits. As shown here, the 1-value bits are at positions 0, 1, and 4, where the positions start at 0 from the right side. A bit's position determines its power of 2. Thus, the rightmost bit, at position 0, represents 2 raised to the power 0, which is 1 ($2^0 = 1$). The bit at position 1 represents



2 raised to the power 1, which is 2 ($2^1 = 2$). And the bit at position 4 represents 2 raised to the power 4, which is 16 ($2^4 = 16$). Add the three powers and you get 19 ($1 + 2 + 16 = 19$). Voila! Appendix 8 contains additional information about the binary number system. Feel free to peruse the appendix now, or wait until after you've read Chapter 12, which introduces another number system—hexadecimal. The appendix provides an in-depth discussion of the binary and hexadecimal number systems, as well as a third number system—octal.

Be aware that main memory is often referred to as *RAM*. RAM stands for *random access memory.* Main memory is considered "random access" because data can be directly accessed at any address; that is, at a "random" address. That's in contrast to some storage devices where data is accessed by starting at the very beginning and stepping through all the data until the target data is reached.

Once again, if you're buying a computer, you'll need to judge the quality of its components. For the main memory/RAM component, you'll need to know whether its capacity is adequate. At the time of this book's writing typical main memory capacities range from 2 GB up to 8 GB, where *GB* stands for *gigabyte*.

Auxiliary Memory

Main memory is *volatile,* which means that data is lost when power to the computer goes off. You might ask: If data is lost when power goes off, how can anyone save anything permanenly on a computer? The answer is something you do (or should do) frequently. When you perform a save command, the computer makes a copy of the main memory data you're working on and stores the copy in auxiliary memory. Auxiliary memory is *nonvolatile,* which means that data is not lost when power to the computer goes off.

One advantage of auxiliary memory over main memory is that it's nonvolatile. Another advantage is that its cost per unit of storage is much less than main memory's cost per unit of storage. A third advantage is that it is more *portable* than main memory (i.e., it can be moved from one computer to another more easily).

The disadvantage of auxiliary memory is that its *access time* is quite a bit slower than main memory's access time. Access time is the time it takes to locate a single piece of data and make it available to the computer for processing.

Auxiliary memory comes in many different forms, the most common of which are hard disks, solid-state drives (SSDs), universal serial bus (USB) flash drives, and compact discs. All these devices are called *storage media,* or simply *storage devices*. Figure 1.4 shows pictures of them.

Hard Disks and Solid-State Drives

Hard disks and *solid-state drives* serve the same basic purpose: They provide the primary permanent storage for a computer. They have different advantages and disadvantages, which make them attractive to different types of computers. Most *desktop* computers (computers that remain stationary on or next to a desk) use hard disks. On the other hand, some *laptop* computers (computers that are portable enough to sit on

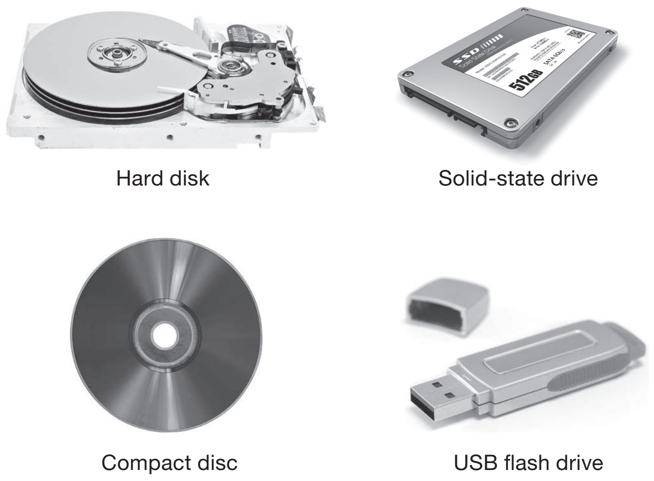


Figure 1.4 Hard-disk drive, solid-state drive, compact disc, and USB flash drive

someone's lap) and many *tablet* computers (computers that use a touch screen for their primary input device, rather than a keyboard and mouse) use solid-state drives instead of hard disks.

Solid-state drives are particularly suitable for laptop and tablet computers because they have no moving mechanical parts. As such, they are more resistant to damage when subject to travel. Also, they are smaller and lighter than hard disks, which once again makes them particularly suitable for laptop and tablet computers. A disadvantage of solid-state drives is their cost: Given a solid-state drive and a hard disk device with the same capacity, the solid-state drive will be quite a bit more expensive. Thus, solid-state drives are usually cost prohibitive for computers with large capacity needs. Desktop computers fall into the large-capacity camp, so they usually use hard disks. The trend is for portable computers to rely on *cloud storage,* where computers transmit their data over the Internet for storage on pools of computers hosted by third-party data centers. With cloud storage, portable computers' local storage needs are reduced and solid-state drives become more affordable.

Although some high-end desktop computers use SSDs, most use hard disks. Hard disks are slower than SSDs because for a computer to access a particular piece of data on a disk, the computer must wait for the disk to spin the data to the place where the data can be read. The spinning and reading mechanisms are part of the disk's *drive*. As a disk drive rotates its disks, its *heads* (electronic sensors) access the disks' data as it spins past.

Access to solid-state-drive storage is faster than access to hard disk storage because solid-state drives don't have to wait for mechanical parts to move. Instead, they simply have to wait for electronic signals to arrive. Electronic signals, traveling at the speed of light, move much faster than spinning disks.

Off-line Storage

Although hard disk and solid-state drives are sometimes located outside a computer's metal case and connected to the computer with a cable (such drives are called *external drives*), in the interest of speed, mosthard disk and solid-state drives are located inside a computer's metal case. Their internal location makes it difficult for them to be transferred from one computer to another. On the other hand, *off-line storage devices,* such as USB flash drives and compact discs, can be easily transferred from one computer to another because they are designed to connect and disconnect to and from a computer easily.

USB flash drives, also called *thumb drives,* are particularly portable because they are the size of a person's thumb and they can be *hot swapped* into virtually any computer. (Hot swapping is when you plug a device into a computer while the computer is on.) The "USB" in USB flash drive stands for "universal serial bus," and it refers to a particular type of connection to the computer. More specifically, it refers to a particular type of connection wire and connection socket. A flash drive uses that type of connection, and we therefore call it a *USB flash drive.* USB flash drives plug into USB ports, where *port* is the formal term for a connection socket. USB ports are ubiquitous on computers, and that is another reason that USB flash drives are particularly portable.

USB flash drives are built with *flash memory,* which is a popular form of nonvolatile storage with no moving mechanical parts. Solid-state drives also use flash memory. However, USB flash drives are much slower than solid-state drives (and slightly slower than hard disks) because USB flash drives are connected to the computer with a narrow, relatively slow USB interface, whereas solid-state drives are connected to the rest of the computer with a wide, relatively fast interface.

Compact discs provide a less expensive and slower form of off-line storage. The most popular types of compact discs can be grouped as follows:

- CD-Audio—for storing recorded music, usually referred to as just "CD" (for compact disc)
- CD-ROM, CD-R, CD-RW—for storing computer data and recorded music
- DVD, DVD-R, DVD-RW—for storing video, computer data, and recorded music
- Blu-ray—for storing video and computer data, designed to replace DVDs

The "ROM" in CD-ROM stands for "read-only memory." *Read-only* memory refers to memory that can be read from, but not written to. Thus, you can read a CD-ROM, but you can't change its contents. With CD-Rs, you can write once and read as many times as you like. With CD-RWs, you can write and read as often as you like.

DVD stands for "digital versatile disc" or "digital video disc." DVDs parallel CD-ROMs in that you can read from them, but you can't write to them. Likewise, DVD-Rs and DVD-RWs parallel CD-Rs and CD-RWs in terms of their reading and writing capabilities.

Blu-ray, also known as Blu-ray Disc (BD), is the name of an optical disc format that was invented to store high-definition videos and large amounts of data. The technology is called Blu-ray because unlike DVDs, which use a red laser, Blu-ray discs are accessed using a blue-violet laser. Eventually, Blu-ray discs will replace DVDs in terms of popularity, but not for several years, because (1) Blu-ray technology is more expensive, (2) Blu-ray technology has slower access speeds, and (3) Blu-ray players are backward compatible (so DVDs can run on Blu-ray players).

Storage Capacity Comparison

Different storage devices have different storage capacities. At the time of this book's writing:

- Typical hard disks have a capacity range from 250 GB up to 3 TB (*TB* stands for *terabyte,* where *tera* is 1 trillion).
- Typical solid-state drives have a capacity range from 120 GB up to 512 GB.
- Typical USB flash drives have a capacity range from 8 GB up to 64 GB.
- Typical CD-ROMs, CD-Rs, and CD-RWs have a capacity of 700 MB (*MB* stands for *megabyte,* where *mega* is 1 million).
- Typical DVDs, DVD-Rs, and DVD-RWs have a capacity range from 4.7 GB up to 8.5 GB.
- Typical Blu-ray discs have a capacity range from 25 GB up to 50 GB.

File Access

To access data on your computer, you'll need to access the file that contains the data. A *file* is a group of related instructions or a group of related data. For example, (1) a program is a file that holds a set of instructions, and (2) a Word document is a file that holds text data created by Microsoft Word.

Files are stored on auxiliary memory storage devices. In order to retrieve a file (for the purpose of viewing, copying, etc.), you need to specify the storage device on which the file is stored. On computers that use Microsoft Windows, the different storage devices are specified using a drive letter followed by a colon. If your computer has a hard disk drive or a solid-state drive, your computer will refer to one of the drives using drive letter C (`C:`). If your computer has additional hard disk drives or solid-state drives, it will refer to them using subsequent drive letters (`D:`, `E:`, etc.). If your computer has compact-disc drives, it will refer to them using the first unused drive letters starting no earlier than `D:`. If your computer has additional storage devices, such as external hard drives and USB flash drives, it will refer to them using the next unused drive letters, starting no earlier than `D:`.

You might have noticed that drive letters A and B were not mentioned in this discussion so far. In the past, `A:` and `B:` were used for floppy disk drives. *Floppy disks* (also called *diskettes*) are off-line storage devices that were very popular from the mid-1970s through about 2005. They are called "floppy" because the original forms of these devices would bend, in contrast to hard disks, which do not bend. Computer manufacturers no longer provide floppy disk drives because floppy disks have been superseded by more durable, greater capacity off-line storage devices, such as USB flash drives and compact discs.

Even though floppy disks are no longer used, their legacy lives on. Because Windows-based computers reserved drive letters A and B for floppy disk drives in the past, Windows-based computers continue to start with drive letter C for hard-disk and solid-state drives. Because floppy disks became synonymous with file storage in the 1980s and 1990s, software manufacturers introduced floppy disk icons that, when clicked, would save the user's current file. Using a floppy disk icon for a file-save operation is still the norm today. This standard floppy disk icon should look familiar: 💾

Common Computer-Hardware Vocabulary

When buying a computer or when talking about computers with your computer friends, you'll want to make sure to understand the vernacular—the terms that people use in everyday speech as opposed to the terms found in textbooks—so that you will be able to understand what's going on. When a computer-savvy person refers to a computer's memory by itself, the person typically means main memory—the computer's RAM. When someone refers to a computer's *disk space,* the person typically means the capacity of the computer's hard disk.

Pace of Computer Improvements

For as long as memory and CPU components have been around, manufacturers of these devices have been able to improve their products' performances at a consistently high rate. For example, RAM and hard disk capacities double approximately every two years. CPU speeds also double approximately every two years.

An *urban legend* is a story that spreads spontaneously in various forms and is popularly believed to be true. The following exchange is a classic Internet urban legend that comments on the rapid pace of computer improvements.[1] Although the exchange never took place, the comments, particularly the first one, are relevant.

[1]Snopes.com, *Rumor Has It,* on the Internet at http://www.snopes.com/humor/jokes/autos.asp (visited in September, 2012).