Modern Software Development Using Java

Second Edition

Paul T. Tymann
Rochester Institute of Technology

G. Michael Schneider
Macalester College

COURSE TECHNOLOGY
CENGAGE Learning™

Australia • Brazil • Japan • Korea • Mexico • Singapore • Spain • United Kingdom • United States

COURSE TECHNOLOGY
CENGAGE Learning™

Modern Software Development Using Java, Second Edition

Paul T. Tymann and G. Michael Schneider

Acquisitions Editor: Amy Jollymore

Senior Product Manager: Alyssa Pratt

Development Editor: Dan Seiter

Editorial Assistant: Erin Kennedy

Senior Production Editor: Catherine DiMassa

Cover Designer: Beth Paquin

Compositor: GEX Publishing Services

Print Buyer: Julio Esperas

ISBN-13: 978-14239-0123-5

ISBN-10: 1-4239-0123-1

Cengage Learning
25 Thomson Place
Boston, Massachusetts 02210
USA

Cengage Learning is a leading provider of customized learning solutions with office locations around the globe, including Singapore, the United Kingdom, Australia, Mexico, Brazil, and Japan. Locate your local office at:
international.cengage.com/region

Cengage Learning products are represented in Canada by Nelson Education, Ltd.

For your lifelong learning solutions, visit **course.cengage.com**

Purchase any of our products at your local college store or at our preferred online store **www.ichapters.com**

Printed in the United States of America
2 3 4 5 6 7 8 9 11 10 09 08

Table of Contents

PREFACE TO THE SECOND EDITION

This text, *Modern Software Development Using Java, Second Edition*, is designed for a second course in computer science. This class is often referred to as "CS2" in curricular guidelines, including the *ACM/IEEE Computing Curriculum 2001* (CC2001). CS2 traditionally follows the introductory programming and problem-solving course that is the starting point for most computer science undergraduates. Because CS2 covers such a large amount of material, some colleges and universities have extended it to a one-year sequence. This text would be appropriate for a two-course, one-year version called CS2/CS3.

This book takes a different view from other texts regarding the appropriate topics to include in CS2, due to differences between software development today and in the not-too-distant past. The field of computing is changing dramatically, and new hardware and software developments are announced daily. The computer science curriculum must adapt if it is to remain current with these new developments as well as the changing background of its students.

For many years, beginning with *ACM Curriculum '78*, CS2 was understood to be a course on data structures, perhaps with supporting material on algorithm analysis, sorting, and searching. For twenty years, it was accepted that CS2 would cover linked lists, stacks, queues, trees, search trees, balanced trees, and hash tables. For more than two decades, computer science students dutifully learned to implement a queue using circular arrays, update pointers in a doubly linked list, and work through the algorithms for balancing a B-tree. Recent versions of the course have made small changes, often to expand coverage of object-oriented programming. However, the majority of time in CS2 is still dedicated to implementing the classic data structures of computer science. Despite the enormous changes in computing and software design in the past two decades, the syllabi of many CS2 classes could have been written in 1980 (with the exception of Java). In our opinion, this is an outmoded view. Just as other courses have adapted to present a more modern outlook, we feel it is time for CS2 to change to address new concerns in computer science and software development.

The basic precept of our text is that data structures, while important, should no longer be the sole focus of CS2. Instead, we should recognize important new developments in software development. We believe that the following three concerns will have the greatest impact on the curriculum:

- Students need to be introduced as early as possible to *all* stages in the software life cycle, including requirements, specification, and design.
- Languages such as Java contain libraries that already provide the functionality required to use a wide range of data structures.
- Topics delayed until junior- or senior-level classes are becoming increasingly important in software development and should be introduced earlier in the curriculum.

This text addresses all three concerns to provide a more modern view of software design and development.

ORGANIZATION AND COVERAGE

Part I: Object-Oriented Design and Development (Chapters 2–4)

Part I (Chapters 2–4) introduces and reviews the software life cycle. Too often, first-year courses focus so heavily on implementation issues that students develop the mistaken impression that "software development = programming." CS1, the first programming course, usually introduces the basic concepts of object-oriented programming, but limited class time often requires brief coverage. Students are exposed to classes, objects, and methods, but have little practical experience using these concepts to solve large, complex problems. In Part I, we expand on this initial presentation and introduce students to all aspects of the problem-solving process.

Chapter 1 is an overview of the software life cycle, especially the phases that must precede implementation—requirements, specification, and design. In Chapter 2, we look more closely at program design as we investigate the following questions:

- How can we determine from the problem specification document which classes we need to include in our design?
- How can we determine the appropriate states and behaviors to include in these classes?
- How can we use a formal design notation such as UML to clearly and unambiguously represent our proposed design?

Chapter 2 also takes an in-depth look at inheritance. The chapter presents different models of inheritance, including specialization, specification, and limitation, and shows how each model enhances the problem-solving process. Chapter 2 discusses object-oriented design in a conceptual, language-independent way, and Chapter 3 examines how these features are realized in Java. Chapter 3 provides a full treatment of Java's object-oriented facilities, including abstract classes, interfaces, public/private/protected visibility, method overloading, generics, and dynamic binding.

Chapter 4 is an extended case study that follows the solution of a problem through its life cycle, from user requirements to implementation and testing. This case study ties together the object-oriented concepts from the previous three chapters and demonstrates how they form a coherent design philosophy.

Part II: Algorithms and Data Structures (Chapters 5–9)

Part II (Chapters 5–9) addresses the topics of algorithms and data structures, which have been the central focus of the second course. These topics are still important, and students must be familiar with the classic data structures of computer science. However, our coverage focuses more on their behavior and best- and worst-case performance characteristics than on the nitty-gritty details of implementation. The appearance of data structure libraries in languages such as Java has dramatically changed the way we should approach this topic. It is no longer as important for students to *build* every data structure they use; however, they must be able to *understand*, *analyze*, and intelligently *select* and *use* the most appropriate data structures to solve a given problem.

Chapter 5 presents the mathematical tools needed to analyze the behavior of data structures; namely, the asymptotic analysis of algorithms, also called Big-O notation.

Next we begin our investigation of the most widely used data structures. It is important that students not view the many structures presented in these chapters as disparate, unrelated issues. Instead, they must see this discussion as a single, integrated topic. Therefore, we first present a taxonomy that categorizes all data structures into one of four fundamental groups based on the nature of the relationship between elements in the collection. Then, in the following chapters, we investigate each of the four groups in this taxonomy.

Chapter 6 looks at the *linear* (1:1) grouping, which includes lists, stacks, queues, and priority queues. We explain why these structures can be assigned the same classification, differing only in the types of operations that are permitted or excluded. We also discuss the conditions under which each of these linear structures can be useful and efficient.

Chapter 7 looks at the *hierarchical* (1: many) grouping, which includes binary trees, search trees, balanced trees, and heaps.

Chapter 8 covers the final two classifications in the taxonomy—*graph* (many:many) structures, and *sets*, in which there is no positional relationship between elements in the collection. The latter discussion focuses on the set types of greatest importance to the computer scientist—maps and hash tables. In all cases, our coverage centers on explaining the performance, behaviors, and characteristics of each type of data structure.

Chapter 9, one of the most important chapters in Part II, provides in-depth coverage of the *Java Collection Framework*, a package of classes that provides functionality for many of the data structures discussed in the previous three chapters. This framework includes multiple implementations of each data structure and methods for creating, accessing, and modifying them. Today, students no longer need to build every structure from scratch; instead, they can reuse existing code from the Java Collection Framework. Code reuse is one of the most important benefits of object-oriented programming, but students rarely get to exploit it. In this book, students can take advantage of class libraries to experience the enormous improvement in productivity that comes from object-oriented programming, inheritance, and code reuse.

Part III: Modern Programming Practices (Chapters 10–13)

Not long ago, topics such as recursion and algorithm analysis were considered too advanced for a first class. Today they are routinely included in CS1, and it is not unusual to see them discussed in a high school course. Such is the way with many ideas in computer science—they initially enter the curriculum as advanced subject matter but then migrate to introductory and intermediate courses.

This process is occurring in CS2 as well. Today, a new computer science student must be comfortable with topics that were previously treated only in advanced classes, including:

- Exception handling and fault-tolerant software

- Streams and stream-oriented programming
- Threads and multithreaded servers
- Graphical user interfaces and event-driven programming
- Human/computer interaction
- Networking
- Client/server programming
- Security

Most of these topics are not part of the typical CS2 course. Instead, they are delayed until junior or senior elective courses in operating systems, networks, and computer graphics. However, most modern software projects involve a visual front end, access to a network, multiple threads of control, and extensive error handling. Therefore, it is much more important for beginning students to be introduced to these critical topics than it is for them to implement a tournament tree or adjust pointers in a doubly linked list. While our treatment of the topics is introductory and not intended to replace material covered in advanced courses, presenting these topics in the first year can be very beneficial. By working with these ideas from the beginning, students will feel more comfortable when they encounter the ideas in later courses. This approach is also fully consistent with the recommendations of the ACM/IEEE CC2001 report.

Chapter 10, "Exceptions and Streams," discusses how to write robust software and shows how Java exception handling can help to achieve this goal. This chapter also explains how streams allow students to build software that, without changing, can take its input from an I/O device, a file, or a network connection.

Chapter 11 examines threads and describes how to write multithreaded code. The chapter also demonstrates how to use the `java.lang.Thread` class to create, schedule, run, and coordinate threads.

Chapter 12 introduces graphical user interfaces (GUIs) and asynchronous (event-driven) programming. The chapter uses both the Abstract Window Toolkit (`java.awt`) and Swing (`javax.swing`) packages to present these ideas. Chapter 12 cannot possibly cover everything in these two massive packages; instead, it focuses on the most important concepts in GUI design—the GUI class hierarchy, containers, components, layout, event listeners, and event handlers. By the end of the chapter, students will be able to build simple graphical interfaces and, most importantly, be prepared to read the Java documentation and learn about components that are not treated in this chapter.

Finally, Chapter 13 covers TCP/IP networking using the `java.net` package. This chapter again focuses on the key ideas of networking, such as sockets, connections, hosts, IP addresses, and datagrams. By the end of this discussion, students will be able to write simple but functional client/server applications. This chapter also introduces students to the topic of network security.

To summarize, the basic philosophy of this text is to build on the foundation laid in CS1 and present the essential concepts and techniques of modern software development. These concepts include the software life cycle, requirements and specification, object-oriented design, formal design notations, object-oriented programming, algorithmic analysis, data structures and data structure libraries, exception handling, streams, threads, graphical user interfaces, and network computing. These topics provide students with an up-to-date view of computer science and software design while addressing the concerns of the CC2001 committee for including modern software design principles and practices in introductory courses.

Although this is not a book on Java programming, we have chosen to illustrate our ideas using Java because of its enormous popularity, the expressive power of the language, and the libraries it provides to developers. No other programming language allows us to so clearly and easily present the range of topics that we cover.

We recommend that you cover Part I first, before any other material. To fully understand and appreciate the ideas in later chapters, students need a solid grasp of object-oriented concepts such as classes, interfaces, and inheritance. However, the material in Parts II and III can be presented in the order the instructor deems appropriate for a specific class. For example, instructors who want to use exception handling in their discussion of data structures could first cover Chapter 10, "Exceptions and Streams," before diving into Part II. Similarly, if instructors want students to build visual front ends for their software, they could cover Chapter 12, "Graphical User Interfaces," early in the course.

CHANGES IN THE SECOND EDITION

We were immensely gratified by the wonderful reception to the First Edition of this text. It is used at a wide range of schools, including research universities, undergraduate institutions, liberal arts colleges, and two-year colleges. We have received many enthusiastic and supportive comments about the philosophy we used in this text. The Second Edition has added a number of new pedagogical features, the most important being the switch to Java 5.0. This has allowed us to exploit the new generic capability of version 5.0 in all our code. (This text is also fully compatible with Java 6.0, which Sun was planning to release in late 2006 or early 2007.) Generic classes and methods can operate on objects of varying types, provide full compile-time safety, and avoid the constant need to do type casting. Generics are the most important new feature of Java 5.0, and they should be a standard part of any modern software project.

In addition to the change to Java 5.0, we have added the following pedagogical features:

- *Special interest boxes* throughout the text present interesting vignettes and biographies related to the main topic under discussion.
- *Challenge work exercises* present more substantive and complex assignments that you can use as course projects or optional homework for students who want a greater challenge.

SUPPLEMENTAL MATERIALS

The following supplemental materials are available when this book is used in a classroom setting. All instructor teaching tools, outlined below, are available with this book on a single CD-ROM. They are also available for download at *www.course.com*.

Electronic Instructor's Manual. The Instructor's Manual that accompanies this textbook includes:

- Additional instructional material to assist in class preparation, including suggestions for lecture topics
- Solutions to all end-of-chapter materials

PowerPoint Presentations. This book comes with Microsoft PowerPoint slides for each chapter. These slides are included as a teaching aid for classroom presentations, either to make available to students on the network for chapter review, or to be printed for classroom distribution. Instructors can add their own slides for additional topics they introduce to the class.

Source Code. All necessary source code is available for students and instructors at *www.course.com*, and is also available on the Instructor Resources CD-ROM.

Solution Files. The solution files for all programming exercises are available for instructors at *www.course.com*, and are also available on the Instructor Resources CD-ROM.

ACKNOWLEDGEMENTS

We want to thank a number of people who provided invaluable assistance in the preparation of this book, including the reviewers who helped us organize our thoughts and create a better structured and more useful text:

- Forrest Baulieu, University of Wisconsin-Green Bay
- Scott Grissom, Grand Valley State University
- Ken Martin, University of North Florida
- Arturo Sanchez, University of North Florida
- Bill Sverdlik, Eastern Michigan University

Many thanks go to the wonderful staff of Course Technology, especially Alyssa Pratt, who provided support whenever we asked for it and helped us see this project through to its successful completion, and Catherine DiMassa, who guided the book through production. We thank Dan Seiter for editing our text. Special thanks go to Michael Singleton, who provided valuable help with example code and other technical aspects of the book. To all of these people, we say many thanks for helping us with this enormous undertaking.

Paul T. Tymann
Rochester Institute of Technology

G. Michael Schneider
Macalester College