

# Preface

---

We have made a number of improvements in this second edition of the book, but the main objectives remain the same. This book is intended as an introductory text on object-oriented programming, suitable for use in a one-semester CS1 course, and assumes no prior programming experience from the students. Those who already have experience in traditional process-oriented programming languages such as C, BASIC, and others also can use this book as an introduction to object-oriented programming, graphical user interface, and event-driven programming. The two main objectives of this book are to teach

- Object-oriented programming.
- The foundations of real-world programming.

Object-orientation has become an important paradigm in all fields of computer science, and it is important to teach object-oriented programming from the first programming course. Teaching object-oriented programming is more than teaching the syntax and semantics of an object-oriented programming language. Mastering object-oriented programming means becoming conversant with the object-oriented concepts and being able to apply them effectively and systematically in developing programs. The book teaches object-oriented programming, and students will learn how to develop true object-oriented programs.

The second objective of this book is to prepare students for real-world programming. Knowing object-oriented concepts is not enough. Students must be

able to apply that knowledge to develop real-world programs. Sample programs in many introductory textbooks are too simplistic. Students rarely encounter sample programs in other textbooks that define more than three classes. But in real-world projects, programmers must use many classes from the libraries and define many classes of their own. In this book, we teach students how to use classes from the class libraries and how to define their own classes. For example, the sample program from Chapter 15 defines 10 classes and uses numerous classes from the existing class libraries.

### New Features in the Second Edition

---

We would like to take this opportunity to thank the adopters of the first edition. We especially appreciate numerous suggestions and encouraging words from the adopters and their students. For the second edition, we focused on improving the strengths of the first edition and incorporating as many suggestions as possible. Because all the suggestions cannot be bound into a single hardcopy of a book, we tried to accommodate varying needs of the adopters by placing materials on our websites. Please see the section on supporting materials for more details on the website contents.

Before we get into the features of the book, we will first highlight briefly what's new with the second edition:

1. **Use of javadoc comments.** Except for the early chapters, all sample code and programs are documented in the standardized javadoc style. The updated javabook classes are also fully documented using the javadoc comments. The HTML documentation files for the javabook classes, generated from the javadoc comments, are available from our websites.
2. **Two-color pages.** We received many accolades for our illustrations in the first edition. We improved them further by using the second color and adding a 3-D appearance. We characterize our style of explaining hard-to-grasp concepts with informative and visually appealing diagrams and figures as *visual teaching*. We believe visual teaching is the most appropriate way to teach introductory programming.
3. **New and improved javabook classes.** Two new classes are added to the javabook package: Clock and SimpleInput. The Clock class provides basic clock functions such as reading the current time, getting today's date, and providing stopwatch functions. Using the stopwatch functions, the programmer can record easily the running time of a program. For example, they can be used conveniently to compare the running time of different sorting algorithms. The second new class, SimpleInput, provides non-GUI-based input routines. A number of

adopters requested the functionality of `InputBox` for the non-GUI environment. We added this class to answer their request. In addition to the two new classes, we made a number of minor improvements to the existing classes.

4. **Swing-based javabook classes.** With the advent of the Swing classes in the Java 2 platform, the Swing-based version of the javabook package is implemented. The direct benefits of using the Swing classes include the simplified implementation of several javabook classes and new functionality such as placing an icon on a `MessageBox` object. Information on the Swing-based javabook classes can be found at our websites. Whether to use the original javabook or the Swing-based javabook depends on the extent that the instructor covers Swing classes in the course. Even if Swing classes are not covered in the course, Swing-based javabook can be used if the instructor does not plan to get into the internal workings of the javabook package.
5. **Additional topics.** Although we feel the detailed coverage of the collection classes belongs to a CS2 book, we received requests from the adopters to include a discussion on `Vector`. We concur with them that it is desirable to introduce the convenience and power of the `Vector` class to the CS1 students. The `Vector` class is described in Chapter 9. Another new topic we included in the second edition is heapsort. After moving the sorting algorithms from the old Chapter 15 to the new Chapter 10, we added heapsort to strengthen the chapter with a nonrecursive  $M\log_2 N$  sorting algorithm. Heapsort serves as a great example of a clever use of an array for storing heap nodes.
6. **Improved supporting materials.** We improved the existing supporting materials and added many new ones. Please read the Supporting Materials section on page xxiv for a detailed information on what's available from our websites.

## Major Features

---

There are many pedagogical features that make this book unique among the introductory textbooks on object-oriented programming. We will describe the major features of this book.

### Feature 1

#### Java

We chose Java for this book. Unlike C++, Java is a pure object-oriented language, and it is an ideal language to teach object-oriented programming because Java is logical and easy to program. Java's simplicity and clean design make it

one of the most easy-to-program object-oriented languages. Java does not include any complex language features that could be a roadblock for beginners in learning object-oriented concepts. Although we use Java, we must emphasize that this book is not about Java programming. As this book is about object-oriented programming, we do not cover every aspect of Java. We do, however, cover enough language features of Java to make students competent Java programmers.

## Feature 2

### The javabook Package

We provide a class library (a *package* in Java terminology) called javabook that includes a number of classes we use throughout the book. We wrote a series of articles in 1993 on how to teach object-oriented programming in the *Journal of Object-Oriented Programming* (Vol. 6, No. 1; Vol. 6, No. 4; and Vol. 6 No. 5). The core pedagogic concept we described in the series is that one must become an object user before becoming an object designer. In other words, before being able to design one's own classes effectively, one first must learn how to use predefined classes. The use of javabook is based on this philosophy.

There are many advantages in using the javabook package:

1. ***It shows students how real-world programs are developed.*** We do not develop practical programs from scratch. Instead, we use predefined classes whenever possible. One of the major benefits of object-oriented programming is the enhanced programmer productivity by reusing the existing classes. Students will get hands-on experience of code reuse by using classes from the javabook package.
2. ***It minimizes the impact of programming language syntax and semantics.*** The use of javabook classes lets students concentrate on learning concepts instead of the Java language features. We have seen many cases where novice programmers started out with a well-designed program, yet ended up with a very poorly constructed program. Often, because they do not understand the programming language fully, their design is not translated into a syntactically and semantically correct program. When they encounter an error while developing a program, instead of correcting the program code, they change their program design. Using predefined classes minimizes the impact of programming language because these predefined classes hide the complexity of underlying programming language. Students will have a much easier time implementing their program design into a working program code using the javabook classes.
3. ***It allows students to write useful programs from very early on, which helps to sustain the students' initial interest and motivation to learn.*** Without using predefined classes, students must learn far too many details of programming language before they can start writing interesting and practi-

cal programs. But before they reach that point, many of them would lose interest in programming, drowning in the boring details of language syntax and semantics. Using the predefined classes from the standard Java libraries such as `java.awt` from the beginning, however, is not practical because these classes require programming sophistication that beginning students do not possess. Easy-to-use and intuitive predefined classes such as the `javabook` classes are more appropriate for beginning programmers.

4. ***It provides a necessary foundation before students can start designing their own classes.*** The ultimate goal of learning object-oriented programming is to master the skills necessary for designing effective classes. But before being able to design such classes, students must first learn how to use existing classes. Again, teaching how to use the standard Java classes to novice programmers from the beginning is not pedagogically sound because the majority of the classes from `java.awt`, `java.io`, and others are not easy enough for beginning programmers to use. We designed the `javabook` classes with novice programmers in mind.
5. ***You can customize the javabook package to meet your needs.*** For example, there is a class called `MainWindow` in the package that serves as a top-level window of a program. You can easily extend this class to display your school's logo when this window appears on the screen. Or you can add a help menu that will list your T.A.'s office and phone numbers. You can extend other `javabook` classes as well. The `javabook` package also can be a training ground for your graduate or upper-division undergraduate students. By designing classes for the `javabook` package used by hundreds of beginning students, they will learn first hand what it takes to make classes reliable and truly reusable.

One concern raised about the use of `javabook` is whether the students would be able to write programs without using the `javabook` package. The answer is, of course, yes. The `javabook` package is not an end, but a means for students to learn the standard package. It is a stepping stone, a kind of training wheel for the standard packages. In addition to the `javabook` classes, we cover many classes from the standard Java packages such as `java.awt` and `java.io`.

The source code of all `javabook` classes is provided, and students are encouraged to study them as they are practical examples of reusable classes. After finishing Chapter 13, students can understand almost all of the `javabook` classes. We say "almost" because some of the classes in `javabook` are implemented using the standard classes that are not explained in the book. If the students take time to look up these standard classes in a reference manual, then they should be able to understand the `javabook` classes 100 percent.

**Feature 3 Full-Immersion Approach**

We adopt a full-immersion approach in which students learn how to use objects from the first program. It is very important to ensure that the core concepts of object-oriented programming are emphasized from the beginning. Our first sample program from Chapter 1 is this:

```
/*
  Program FunTime

  The program will allow you to draw a picture by
  dragging a mouse (move the mouse while holding the left mouse
  button down; hold the button on Mac). To erase the picture and
  start over, click the right mouse button (command-click on Mac).
*/

import javabook.*;

class FunTime
{
    public static void main(String[ ] args)
    {
        SketchPad  doodleBoard;
        doodleBoard = new SketchPad();
        doodleBoard.setVisible( true );
    }
}
```

This program captures the most fundamental notion of object-oriented programming. That is, an object-oriented program uses objects. As obvious as it may sound, many introductory books do not really emphasize this fact. In the program, we use a `SketchPad` object called `doodleBoard` that allows the user to draw a picture. Almost all other introductory textbooks begin with a sample program such as

```
/*
  Hello World Program
*/

class HelloWorld
{
    public static void main(String args[])
    {
        System.out.println("Hello World");
    }
}
```

or

```

/*
   Hello World Applet
*/
import java.applet.*;
import java.awt.*;

public class HelloWorld extends Applet
{
    public void paint( Graphics g)
    {
        g.drawString("Hello World", 50, 50);
    }
}

```

Both programs have problems. They do not illustrate the key concept that object-oriented programs use objects. The first program does indeed use an object `System.out`, but the use of `System.out` does not illustrate the object declaration and creation. Beginners normally cannot differentiate classes and objects. So it is very important to emphasize the concept that you need to declare and create an object from a class before you can start using the object. Our first sample program does this.

Another problem with the `System.out` program is that no real window-based programs use it for output. Some textbooks not only use `System.out` in their first program, they rely on `System.out` almost exclusively for program output. This is not real-world programming. In this book, we use `System.out` only to output data for verification purposes while developing programs.

The second `HelloWorld` program is an applet, which, as its name suggests, is a mini-application with a very specific usage. Although applets are fun, teaching applets exclusively is a problem because students will learn only a very limited view of programming. We will discuss more on applications versus applets later in the preface.

Another major problem with these two programs is that they are not adaptable to real-world situations. In contrast, our first sample program can be a main program of a commercial application by replacing `SketchPad` with another class, say, `WordProcessor`. In fact, our second sample program from Chapter 2 is this: (Note: This is the first program we actually explain line by line.)

```

/*
   Program MyFirstApplication
   The first sample Java application.
*/
import javabook.*;

```

```
class MyFirstApplication
{
    public static void main(String args[])
    {
        MainWindow mainWindow;
        mainWindow = new MainWindow();//create and
        mainWindow.setVisible( true );//display a window
    }
}
```

The structure of this program is identical to the structure of the first sample program. Our second sample program reinforces the concept that we program by using objects and by changing objects, we create a different program.

## Feature 4

### Illustrations

We believe a picture is worth a thousand words. Difficult concepts can be explained nicely with lucid illustrations. We use *object diagrams* to show the relationships among objects and classes. Diagrams are an important tool for designing and documenting programs, and no programmers will develop real-world software applications without using some form of diagramming tools. We use simple and informal diagrams, but the diagrams we use in this book are modeled after the industry standard object diagrams. After becoming comfortable with the object diagrams in this book, students are well prepared to study more formal object-oriented design methodology. For those who would like to introduce formal object diagrams, we have UML diagrams for the sample programs and javabook classes available for viewing and downloading from our websites.

This book includes numerous illustrations that are used as a pedagogic tool to explain core concepts such as inheritance, difference between private and public methods, and so forth. Notations used in the object diagrams are used consistently in all types of illustrations. Figure 1 is one example from Chapter 2, and there are over 230 such illustrations and diagrams in this book. Other representative illustrations can be found on pages 90, 165, 243, 378, 387, 428, 498, 669, and 724.

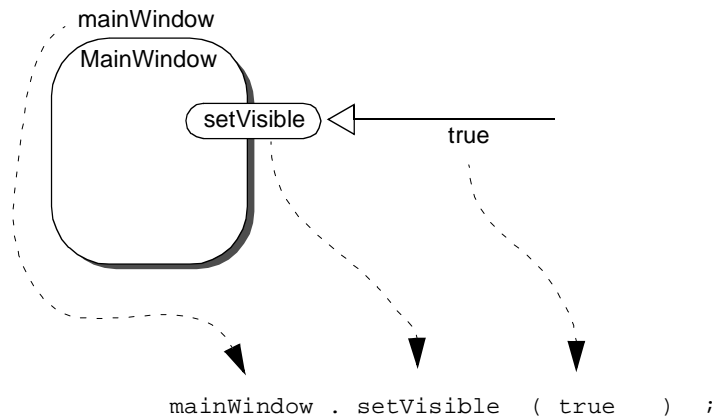
In addition to object diagrams, we use *method call sequence diagrams* that indicate the sequence of method calls such as the one shown in Figure 15.1 on page 711. The method call sequence diagrams are very useful in showing the flow of messages. We use method call sequence diagrams extensively in documenting an advanced sample program in Chapter 15.

## Feature 5

### Incremental Development

We teach object-oriented software engineering principles in this book. Instead of dedicating a separate chapter for the topic, we interleave program development principles and techniques with other topics. Every chapter from Chapter 2

**FIGURE 1** Correspondence between message sending as represented in the object diagram and in the actual Java statement.



to Chapter 14 includes at least one sample program to illustrate the topics covered in the chapter, and we develop the program using the same design methodology consistently. Chapter 15 is the case study chapter in which we develop a substantially large program for the CS1 standard.

One major problem with many of the other introductory programming books on the market today is that they teach a two-decade-old structured programming, which just does not work with object-oriented programs. This book really teaches a software design methodology that is conducive to object-oriented programming. All sample programs in this book are developed by using a technique we characterize as incremental development. The incremental development technique is based on the modern iterative approach (some call it a spiral approach), which is a preferred methodology of professional object-oriented programmers.

Beginning programmers tend to mix the high-level design and low-level coding details, and their thought process gets all tangled up. Presenting the final program is not enough. If we want to teach students how to develop programs, we must show the development process. An apprentice will not become a master builder just by looking at finished products, whether they are furniture or houses. Software construction is no different. It is often the case with other textbooks that a single chapter is dedicated to showing software development. This is not enough. We must show the development process more than just once. In this book, we develop every sample program incrementally to show students how to develop programs in a logical and methodical manner.

Source code of all sample programs at every step of development is available from our websites. However, we do not encourage students to simply fol-

low the development presented in the book and read the source code. We encourage students to actually build the sample programs following the development steps presented in the book. This is the surest and quickest way for the students to truly master the software development.

## Feature 6

### Design Guidelines, Helpful Reminders, and Quick Checks

Throughout the book, we include design guidelines and helpful reminders. Almost every section of the chapters is concluded with a number of Quick Check questions to make sure that students have mastered the basic points of the section.

Design guidelines are indicated with a pencil icon like this:



***Design a class that implements a single well-defined task. Do not overburden the class with multiple tasks.***

Helpful reminders come in different styles. The first style is indicated with a thumbtack icon like this:



***Watch out for the off-by-one error (OBOE).***

The second style is Dr. Caffeine's monologue:

Dr. Caffeine

*On occasions, programming can be very frustrating because no amount of effort on your part would make the program run correctly. You are not alone. Professional programmers often have the same feeling, including this humble self. But, if you take time to think through the problem and don't lose your cool, you will find a solution. If you don't, well, it's just a program. Your good health is much more important than a running program and a good grade.*

The third style is a dialogue between Dr. Caffeine and his honor students Ms. Latte or Mr. Espresso. Ms. Latte and Mr. Espresso appear in alternate chapters.

Dr. & Ms.

***Ms. Latte:*** *I appear in the odd-numbered chapters and ask great questions.*

***Dr. Caffeine:*** *That's right, and your questions are insightful and helpful to other students.*

Dr. & Mr.

***Mr. Espresso:*** *I appear in the even-numbered chapters and also ask questions.*

***Dr. Caffeine:*** *Yes, and I like your questions, too.*

When we have materials related to the topic covered in the book on our Web sites, we indicate this availability with the following icon:



*The document that explains the Swing classes is available from our website at [www.drcaffeine.com/additionaltopics](http://www.drcaffeine.com/additionaltopics).*

Quick Check questions appear at the end of the sections with the following banner:



### Quick Check

1. Who are the two honor students of Dr. Caffeine?
2. Name the purpose of the pencil and thumbtack icons.

## Feature 7

### Graphical User Interface and Event-Driven Programming

Since modern real-world programs are GUI-based and event-driven, we cannot skirt around them if we want to teach the foundation of real-world programming. Some may feel that GUI-based and event-driven programs are difficult topics for the beginning students to master. Although they are not trivial, we believe they can be made approachable to the beginning students by presenting the topics piecemeal. We cover in depth GUI-based and event-driven programming in Chapter 13. However, students will start writing GUI-based programs exclusively from Chapter 2 using the `javabook` classes and learn the rudiments of event handling in Chapter 5. By the time they reach Chapter 13, they are well prepared to absorb the topics. (For this reason, it is preferable to cover Chapter 5, but as we explain later, you may skip the chapter if you do not wish to cover applets.)

## Feature 8

### Applications and Applets

We teach both Java applications and applets, but we put emphasis on developing applications. Some introductory books rely heavily or exclusively on applets, but the use of applets is limited. We don't write applets exclusively in the real world, and an introductory programming book should not teach just applets. At the same time, we should not shy away from teaching applets. We should teach both. Although the use of applets is limited, writing applets is generally fun for students, and we can use applets as an effective pedagogic tool. In this book, we use applets to introduce the basics of event handling and GUI-based programming in Chapter 5.

Although we cover applets and recommend that instructors teach them, we organize the book so the instructors can skip the coverage on applets without losing the continuity.

## Supplement Materials

---

All supplement materials are available from our websites at [www.mhhe.com/wu2](http://www.mhhe.com/wu2) and [www.drcaffeine.com](http://www.drcaffeine.com). The website [www.drcaffeine.com](http://www.drcaffeine.com) is under the author's control, so it will be updated more frequently than the one maintained by McGraw-Hill. Materials available from our websites include

1. **Source code for all sample programs and the javabook classes.** For all sample programs, source code at every development step is available. Both byte code and source code are available for the javabook classes. Documentation for the javabook package in HTML format, generated from the javadoc comments, is also available.
2. **Additional software packages.** The Swing-based javabook classes are available for those who wish to use new Java GUI objects. For those who wish to put more “fun” into programming exercises, we provide the Turtle class that implements the Logo turtle. The Turtle class and the supporting classes are contained in the galapagos package.
3. **Additional topics.** To satisfy the varying needs of instructors, we place additional topics on our websites. Some of the topics available include
  - Swing classes
  - Linked lists and dynamic allocation
  - Different sample programs

We will try our best to put out new topics periodically.

4. **Hand outs on how to use Java compilers.** We have step-by-step instructions on how to edit, compile, and run the programs using the plain JDK compiler and commercial compilers such as Symantec Visual Cafe, Borland JBuilder, Metrowerks CodeWarrior, and others.
5. **Solutions to the Quick Check and selected chapter exercises.** Solutions to the Quick Check exercises are available to everybody. At present, we limit the solutions to the chapter exercises to the instructors only. Instructors can download the solutions and make them available to their students if they wish.
6. **Transparency masters.** PowerPoint slides are available in animated and nonanimated versions. Both versions of the slides can be viewed online via a web browser. Adopters of the book also can download and edit the slides to customize them for their courses.

- 7. Chapter notes and suggestions for teaching.** This material is available to the instructors who adopt the textbook.

### McGraw-Hill Online Learning Center

The book's Online Learning Center at [www.mhhe.com/wu2](http://www.mhhe.com/wu2) is a “digital cartridge” that contains the book's pedagogy and supplements. As students read the book, they can go online to take self-grading quizzes, access the source code, and review PowerPoint slides for the relevant chapter. There is also an instructor's portion of the Online Learning Center, which contains PowerPoint slides, source code, and password-protected solutions organized by chapter for easy reference.

### Book Organization

---

There are 17 chapters in this book. There are more than enough topics for one semester. Basically the chapters should be covered in linear sequence. Chapter 0 can be skipped or assigned as an outside reading if you want to jump right into programming. If you do not wish to cover applets, then you can skip Chapter 5 altogether. Section 3.8 on numerical representation and Section 7.10 on recursive methods are optional and can be skipped.

In the following, we will give a short description for each chapter:

- **Chapter 0** is an optional chapter. We provide background information on computers and programming languages. This chapter can be skipped or assigned as an outside reading if you wish to start with object-oriented programming concepts.
- **Chapter 1** provides a conceptual foundation of object-oriented programming. We describe the key components of object-oriented programming and illustrate each concept with a diagrammatic notation. The first sample program we present in this chapter is a fun drawing program.
- **Chapter 2** covers the basics of Java programming and the process of editing, compiling, and running a program. We introduce both application and applet in this chapter. We also introduce two classes—`MainWindow` and `MessageBox`—from the `javabook` package. The last section of the chapter that covers an applet can be skipped if you do not cover applets.
- **Chapter 3** introduces variables, constants, and expressions for manipulating numerical data. We explain the standard `Math` class from `java.lang` and introduce and use two more classes—`InputBox` and `OutputBox`—from the `javabook` package. These classes handle the in-

put and output of an application program. The optional section explains how the numerical values are represented in memory space.

- **Chapter 4** teaches how to define instantiable classes. The key topics covered in this chapter are constructors, visibility modifiers (public and private), local variables, parameter passing, and value-returning methods. By the end of Chapter 4, students will have a basic understanding of how to use classes from the packages and how to define their own (simple) classes.
- **Chapter 5** covers applets. We teach how to process input with applets. GUI objects `Label`, `TextField`, and `Button` from the `java.awt` package are introduced. This chapter provides a first glimpse of event handling that serves as a nice foundation for a fuller coverage of GUI objects and event-driven programming in Chapter 12. We recommend instructors cover this chapter, but we understand some prefer not to cover applets. If that is the case, this chapter can be skipped without any loss of continuity.
- **Chapter 6** explains the selection statements `if` and `switch`. We cover boolean expressions and nested-`if` statements. To illustrate the use of selection statements, the `ListBox` class from the `javabook` package is introduced and used in the sample program.
- **Chapter 7** explains the repetition statements `while`, `do-while`, and `for`. We introduce two more classes—`ResponseBox` and `Format`—from the `javabook` package and use them in the sample program. The optional last section of the chapter introduces recursion as another technique for repetition.
- **Chapter 8** covers nonnumerical data types: characters and strings. Both the `String` and `StringBuffer` classes are explained in the chapter. Using these objects, we explain the difference between primitive and reference data types. We also explain how the objects are passed as parameters to methods and how they are returned from the methods.
- **Chapter 9** teaches arrays. We cover arrays of primitive data types and of objects. Arrays are objects in Java, and we reiterate how objects are passed to methods using arrays as an example in this chapter. The self-referencing pointer `this` is introduced here. We describe how to process two-dimensional arrays and explain that a two-dimensional array is really an array of arrays in Java. The `Vector` class from the `java.util` package is introduced and explained at the end of the chapter.
- **Chapter 10** presents searching and sorting algorithms. Both  $N^2$  and  $N\log_2 N$  sorting algorithms are covered. You may skip the mathematical

analysis of searching and sorting algorithms depending on the students' background.

- **Chapter 11** explains the file I/O. Standard objects such as File and FileDialog objects from java.awt and java.io are explained. We cover all types of file I/O, from a low-level byte I/O to a high-level object I/O. This coverage of object I/O is unique among the introductory textbooks. As a part of file processing, we introduce exception handling in this chapter.
- **Chapter 12** explores reusable classes and describes package organization. In the chapter, we show how to classify objects into four categories as an aid to designing classes. As an illustration of how a class can be made reusable, we will convert the earlier sample classes into reusable classes.
- **Chapter 13** covers event-driven programming and GUI objects. The GUI objects we explain in this chapter are Button, Menu, Dialog, Frame, and Applet. GUI objects introduced in Chapter 5 are also used in this chapter. We will also teach the processing of mouse events.
- **Chapter 14** discusses inheritance and polymorphism and how to use them effectively in program design. The effect of inheritance for member accessibility and constructors is explained. We also explain the purpose of abstract classes and abstract methods.
- **Chapter 15** is a case study chapter. We will use the techniques learned in the book to develop a large program. The program uses objects from both the javabook and standard Java packages in addition to the 10 classes we design for the program. No other introductory textbooks include a sample program that is composed of as large a number of classes as this sample program.
- **Chapter 16** covers recursion. Because we want to show the examples where the use of recursion really shines, we did not include any recursive algorithm (other than those used for explanation purposes) that really should be written nonrecursively.

## Acknowledgments

---

First, I would like to thank the following reviewers for their comments, suggestions, and encouragement.

Lewis Barnett	University of Richmond
Peter Biggs	Brigham Young University
Allen Brookes	Linfield College

Edward Chow	University of Colorado, Colorado Springs
Stephanie Crouch	Texas A & M University
Ann Ford	University of Michigan
Homer C. Gerber	University of Central Florida
Michael Hoffhines	Maui Community College
Lily Hou	Carnegie Mellon University
Faisal Kaleem	Florida International University
Saroja Kanchi	Kettering University
Alan Kaplan	Clemson University
Chung Lee	California State Polytechnic University at Pomona
John Lowther	Michigan Technical University
Keith B. Olson	Montana Tech of the University of Montana
John Phillips	Dodge City Community College
Bina Ramamurthy	SUNY at Buffalo
Arturo Sanchez-Ruiz	University of Massachusetts, Dartmouth
Nan C. Schaller	Rochester Institute of Technology
Zhong Shao	Yale University
Lou Steinberg	Rutgers University
Deborah Trytten	University of Oklahoma
Allen Tucker	Bowdoin College
Paul Tymann	Rochester Institute of Technology
John D. Tvedt	The Catholic University of America
Ming Wang	Embry-Riddle Aeronautical University
Barbara Williams	North Hennepin Community College

First and foremost, I thank the instructors and their students who used the first edition. I would like to thank all those who took time to comment on the book and detect the errors. I thank Andy Duncan and Chris Steketee especially for reporting all those errors. I thank Chris Eagle for the all-around help I received in implementing the Swing version of the javabook classes. I thank Jim Skrentny of the University of Wisconsin, Madison and his graduate students for finding errors in the final manuscript.

I thank my friends at McGraw-Hill. On the production side, I thank Scott Scheidt and Pam Verros. On the sales side, I thank John Wannamacher and all the sales reps for what they're doing to sell the book. On the editorial side, I thank Emily Gray and Betsy Jones. When I speak to the editors from other publishing companies, they frequently ask me, "Are you happy with your editorial staff?" or some variation of it. My answer is invariably yes. Writing a textbook is a hard work and often a real drudgery, but it was always a pleasure to deal with Betsy and Emily.

Finally, I would like to thank my family for their love and support. Thank you, Mari and Iris.