Preface

This book is an introduction to object-oriented programming using the Java programming language. We use the object-first approach where objects are used from the first sample program. Object-oriented thinking is emphasized and promoted from the beginning. Students learn how to use objects first and then learn how to define their own objects.

Key Changes in the 5th Edition

The key differences between this edition and the fourth edition are as follows:

1. **More Discussion on Java 5.0 Features and Java 6.0 Compatibility.** Many of the new Java 5.0 features are explained and used in the sample programs. They include the enumerator type, the for-each loop construct, auto boxing and unboxing, and the generics.

2. **Exclusive Use of Console Input and Output.** All the GUI related topics, including the JOptionPane class, are moved to Chapter 14. Sample programs before Chapter 14 use the standard console input (Scanner) and output (System.out). Those who want to use JOptionPane for simple input and output can do so easily by covering Section 14.1 before Chapter 3.

3. **More Examples from Natural Sciences.** In several key chapters, we illustrate concepts using examples from biology and chemistry. For example, in Chapter 4, we use the elements in the periodic table to illustrate the concept of programmer-defined classes. In Chapter 9, we demonstrate how the string processing techniques are applied to implement DNA sequencing and other common DNA operations.

4. **Level-by-level Organization for Programming Exercises.** Programming exercises at the end of chapters are organized into three levels of difficulties. The one-star level exercises require the basic understanding of the materials covered in the chapter. The two-star level exercises require some additional thinking beyond the basic understanding. The three-star level exercises are

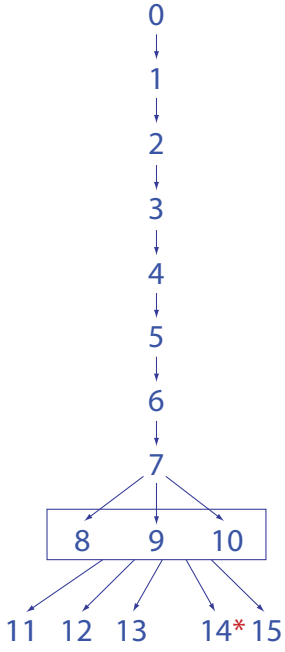
most difficult and require significant effort. For some of the three-star exercises, students must find or study additional information beyond those presented in the book. Please keep in mind that the level of difficulties is only a general guideline. One student may find some level-three exercises much easier than level-two exercises, for example.

Book Organization

There are 16 chapters in this book, numbered from 0 to 15. The first 11 chapters cover the core topics that provide the fundamentals of programming. Chapters 11 to 15 cover intermediate-level topics such as sorting, searching, recursion, inheritance, polymorphism, and file I/O. There are more than enough topics for one semester. After the first 11 chapters (Ch 0 to Ch 10), instructors can mix and match materials from Chapters 11 to 15 to suit their needs. We first show the dependency relationships among the chapters and then provide a brief summary of each chapter.

Chapter Dependency

For the most part, chapters should be read in sequence, but some variations are possible, especially with the optional chapters. Here's a simplified dependency graph:



*Note: Some examples use arrays, but the use of arrays is not an integral part of the examples. These examples can be modified to those that do not use arrays. Many topics from the early part of the chapter can be introduced as early as after Chapter 2.

Brief Chapter Summary

Here is a short description of 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 using UML.
- **Chapter 2** covers the basics of Java programming and the process of editing, compiling, and running a program. From the first sample program presented in this chapter, we emphasize object-orientation. We will introduce the standard classes String, Date, and SimpleDateFormat so we can reinforce the notion of object declaration, creation, and usage. Moreover, by using these standard classes, students can immediately start writing practical programs. We describe and illustrate console input with System.in and the Scanner class and output with System.out.
- **Chapter 3** introduces variables, constants, and expressions for manipulating numerical data. We explain the standard Math class from java.lang and introduce more standard classes (GregorianCalendar and DecimalFormat) to continually reinforce the notion of object-orientation. We describe additional methods of the Scanner class to input numerical values. Random number generation is introduced in this chapter. The optional section explains how the numerical values are represented in memory space.
- **Chapter 4** teaches the basics of creating programmer-defined classes. We keep the chapter accessible by introducting only the fundamentals with illustrative examples. The key topics covered in this chapter are constructors, visibility modifiers (public and private), local variables, and passing data to methods. We provide easy-to-grasp illustrations that capture the essence of the topics so the students will have a clear understanding of them.
- **Chapter 5** explains the selection statements if and switch. We cover boolean expressions and nested-if statements. We explain how objects are compared by using equivalence (==) and equality (the equals and compareTo methods). We use the String and the programmer-defined Fraction classes to make the distinction between the equivalence and equality clear. Drawing 2-D graphics is introduced, and a screensaver sample development program is developed. We describe the Java 5.0 feature called *enumerated type* in this chapter.
- **Chapter 6** explains the repetition statements while, do–while, and for. Pitfalls in writing repetition statements are explained. One of the pitfalls to avoid is the use of float or double for the data type of a counter variable. We illustrate this pitfall by showing a code that will result in infinite loop. Finding the greatest common divisor of two integers is used as an example of a nontrivial loop statement. We show the difference between the straightforward (brute-force) and the clever (Euclid's) solutions. We introduce the Formatter class and show

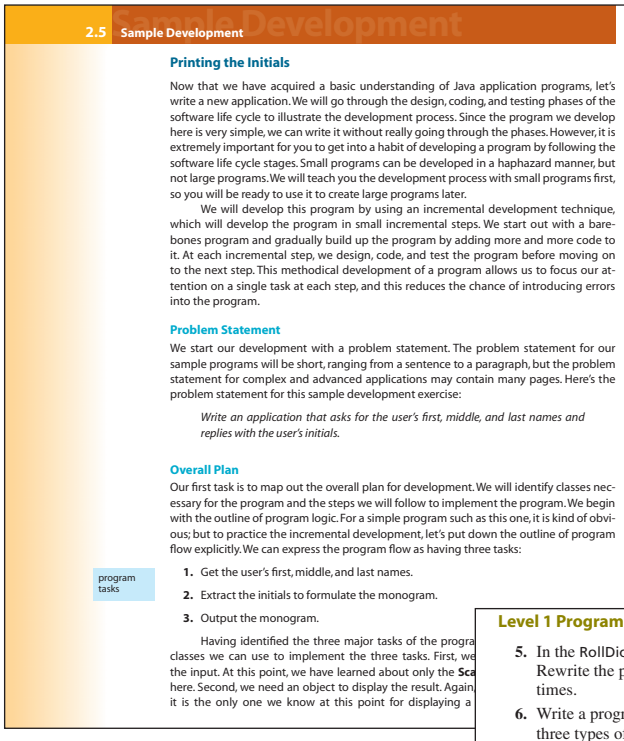
how the output can be aligned nicely. The optional last section of the chapter introduces recursion as another technique for repetition. The recursive version of a method that finds the greatest common divisor of two integers is given.

- **Chapter 7** is the second part of creating programmer-defined classes. We introduce new topics related to the creation of programmer-defined classes and also repeat some of the topics covered in Chapter 4 in more depth. The key topics covered in this chapter are method overloading, the reserved word this, class methods and variables, returning an object from a method, and pass-by-value parameter passing. As in Chapter 4, we provide many lucid illustrations to make these topics accessible to beginners. We use the Fraction class to illustrate many of these topics, such as the use of this and class methods. The complete definition of the Fraction class is presented in this chapter.
- **Chapter 8** teaches exception handling and assertions. The focus of this chapter is the construction of reliable programs. We provide a detailed coverage of exception handling in this chapter. We introduce an assertion and show how it can be used to improve the reliability of finished products by catching logical errors early in the development.
- **Chapter 9** covers nonnumerical data types: characters and strings. Both the String and StringBuffer classes are explained in the chapter. Another string class named StringBuilder is briefly explained in this chapter. An important application of string processing is pattern matching. We describe pattern matching and regular expression in this chapter. We introduce the Pattern and Matcher classes and show how they are used in pattern matching. One section is added to discuss the application of string processing in bioinformatics.
- **Chapter 10** teaches arrays. We cover arrays of primitive data types and of objects. An array is a reference data type in Java, and we show how arrays are passed to methods. We describe how to process two-dimensional arrays and explain that a two-dimensional array is really an array of arrays in Java. Lists and maps are introduced as a more general and flexible way to maintain a collection of data. The use of ArrayList and HashMap classes from the java.util package is shown in the sample programs. Also, we show how the WordList helper class used in Chapter 9 sample development program is implemented with another map class called TreeMap.
- **Chapter 11** presents searching and sorting algorithms. Both N^2 and $N\log_2 N$ sorting algorithms are covered. The mathematical analysis of searching and sorting algorithms can be omitted depending on the students' background.
- **Chapter 12** explains the file I/O. Standard classes such as File and JFileChooser are explained. We cover all types of file I/O, from a low-level byte I/O to a high-level object I/O. We show how the file I/O techniques are used to implement the helper classes—Dorm and FileManager—in Chapter 8 and 9 sample development programs. The use of the Scanner class for inputting data from a textfile is also illustrated in this chapter.

- **Chapter 13** 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 14** covers GUI and event-driven programming. Only the Swing-based GUI components are covered in this chapter. We show how to use the JOptionPane class for a very simple GUI-based input and output. GUI components introduced in this chapter include JButton, JLabel, ImageIcon, JTextField, JTextArea, and menu-related classes. We describe the effective use of nested panels and layout managers. Handling of mouse events is described and illustrated in the sample programs. Those who do not teach GUI can skip this chapter altogether. Those who teach GUI can introduce the beginning part of the chapter as early as after Chapter 2.
- **Chapter 15** 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.

Hallmark Features of the Text

Problem Solving



2.5 Sample Development

Printing the Initials

Now that we have acquired a basic understanding of Java application programs, let's write a new application. We will go through the design, coding, and testing phases of the software life cycle to illustrate the development process. Since the program we develop here is very simple, we can write it without really going through the phases. However, it is extremely important for you to get into a habit of developing a program by following the software life cycle stages. Small programs can be developed in a haphazard manner, but not large programs. We will teach you the development process with small programs first, so you will be ready to use it to create large programs later.

We will develop this program by using an incremental development technique, which will develop the program in small incremental steps. We start out with a bare-bones program and gradually build up the program by adding more and more code to it. At each incremental step, we design, code, and test the program before moving on to the next step. This methodical development of a program allows us to focus our attention on a single task at each step, and this reduces the chance of introducing errors into the program.

Problem Statement

We start our development with a problem statement. The problem statement for our sample programs will be short, ranging from a sentence to a paragraph, but the problem statement for complex and advanced applications may contain many pages. Here's the problem statement for this sample development exercise:

> *Write an application that asks for the user's first, middle, and last names and replies with the user's initials.*

Overall Plan

Our first task is to map out the overall plan for development. We will identify classes necessary for the program and the steps we will follow to implement the program. We begin with the outline of program logic. For a simple program such as this one, it is kind of obvious; but to practice the incremental development, let's put down the outline of program flow explicitly. We can express the program flow as having three tasks:

program tasks

1. Get the user's first, middle, and last names.
2. Extract the initials to formulate the monogram.
3. Output the monogram.

Having identified the three major tasks of the progra
classes we can use to implement the three tasks. First, we
the input. At this point, we have learned about only the **Sca**
here. Second, we need an object to display the result. Again,
it is the only one we know at this point for displaying a

Sample Development Programs

Most chapters include a sample development section that describes the process of incremental development.

Level-by-level Organization for Programming Exercises

Level 1 Programming Exercises ★

5. In the RollDice program, we created three Die objects and rolled them once. Rewrite the program so you will create only one Die object and roll it three times.
6. Write a program that computes the total ticket sales of a concert. There are three types of seatings: A, B, and C. The program accepts the number of tickets sold and the price of a ticket for each of the three types of seats. The total sales are computed as follows:

```
totalSales = numberOfA_Seats * pricePerA_Seat +
             numberOfB_Seats * pricePerB_Seat +
             numberOfC_Seats * pricePerC_Seat;
```

Write this program, using only one class, the main class of the program.

Define a new class named Temperature. The class has two accessors—toFahrenheit and toCelsius—that return the temperature in the specified unit and two mutators—setFahrenheit and setCelsius—that assign the temperature in the specified unit. Maintain the temperature internally in degrees Fahrenheit. Using this class, write a program that inputs temperature in degrees Fahrenheit and outputs the temperature in equivalent degrees Celsius.

Development Exercises

For the following exercises, use the incremental development methodology to implement the program. For each exercise, identify the program tasks, create a design document with class descriptions, and draw the program diagram. Map out the development steps at the start. Present any design alternatives and justify your selection. Be sure to perform adequate testing at the end of each development step.

11. In the sample development, we developed the user module of the keyless entry system. For this exercise, implement the administrative module that allows the system administrator to add and delete Resident objects and modify information on existing Resident objects. The module will also allow the user to open a list from a file and save the list to a file. Is it proper to implement the administrative module by using one class? Wouldn't it be a better design if we used multiple classes with each class doing a single, well-defined task?
12. Write an application that maintains the membership lists of five social clubs in a dormitory. The five social clubs are the Computer Science Club, Biology Club, Billiard Club, No Sleep Club, and Wine Tasting Club. Use the Dorm class to manage the membership lists. Members of the social clubs are Resident objects of the dorm. Use a separate file to store the membership list for each club. Allow the user to add, delete, and modify members of each club.

Development Exercises

give students an opportunity to practice incremental development.

Object-Oriented Approach

We take the object-first approach to teaching object-oriented programming with emphasis on proper object-oriented design. The concept of objects is clearly illustrated from the very first sample program.

```
/*
    Chapter 2 Sample Program: Displaying a Window
    File: Ch2Sample1.java
*/
import javax.swing.*;
class  Ch2Sample1 {
   public static void main(String[] args) {
       JFrame    myWindow;
       myWindow = new JFrame();
       myWindow.setSize(300, 200);
       myWindow.setTitle("My First Java Program");
       myWindow.setVisible(true);
   }
}
```

Good practices on object-oriented design are discussed throughout the book and illustrated through numerous sample programs.

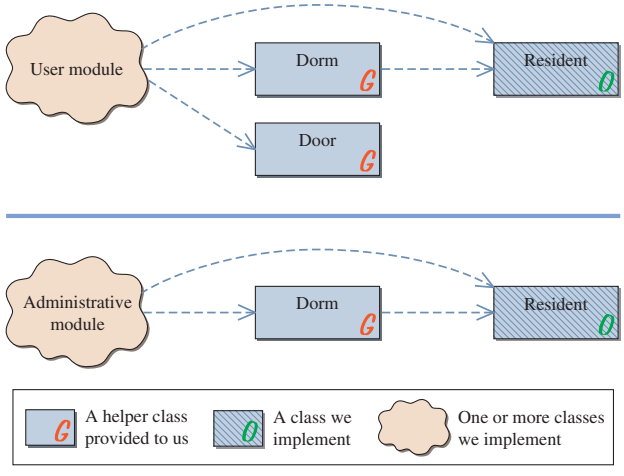


Figure 8.8 Program diagrams for the user and administrative modules. Notice the same **Dorm** and **Resident** classes are used in both programs. User and administrative modules will include one or more classes (at least one is programmer-defined).

Illustrative Diagrams

Illustrative diagrams are used to explain all key concepts of programming such as the difference between object declaration and creation, the distinction between the primitive data type and the reference data type, the call-by-value parameter passing, inheritance, and many others.

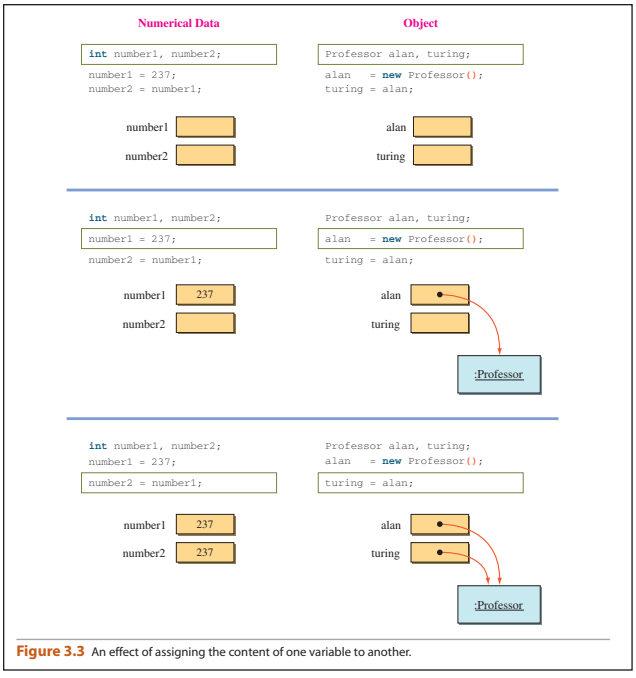


Figure 3.3 An effect of assigning the content of one variable to another.

Lucid diagrams are used effectively to explain data structures and abstract data types.

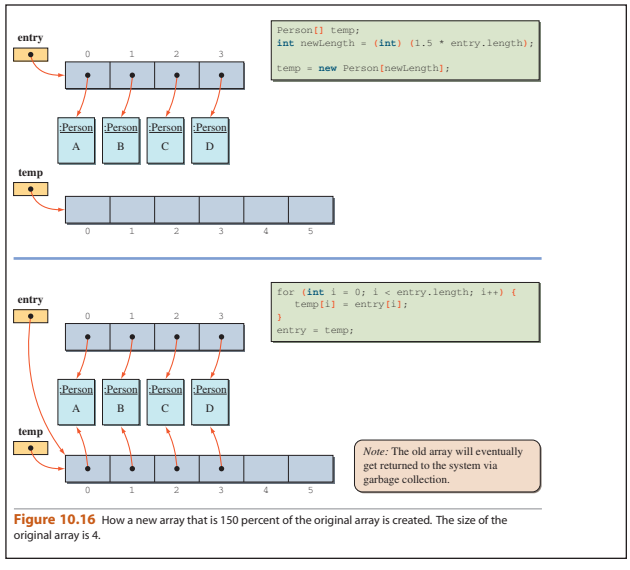


Figure 10.16 How a new array that is 150 percent of the original array is created. The size of the original array is 4.

Student Pedagogy

Design Guidelines

Always define a constructor and initialize data members fully in the constructor so an object will be created in a valid state.

Design Guidelines
provide tips on good program design.

Things to Remember
boxes provide tips for students to remember key concepts.

Things to Remember

List the **catch** *blocks in the order of specialized to more general exception classes. At most one* **catch** *block is executed, and all other* **catch** *blocks are ignored.*

Hints, & Tips, Pitfalls

It is not necessary to create an object for every variable we use. Many novice programmers often make this mistake. For example, we write

```
Fraction f1, f2;
f1 = new Fraction(24, 36);
f2 = f1.simplify( );
```

We didn't write

```
Fraction f1, f2;
f1 = new Fraction(24, 36);
f2 = new Fraction(1, 1); //not necessary
f2 = f1.simplify( );
```

because it is not necessary. The ***simplify*** method returns a ***Fraction*** object, and in the calling program, all we need is a name we can use to refer to this returned ***Fraction*** object. Don't forget that the object name (variable) and the actual object instance are two separate things.

Tips, Hints, and Pitfalls
provide important points for which to watch out.

You Might Want to Know
boxes give students interesting bits of information.

You Might Want to Know

We can turn our simulation program into a real one by replacing the **Door** class with a class that actually controls the door. Java provides a mechanism called Java Native Interface (JNI) which can be used to embed a link to a low-level device driver code, so calling the **open** method actually unlocks the door.

Quick CHECK

1. What will be displayed on the console window when the following code is executed and the user enters abc123 and 14?

```
Scanner scanner = new Scanner(System.in);
try {
   int num1 = scanner.nextInt();
   System.out.println("Input 1 accepted");
   int num2 = scanner.nextInt();
   System.out.println("Input 2 accepted");
} catch (InputMismatchException e) {
   System.out.println("Invalid Entry");
}
```

Quick Check
exercises at the end of the sections allow students to test their comprehension of topics.

Supplements for Instructors and Students

The book is supported by a rich array of supplements available through the text's website located at www.mhhe.com/wu

For Instructors, a complete set of PowerPoints, solutions to the chapter exercises, and other resources are provided.

For Students, source code for all example programs, answers to Quick Check exercises, and other resources are provided, as well as the optional galapagos package, which includes the Turtle class that is necessary in solving various chapter exercises.

Acknowledgments

I would like to thank my good friends at McGraw-Hill's editorial and production departments. Needless to say, without their help, this book would not have seen the light of the day. I thank especially Raghu Srinivasan and Lorraine Buczek for their infinite patience.

External reviewers are indispensable in maintaining the accuracy and improving the quality of presentation. Numerous professors have participated as reviewers over the course of five editions, and I thank them all again for their comments, suggestions, and encouragement. I especially thank the reviewers of the Comprehensive edition for their valuable input towards the revision of this fifth edition text.

Personal Story

In September, 2001, I changed my name for personal reasons. Prof C. Thomas Wu is now Prof Thomas W. Otani. To maintain continuity and not to confuse people, we continue to publish the book under my former name. For those who care to find out a little about my personal history can do so by visiting www.mhhe.com/wu