

A Comprehensive Introduction to Object-Oriented Programming with **Java**TM

C. Thomas Wu

Naval Postgraduate School



Higher Education

Boston Burr Ridge, IL Dubuque, IA New York San Francisco St. Louis
Bangkok Bogotá Caracas Kuala Lumpur Lisbon London Madrid Mexico City
Milan Montreal New Delhi Santiago Seoul Singapore Sydney Taipei Toronto



Higher Education

A COMPREHENSIVE INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING WITH JAVA

Published by McGraw-Hill, a business unit of The McGraw-Hill Companies, Inc., 1221 Avenue of the Americas, New York, NY 10020. Copyright © 2008 by The McGraw-Hill Companies, Inc. All rights reserved. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written consent of The McGraw-Hill Companies, Inc., including, but not limited to, in any network or other electronic storage or transmission, or broadcast for distance learning.

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

This book is printed on acid-free paper.

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

ISBN 978-0-07-352339-2

MHID 0-07-352339-9

Publisher: *Alan R. Apt*

Executive Marketing Manager: *Michael Weitz*

Senior Project Manager: *Sheila M. Frank*

Lead Production Supervisor: *Sandy Ludovissy*

Associate Media Producer: *Christina Nelson*

Designer: *Rick D. Noel*

Cover Designer: *Elise Lansdon*

(USE) Cover Image: *breaking wave on foaming ocean surface, ©Ron Dahlquist/Getty Images*

Compositor: *ICC Macmillan Inc.*

Typeface: *10.5/12 Times Roman*

Printer: *R. R. Donnelley Crawfordsville, IN*

Library of Congress Cataloging-in-Publication Data

Wu, C. Thomas.

A comprehensive introduction to object-oriented programming with Java / C. Thomas

Wu. — 1st ed.

p. cm.

ISBN 978-0-07-352339-2 — ISBN 0-07-352339-9

1. Object-oriented programming (Computer science) 2. Java (Computer program language) I. Title.

QA76.64.W77 2008

005.1'17—dc22

2006048064

To my family



Contents

Preface		xiii
Key Differences from the Standard Edition		xiii
Book Organization		xiv
Hallmark Features of the Text		xviii
0	Introduction to Computers and Programming Languages	1
0.1	A History of Computers	2
0.2	Computer Architecture	4
0.3	Programming Languages	11
0.4	Java	12
1	Introduction to Object-Oriented Programming and Software Development	15
1.1	Classes and Objects	16
1.2	Messages and Methods	18
1.3	Class and Instance Data Values	20
1.4	Inheritance	23
1.5	Software Engineering and Software Life Cycle	24

2 Getting Started with Java 29

- 2.1 The First Java Program 30
- 2.2 Program Components 39
- 2.3 Edit-Compile-Run Cycle 49
- 2.4 Sample Java Standard Classes 52
- 2.5 Sample Development 67

3 Numerical Data 81

- 3.1 Variables 82
- 3.2 Arithmetic Expressions 90
- 3.3 Constants 95
- 3.4 Displaying Numerical Values 97
- 3.5 Getting Numerical Input 103
- 3.6 The **Math** Class 109
- 3.7 Random Number Generation 113
- 3.8 The **GregorianCalendar** Class 115
- 3.9 Sample Development 120
- 3.10 Numerical Representation (*Optional*) 131

4 Defining Your Own Classes—Part 1 145

- 4.1 First Example: Defining and Using a Class 146
- 4.2 Second Example: Defining and Using Multiple Classes 156
- 4.3 Matching Arguments and Parameters 160
- 4.4 Passing Objects to a Method 162
- 4.5 Constructors 167
- 4.6 Information Hiding and Visibility Modifiers 172
- 4.7 Class Constants 175
- 4.8 Local Variables 183
- 4.9 Calling Methods of the Same Class 185
- 4.10 Changing Any Class to a Main Class 189
- 4.11 Sample Development 190

5 Selection Statements 213

5.1	The if Statement	214
5.2	Nested if Statements	225
5.3	Boolean Expressions and Variables	231
5.4	Comparing Objects	239
5.5	The switch Statement	244
5.6	Drawing Graphics	248
5.7	Enumerated Constants	258
5.8	Sample Development	264

6 Repetition Statements 295

6.1	The while Statement	296
6.2	Pitfalls in Writing Repetition Statements	305
6.3	The do-while Statement	311
6.4	Loop-and-a-Half Repetition Control	315
6.5	The for Statement	319
6.6	Nested for Statements	324
6.7	Formatting Output	326
6.8	Loan Tables	331
6.9	Estimating the Execution Time	334
6.10	Recursive Methods (<i>Optional</i>)	338
6.11	Sample Development	343

7 Defining Your Own Classes—Part 2 365

7.1	Returning an Object from a Method	366
7.2	The Reserved Word this	370
7.3	Overloaded Methods and Constructors	378
7.4	Class Variables and Methods	383

7.5	Call-by-Value Parameter Passing	387
7.6	Organizing Classes into a Package	394
7.7	Using Javadoc Comments for Class Documentation	395
7.8	The Complete Fraction Class	400
7.9	Sample Development	410

8

Exceptions and Assertions

437

8.1	Catching Exceptions	438
8.2	Throwing Exceptions and Multiple catch Blocks	445
8.3	Propagating Exceptions	450
8.4	Types of Exceptions	458
8.5	Programmer-Defined Exceptions	461
8.6	Assertions	463
8.7	Sample Development	469

9

Characters and Strings

487

9.1	Characters	488
9.2	Strings	491
9.3	Pattern Matching and Regular Expression	502
9.4	The Pattern and Matcher Classes	509
9.5	Comparing Strings	513
9.6	StringBuffer and StringBuilder	515
9.7	Sample Development	521

10

Arrays and Collections

543

10.1	Array Basics	544
10.2	Arrays of Objects	555
10.3	The For-Each Loop	565

10.4	Passing Arrays to Methods	569
10.5	Two-Dimensional Arrays	576
10.6	Lists and Maps	583
10.7	Sample Development	596

11 Sorting and Searching 619

11.1	Searching	620
11.2	Sorting	624
11.3	Heapsort	632
11.4	Sample Development	645

12 File Input and Output 669

12.1	File and JFileChooser Objects	670
12.2	Low-Level File I/O	679
12.3	High-Level File I/O	684
12.4	Object I/O	693
12.5	Sample Development	700

13 Inheritance and Polymorphism 713

13.1	A Simple Example	714
13.2	Defining Classes with Inheritance	717
13.3	Using Classes Effectively with Polymorphism	721
13.4	Inheritance and Member Accessibility	724
13.5	Inheritance and Constructors	729
13.6	Abstract Superclasses and Abstract Methods	733
13.7	Inheritance versus Interface	738
13.8	Sample Development	739

14 GUI and Event-Driven Programming 765

14.1	Simple GUI I/O with JOptionPane	768
14.2	Customizing Frame Windows	771
14.3	GUI Programming Basics	777
14.4	Text-Related GUI Components	787
14.5	Layout Managers	798
14.6	Effective Use of Nested Panels	808
14.7	Other GUI Components	817
14.8	Menus	835
14.9	Handling Mouse Events	839

15 Recursive Algorithms 859

15.1	Basic Elements of Recursion	860
15.2	Directory Listing	861
15.3	Anagram	863
15.4	Towers of Hanoi	866
15.5	Quicksort	868
15.6	When Not to Use Recursion	873

16 Memory Allocation Schemes and Linked Data Structures 879

16.1	Contiguous Memory Allocation Scheme	881
16.2	Noncontiguous Memory Allocation Scheme	886
16.3	Manipulating Linked Lists	890
16.4	Linked Lists of Objects	903
16.5	Sample Development	908

17 Generics and Type Safety 945

- 17.1 Generic Classes 946
- 17.2 Generics and Collections 961
- 17.3 Generics, Inheritance, and Java Interface 969
- 17.4 Additional Topics and Pitfalls 974

18 List ADT 981

- 18.1 The List ADT 982
- 18.2 The List Interface 988
- 18.3 The Array Implementation of the List ADT 992
- 18.4 The Linked-List Implementation of the List ADT 1001
- 18.5 The Linked Implementation with the Head Node 1018
- 18.6 The Iterator Design Pattern 1022
- 18.7 Sample Development 1027

19 Stack ADT 1035

- 19.1 The Stack ADT 1036
- 19.2 The Stack Interface 1040
- 19.3 The Array Implementation 1042
- 19.4 The Linked-List Implementation 1047
- 19.5 Implementation Using **NPSList** 1052
- 19.6 Sample Applications: Matching HTML Tags 1053
- 19.7 Sample Applications: Solving a Maze with Backtracking 1060

20	Queue ADT	1069
20.1	The Queue ADT	1070
20.2	The Queue Interface	1073
20.3	The Array Implementation	1075
20.4	The Linked-List Implementation	1082
20.5	Implementation Using NPSList	1088
20.6	Priority Queue	1089
Appendix A		1099
Appendix B		1107
Appendix C		1133
Appendix D		1155
Index		1163

Preface

This book is an in-depth introduction to object-oriented programming using the Java programming language. In addition to covering traditional topics for a CS1 course, some of the more advanced topics such as recursion and linked lists are included to provide a comprehensive coverage of beginning to intermediate-level materials. There are more materials in the book than what are normally covered in a typical CS1 course. An instructor may want to teach some of the chapters on data structures in an advanced CS1 course. Topics covered in Chapters 16 to 20 are also suitable for use in a CS2 course.

Key Differences from the Standard Edition

This comprehensive edition is based on *An Introduction to Object-Oriented Programming with Java*, Fourth Edition. The key differences between this comprehensive version and the fourth edition standard version are as follows:

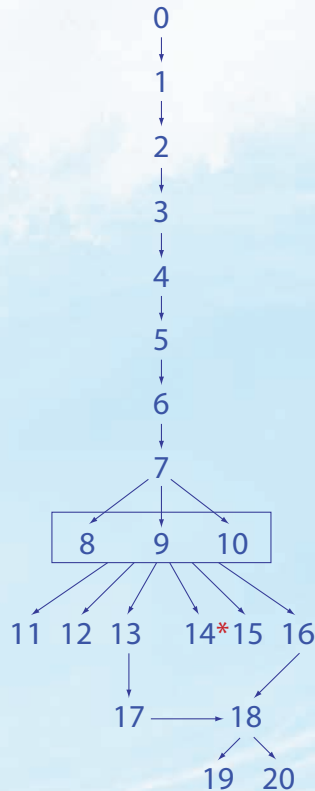
- 1. Data Structures Chapters.** Chapter 16 covers topics on managing linked nodes. Using this as the foundation, Chapters 18 through 20 present three abstract data types (ADTs) List, Stack, and Queue, respectively. For all three ADTs, both array-based and linked-list implementations are shown, and their relative advantages and disadvantages are discussed.
- 2. More Discussion on Java 5.0 Features.** 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. One complete chapter (Chapter 17) is dedicated to the generics.
- 3. 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.

Book Organization

There are 21 chapters in this book, numbered from 0 to 20. 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. And Chapters 16 to 20 cover topics related to data structures. 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 20 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 new `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 introducing 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 new 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 (new to Java 5.0) 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` (new to Java 5.) 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.
- **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 $M\log_2 N$ 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.
- **Chapter 16** covers contiguous and noncontiguous memory allocation schemes and introduces the concept of linked lists. Ample examples are provided to illustrate the manipulation of linked lists of primitive data types and linked lists of objects. This chapter lays the necessary foundation for the students to learn different techniques for implementing the abstract data types covered in Chapters 18 through 20.
- **Chapter 17** covers new Java 5.0 generics in detail. The chapter describes how generic classes are defined and how the type safety is supported by generics. A concrete example of using generics is shown by defining a simple linked list with generic nodes.
- **Chapter 18** introduces the concept of abstract data types (ADT) and covers the List ADT. Key features of the List ADT are explained and two implementations using an array and a linked list are shown. The iterator pattern to traverse the elements in the List ADT is introduced.
- **Chapter 19** covers the Stack ADT. Key features of the Stack ADT are explained and two implementations using an array and a linked list are shown. Sample applications that use stacks are described.
- **Chapter 20** covers the Queue ADT. Key features of the Stack ADT are explained and two implementations using an array and a linked list are shown. A special type of queue called a *priority queue* is also introduced in this chapter.

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:

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 program, we will now identify the classes we can use to implement the three tasks. First, we need an object to handle the input. At this point, we have learned about only the **Scanner** class, so we will use it here. Second, we need an object to display the result. Again, we will use **System.out**, as it is the only one we know at this point for displaying a string value. For the string

program
tasks

Sample Development Programs

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

Development Exercises give students an opportunity to practice incremental development.

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.

8. 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?
9. 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

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.

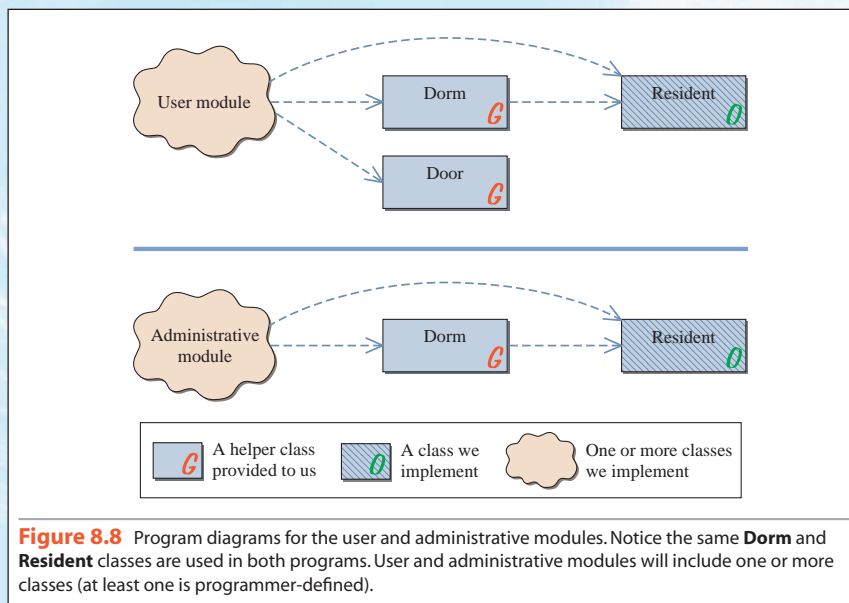
```

/*
   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);
    }
}

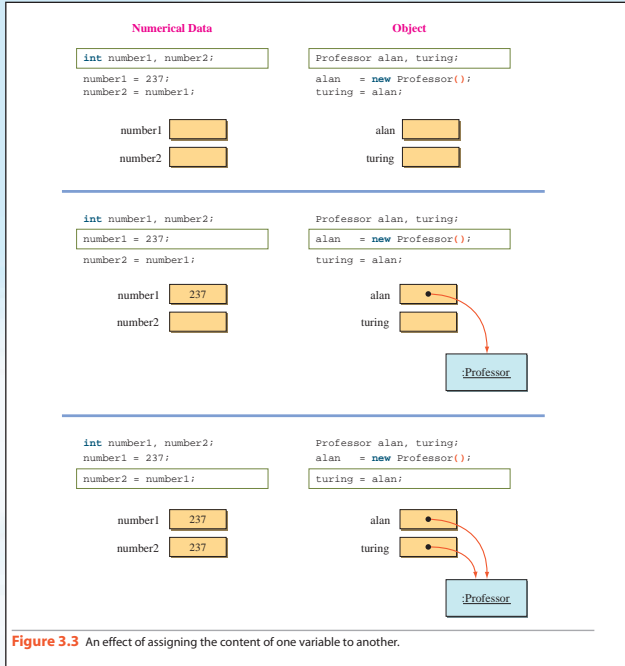
```

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



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.



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

