大学计算机教育丛书（影印版）

An Introduction to Object-Oriented Programming with

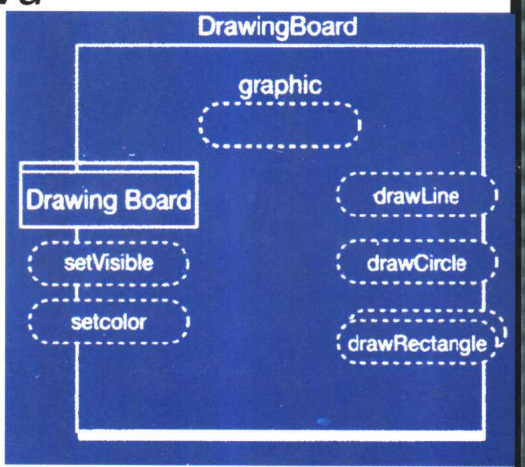
JAVA

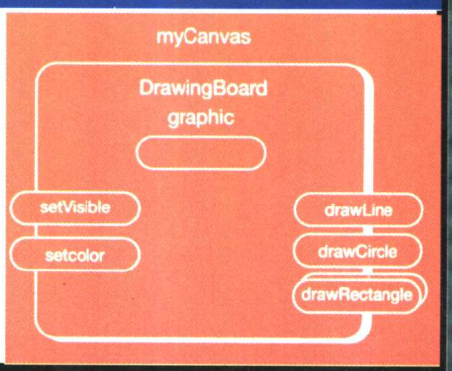
second edition

C. Thomas Wu

JAVA 面向对象程序设计

第2版





清华大学出版社
http://www.tup.tsinghua.edu.cn
McGraw-Hill
http://www.mhhe.com


Java
JAVA

An Introduction to Object-Oriented Programming with Java

second edition

Java 面向对象程序设计

第 2 版

C. Thomas Wu

Naval Postgraduate School

清华大学出版社

McGraw-Hill Companies, Inc.

(京)新登字 158 号

An Introduction to Object-Oriented Programming with Java 2nd ed.
C. Thoms Wu

北京市版权局著作权合同登记号:01-2000-3920

图书在版编目(CIP)数据

Java 面向对象程序设计:第 2 版:英文/吴(Wu,T.C.)著. -2 版(影印版). -北京:清华大学出版社,2001.1
(大学计算机教育丛书)
ISBN 7-302-04220-9

Ⅰ.J… Ⅱ.吴… Ⅲ.面向对象语言-程序设计-英文 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2000)第 83749 号

出版者:清华大学出版社(北京清华大学学研大厦,邮编 100084)
http://www.tup.tsinghua.edu.cn
印刷者:清华大学印刷厂
发行者:新华书店总店北京发行所
开 本:787×960 1/16 印张:56.25
版 次:2001 年 1 月第 1 版 2001 年 1 月第 1 次印刷
书 号:ISBN 7-302-04220-9/TP·2480
印 数:0001~3000
定 价:72.00 元

出版者的话

今天，我们的大学生、研究生和教学、科研工作者，面临的是一个国际化的信息时代。他们将需要随时查阅大量的外文资料；会有更多的机会参加国际性学术交流活动；接待外国学者；走上国际会议的讲坛。作为科技工作者，他们不仅应有与国外同行进行口头和书面交流的能力，更为重要的是，他们必须具备极强的查阅外文资料获取信息的能力。有鉴于此，在原国家教委所颁布的“大学英语教学大纲”中有一条规定：专业阅读应作为必修课程开设。同时，在大纲中还规定了这门课程的学时和教学要求。有些高校除开设“专业阅读”课之外，还在某些专业课拟进行英语授课。但教、学双方都苦于没有一定数量的合适的英文原版教材作为教学参考书。为满足这方面的需要，我们陆续精选了一批国外计算机科学方面最新版本的著名教材，进行影印出版。我社获得国外著名出版公司和原著作者的授权将国际先进水平的教材引入我国高等学校，为师生们提供了教学用书，相信会对高校教材改革产生积极的影响。

我们欢迎高校师生将使用影印版教材的效果、意见反馈给我们，更欢迎国内专家、教授积极向我社推荐国外优秀计算机教育教材，以利我们将《大学计算机教育丛书（影印版）》做得更好，更适合高校师生的需要。

清华大学出版社

《大学计算机教育丛书（影印版）》项目组

1999.6

Class Declaration Summary

main class p. 56
main method p. 57
object declaration p. 40
object creation p. 42
constant p. 98
visibility modifier p. 157
variables p. 84
arrays p. 411
this p. 439
constructor p. 152
array creation p. 413
String p. 359
primitive data types p. 85
try-catch block p. 541
file I/O p. 520
while p. 290
for p. 308
do-while p. 299
if-then-else p. 230
if-then p. 236
switch p. 252
exceptions p. 528
System.out p. 175

```
class SampleMain
{
    public static void main( String args[] )
    {
        Sample   sample1;
        sample1 = new Sample();
        sample1.readFile();
    }
}
class Sample
{
    public      static final int DEFAULT_SIZE = 10;

    private     String    filename;
    private     int       javaCnt;
    private     String[]  value;

    public Sample( )
    {
       this( "sample.txt" , DEFAULT_SIZE );
    }
    public Sample( String filename , int size )
    {
       this.filename = filename;
       value = new String[size];
    }
    public void readFile( )
    {
        String      inputLine;
        int         cnt = 0;

        try {
          File            inFile     = new File(filename);
          FileReader      fileReader = new FileReader(inFile);
          BufferedReader  bufReader  = new BufferedReader(fileReader);

          inputLine = bufReader.readLine();
          while ( !inputLine.equalsIgnoreCase ("END") ) {
                if (inputLine.equalsIgnoreCase("JAVA") {
                    javaCnt++;
                }
                else {
                   value[cnt] = inputLine;
                   cnt++;
                }
                inputLine = bufReader.readLine();
          } //while
          bufReader.close();
        }
        catch (IOException e) {
            System.out.println("Error in input");
        }
    }
}
```

import statement p. 53
java.awt p. 54
javabook p. 53
ActionListener p. 199
Button p. 605
TextField p. 618
Frame p. 605
Dialog p. 636
MenuItem p. 620
MenuBar p. 620
Menu p. 620
super p. 562
null p. 206
layout manager p. 197
protected p. 659
insets p. 779
actionPerformed p. 613
action event p. 611
mouse event p. 626
window event p. 614

Class Declaration Summary

```
import java.awt.*;

import javabook.*;

class SampleDialog extends JavaBookDialog implements ActionListener
{

   private     TextField     editBox;
   private     Button        button;
   private     Frame         myOwner;


   public SampleDialog(Frame owner)
   {
      //set the properties of the dialog
      super(owner);
      setTitle("Sample Dialog");
      setResizable(false);
      setLayout(null);
      myOwner = owner;

      //create and add GUI objects to the dialog
      editBox = new TextField("");
      button = new Button("CLEAR");
      add(editBox);
      add(okButton);

      //set the dialog as an action listener
      okButton.addActionListener(this);
   }

   protected void adjustSize()
   {
      addNotify();

      Insets inset = getInsets();

      setSize(inset.left + inset.right + 150,
                inset.top + inset.bottom + 100);

      editBox.setBounds (inset.left + 30, inset.top + 25, 150, 20);
      button.setBounds(inset.left + 75, inset.top + 155, 50, 25);
   }

   public void actionPerformed(ActionEvent event)
   {
      //tell the owner frame the value entered by the user
      myOwner.valueEntered(editBox.getText() );

      editBox.setText("");      //clears the entry
   }

}
```

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 $N\log_2 N$ 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.

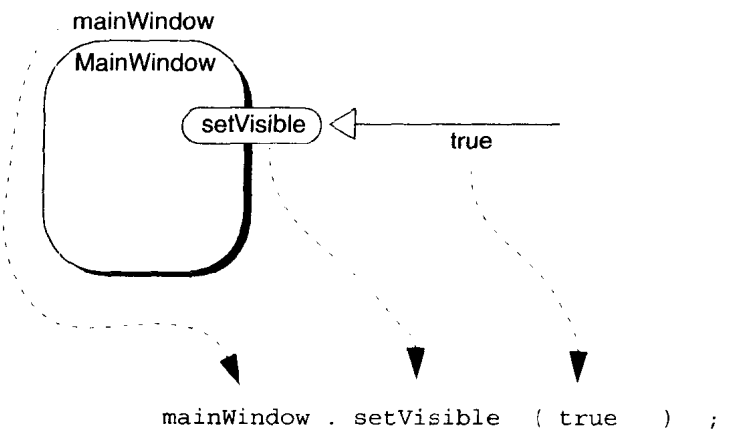
This book includes numerous illustrations that are used as a pedagogic tool to explain core concepts such as inheritance, difference between private and public methods, and so forth. Notations used in the object diagrams are used consistently in all types of illustrations. Figure 1 is one example from 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 programing, 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:

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.

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.*

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

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