国外计算机科学经典教材(影印版)

THOMSON

JAVA

for Engineers and Scientists

JAVA编程原理

——面向工程和科学人员

(影印版)

(美) Gary J. Bronson 著

清华大学出版社

Gary J. Bronson
JAVA for Engineers and Scientists
EISBN: 0-534-38453-6

北京市版权局著作权合同登记号　图字：01-2003-4325

本书封面贴有清华大学出版社激光防伪标签，无标签者不得销售。

图书在版编目(CIP)数据

JAVA 编程原理——面向工程和科学人员=JAVA for Engineers and Scientists / (美) 布朗松(Gary J. Bronson)著. —影印本. — 北京：清华大学出版社，2003
ISBN 7-302-07271-X
I. J… II.布… III. JAVA 语言—程序设计—英文 IV. TP312
中国版本图书馆 CIP 数据核字(2003)第 089613 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**客户服务**：010-62776969
组稿编辑：曹康
文稿编辑：于平
封面设计：孔祥丰
印 刷 者：北京市清华园胶印厂
装 订 者：化甲屯小学装订二厂
发 行 者：新华书店总店北京发行所
开　　本：185×260　**印张**：46.25　**字数**：1183 千字
版　　次：2003 年 11 月第 1 版　2003 年 11 月第 1 次印刷
书　　号：ISBN 7-302-07271-X/TP・5278
印　　数：1～3000
定　　价：78.00 元

Preface

Java™ is rapidly emerging as one of the preeminent applications languages for Windows® -based systems. A major reason is that Java is a true object-oriented language that now provides a complete set of visual objects that can easily be assembled into a working graphical user interface (GUI—pronounced goo-eey) and a complete set of input/output and mathematical classes.

From both a teaching and learning viewpoint, Java requires familiarity with four elements, all of which are required for object-oriented graphical-based programming. These are:

- The concept of object-oriented program class code
- The visual objects required in creating a graphical user interface
- The input, output, and mathematical classes required for creating engineering and scientific programs
- The concept of event-based programming, where the user, rather than the programmer, determines the sequence of operations to be executed.

The major objective of this textbook is to introduce each of these elements, within the context of sound programming principles, in a manner that is accessible to the beginning programmer. Its purpose is both to provide you with the tools, techniques, and understanding necessary to create and maintain Java programs as well as to prepare a solid foundation for more advanced work. Thus, the basic goal of this text is that all topics be presented in a clear and unambiguous manner appropriate to a student taking an introductory course in Java programming.

In using this text, no prerequisites are assumed. The large numbers of examples and exercises in the text are drawn from basic engineering and scientific disciplines and are appropriate to an introductory language-based course.

Distinctive Features

Writing Style

I firmly believe that for a textbook to be useful it must provide a clearly defined supporting role to the leading role of the professor. Once the professor sets the stage, however, the textbook must encourage, nurture, and assist the student in acquiring and owning the material presented in class. To do this, the text must be written in a manner that makes sense to the student. Thus, first and foremost, I feel that the writing style used to convey the concepts presented is the most important and distinctive aspect of the text.

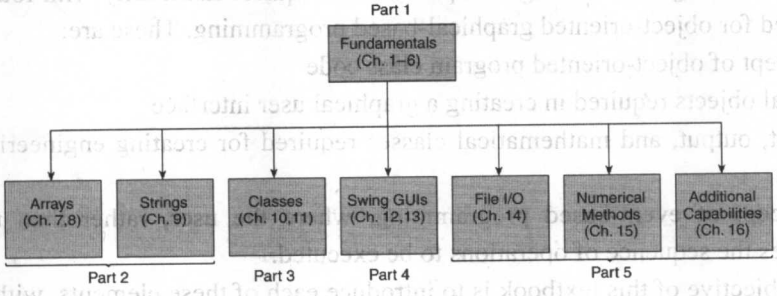
Flexibility

To be an effective teaching resource, this text is meant to provide a flexible tool that each professor can use in a variety of ways depending on *how many* programming concepts and programming techniques are to be introduced in a single course and *when* they are to be introduced. This is accomplished by partitioning the text into five parts and providing a varied number of Chapter Supplements that contain enrichment and breadth material.

Part One presents the fundamental object-oriented structure and procedural elements of Java.

Additionally, both keyboard and dialog-based data entry are presented. This permits an early introduction of the swing package of visual objects as well as providing a firm grounding in basic Java Development Kit (JDK) techniques.

Once Part One is completed, the material in Parts Two, Three, Four, and Five are *interchangeable*. For example, in a more traditional introduction to programming type of course, Part One would be followed by Chapter 14 (File I/O) and Part Two, Array and String Reference Types. However, if a requirement is that the course must emphasize class design and development, Part One would be followed by Part Three. In a third instance, if the course is to have a more visual and GUI-based slant, Part One can just as easily be followed by Part Four. In each of these cases, a "pick-and-choose" approach to course structure can be implemented. This flexibility of topic introduction is illustrated by the following topic dependence chart.



Software Engineering

Although this is primarily an introduction to Java text, as opposed to a CS1 introduction to programming book, the text is meant to familiarize students with the fundamentals of software engineering from both a procedural and object-oriented viewpoint. In most cases, however, the more general programming aspects are interwoven within the text's main language component precisely because the text is meant to introduce and strengthen the *why* as well as the *how*.

Applications

Starting with Chapter 2, the majority of chapters contain an Applications section, with an average of two completed applications per chapter. Each application demonstrates effective problem solving within the context of a complete program solution. This is done both for method and class design.

Program Testing

Every single Java program in this text has been successfully entered and executed using Sun® Java 2.0 (both Versions 1.2 and 1.3). The Brooks/Cole Web site (see Appendix J) provides all programs that are included within the text. This will permit students to both experiment and extend the existing programs and more easily modify them as required by a number of end of section exercises.

Pedagogical Features

To facilitate the goal of making Java accessible as a first-level course, the following peda-gogical features have been incorporated into the text.

End-of-Section Exercises

Almost every section in the book contains numerous and diverse skill-builder and programming

exercises. Additionally, solutions to selected odd-numbered exercises are provided on the Web site www.brookscole.com.

Common Programming Errors and Chapter Review

Each chapter provides a section on common programming errors. Additionally, each chapter contains an end of chapter section that reviews the main topics covered in the chapter.

Chapter Supplement Sections

Given the many different emphases that can be applied in teaching Java, a number of basic and enrichment topics have been included. These sections vary between such basic material as understanding bits and bytes, practical material, such as formatting, and theoretical material, such as insides and outsides. The purpose of these sections is to provide flexibility as to the choice of which topics to present and the timing of when to present them.

Point of Information Notes

These shaded boxes are primarily meant as a reference for commonly used tasks, for quick reference, for highlighting professional programming techniques, and to provide additional concept material.

Appendixes and Supplements

An expanded set of appendixes is provided. These include appendixes on key-words, operator precedence, Unicode codes, and packages. Additionally, the Web site www.brookscole.com contains solutions to selected odd-numbered exercises.

Acknowled Gments

This book began as an idea. It became a reality only due to the encouragement, skills, and efforts supplied by many people. I would like to acknowledge their contributions.

First and foremost, I wish to acknowledge and thank Kallie Swanson, my editor at Brooks/Cole, for her continuous faith and encouragement that I would complete the text no matter how many missed deadlines I went through.

As the manuscript was being developed, the skills and efforts of many other people at Brooks/Cole were required. I am very grateful and thank Carla Vera for handling numerous scheduling and review details that permitted me to concentrate on the actual writing of the text. I especially wish to express my very deep gratitude to the following individual reviewers: Sedar Kirli, University of Florida; Rubin H. Landau, Oregon State University; Chi N. Thai, University of Georgia; Guoliang (Larry) Xue, Arizona State University.

Each of these reviewers supplied extremely detailed and constructive reviews of both the original manuscript and a number of revisions. Their suggestions, attention to detail, and comments were extraordinarily helpful as the manuscript evolved and matured through the editorial process. Any errors that may now appear in the text are clearly and solely my own responsibility. Should you find any, I would very much like to hear from you through my editor, Kallie Swanson, at Brooks/Cole.

Special acknowledgement goes to G.J. Borse of Lehigh University for the material presented in Chapter 15 and the information contained within the Career Choice boxes, which Dr. Borse graciously permitted me to use from his FORTAN 77 and Numerical Methods for Engineers, Second

Edition.[1]

Once the review process was completed, the task of turning the final manuscript into a textbook again required a dedicated production staff. For this I especially want to thank Kelsey McGee, the production editor at Brooks/Cole. Kelsey and I have now worked on the majority of my published books, and her attention to detail and very high standards have helped immensely to improve the quality of all of my texts. Next, I would like to thank Frank Hubert, the copyeditor, and the team at The Book Company, especially Dustine Friedman. Once this text moved to the production stage, this team of people seemed to take personal ownership of the text, and I am very grateful to them.

I would also like to gratefully acknowledge the direct encouragement and support of Fairleigh Dickinson University. Specifically, this includes the constant encouragement, support, and positive academic climate provided by my Dean, Dr. David Steele, my Associate Dean, Dr. Ron Heim, and my Chairperson, Dr. YoungBoem Kim. Without their support, this text could not have been written.

Finally, I deeply appreciate the patience, understanding, and love provided by my wife, friend, and partner, Rochelle.

Gary Bronson

1 G.J.Borse, FORTAN 77 and Numerical Methods for Engineers, 2nd .Ed, © 1991, Brooks/Cole, an imprint of Thomson Learning.

Contents

PART 1 Fundamentals

PART 3 Creating Classes

PART 4 Creating Swing-Based GUIs

PART 5 Additional Programming Topics

PART 1

Fundamentals

CHAPTER

1

Getting Started

In this chapter, we provide both a brief background on programming languages and a specific structure that will be used throughout the text for constructing Java programs. Additionally, we explain the concepts of methods and classes. We then describe two specific methods provided in Java, print() and println(), which are used within the context of a complete program for displaying data on a video screen. Finally, we present and use the showMessageDialog() method within the context of a complete program to construct a simple graphical user interface (GUI).

1.1 Introduction to Programming

A computer is a machine and like other machines, such as an automobile or lawn mower, it must be turned on and then driven, or controlled, to perform its intended task. In an automobile, for example, control is provided by the driver, who sits inside and directs the car. In a computer, the driver is a set of instructions, called a program. More formally, a computer program is a self-contained set of instructions used to operate a computer to produce a specific result. Another term for a program or set of programs is software, and we will use both terms interchangeably throughout the text.

The process of writing a program, or software, is called programming, and the set of instructions that can be used to construct a program is called a programming language. Available programming languages come in a variety of forms and types.

Machine Language

At its most fundamental level, the only programs that can actually operate a computer are machine language programs. Such programs, which are also referred to as executable programs, or

executables for short, consist of a sequence of instructions composed of binary numbers such as:[1]

```
11000000 000000000001 000000000010
11110000 000000000010 000000000011
```

Such machine language instructions consist of two parts: an instruction part and an address part. The instruction part, which is referred to as the opcode (short for operation code), is usually the leftmost set of bits in the instruction. It tells the computer the operation to be performed, such as add, subtract, multiply, and so on. The rightmost bits specify the memory addresses of the data to be used. For example, assuming that the 8 leftmost bits of the first instruction just listed are the operation code to add and the next two groups of 12 bits are the addresses of the two operands to be added, this instruction would be a command to "add the data in memory location 1 to the data in memory location 2."[2] Similarly, assuming that the opcode 111100000 means multiply, the next instruction is a command to "multiply the data in memory location 2 by the data in memory location 3."

Assembly Languages

Although each class of computers, such as IBM personal computers and Macintosh computers, have their own particular machine language, it is very tedious and time consuming to write machine language programs. One of the first advances in programming was the substitution of wordlike symbols, such as ADD, SUB, MUL, for the binary opcodes and both decimal numbers and labels for memory addresses. For example, using these symbols and decimal values for memory addresses, the previous two machine language instructions can be written as:

```
ADD 1, 2
MUL 2, 3
```

Programming languages that use this type of symbolic notation are referred to as assembly languages. Since computers can only execute machine language programs, the set of instructions contained within an assembly language program must be translated into a machine language program before it can be executed on a computer. Translator programs that perform this function for assembly language programs are known as assemblers.

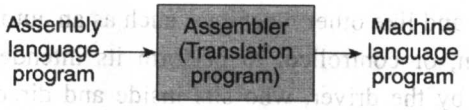


FIGURE 1.1 Assembly Programs Must Be Translated

Low- and High-Level Languages

Both machine-level and assembly languages are classified as low-level languages. This is because both of these language types use instructions that are directly tied to one type of computer.[3] Therefore, an assembly language program is limited to the specific computer type for which the program is written. These programs do, however, permit using special features of a computer that are different from other machines.

---

1 Review the supplement at the end of this chapter if you are unfamiliar with binary numbers.

2 To obtain the address values as decimal numbers, convert the binary values to decimal using the method presented in the chapter supplement.

3 In actuality, a low-level language is defined for the processor around which the computer is constructed. These include the Intel microprocessor chip for IBM-type personal computers, Motorola chips for Applebased computers, and Alpha chips for many Hewlett-Packard Compaq-based computers.

In contrast to low-level languages are languages that are classified as high-level. A highlevel language uses instructions that resemble written languages, such as English, and can be run on a variety of computer types, such as an IBM, Apple, or Hewlett-Packard computer. Pascal, Microsoft® Visual Basic®, C, C++, and Java are all high-level languages. Using Java, an instruction to add two numbers and multiply by a third number can be written as:

```
result = (first + second) * third;
```

Programs written in a computer language (high or low level) are referred to interchangeably as both source programs and source code. Once a program is written in a high-level language, it must also, like a low-level assembly program, be translated into the machine language of the computer on which it will be run. This translation can be accomplished in two ways. A unique feature of Java, as you will see, is that it uses both translation techniques, one after another.

When each statement in a high-level source program is translated individually and executed immediately upon translation, the programming language used is called an interpreted language, and the program doing the translation is called an interpreter.

When all of the statements in a high-level source program are translated as a complete unit before any one statement is executed, the programming language is called a compiled language. In this case, the program doing the translation is called a compiler. Both compiled and interpreted versions of a language can exist, although typically one predominates. For example, although compiled versions of BASIC do exist, BASIC is predominantly an interpreted language. Similarly, although interpreted versions of C++ exist, C++ is predominantly a compiled language.

A Java program's translation is a modification of the traditional process that uses *both* a compiler and interpreter. As shown in Figure 1.2, the output of the compilation step is a program in bytecode format. This bytecode is a machine code that *is not* geared to a particular computer's internal processor but rather to a computer referred to as a Java Virtual Machine (JVM). The Java Virtual Machine computer is not a physical machine but rather a software program that can read the bytecode produced by the compiler and execute it.

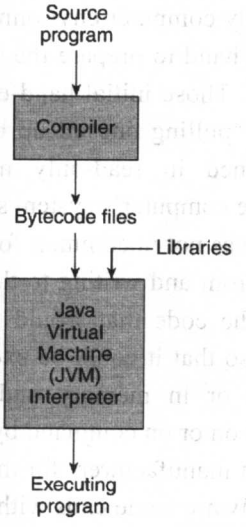


FIGURE 1.2 Translating a Java Program

The computer on which the JVM runs is referred to as the host computer. As shown in Figure 1.2, it is within the host computer's JVM that the bytecode is finally translated into a machine

language code appropriate to the host computer. Specifically, the JVM is an interpreter that translates each bytecode instruction, as it is encountered, into a computer-specific machine code that is immediately executed by the computer.

It is this two-phase translation process that provides Java with its cross-platform capability that permits each computer, regardless of its internal processor type, to execute the same Java program. It does this by placing the machine-specific details of the final translation step within the host computer's JVM rather than on the computer under which the source code was compiled.

Procedure and Object Orientations

High-level languages are further classified as either procedure oriented or object oriented. In a procedure-oriented language, the available instructions are used to create selfcontained units referred to as procedures. The purpose of a procedure is to accept data as input and transform the data in some manner to produce a specific result as an output. Until the mid-1990s, the majority of high-level languages were procedure oriented.

Within the past few years, a second orientation referred to as object oriented has taken center stage. One of the motivations for object-oriented languages was the development of graphical screens and support for graphical user interfaces (GUIs) capable of displaying multiple windows. In such an environment, each window on the screen can conveniently be considered an object with associated characteristics, such as color, position, and size. Using an object approach, a program must first define the objects it will be manipulating, which includes describing both the general characteristics of the objects themselves and specific units to manipulate them, such as changing size and position and transferring data between objects. Java is classified as an object-oriented language.

Application and System Software

Two logical categories of computer programs are application software and system software. Application software consists of programs written to perform particular tasks required by the users. Most of the examples in this book would be considered application software.

System software is the collection of programs that must be readily available to any computer system for it to operate at all. In the early computer environments of the 1950s and 1960s, a user had to initially load the system software by hand to prepare the computer to do anything. This was done with rows of switches on a front panel. Those initial hand-entered commands were said to boot the computer, an expression derived from "pulling oneself up by the bootstraps." Today, the so-called bootstrap loader is internally contained in read-only memory (ROM) and is a permanent, automatically executed component of the computer's system software.

Additionally, before the 1960s, it was not uncommon for the user to also have to load a separate set of programs necessary for reading from and writing to the input and output devices. Similarly, it was the user's responsibility to find the code that would translate an application program to the computer's internal machine language so that it could be executed. Typically, most of these utilities are now kept on either a hard disk or in memory and are booted into the computer either automatically when the system is turned on or on command by the user.

Various terms are used by different manufacturers for the collection of system utilities called the operating system. Often, the system software name ends with OS or DOS (for disk operating system). Additional tasks handled by modern operating systems include memory, input and output, and secondary storage management. Many systems handle very large programs, as well as multiple users concurrently, by dividing programs into segments or pages that are moved between the disk and memory as needed. Such operating systems create a virtual memory, which appears to be as large as

necessary to handle any job, and a *multiuser* environment is produced that gives each user the impression that the computer and peripherals are his or hers alone. Additionally, many operating systems, including most windowed environments, permit each user to run multiple programs. Such operating systems are referred to as multitasking systems.

TABLE 1.1 Common Operating System Commands

Task	DOS Command	UNIX Command
Displays the directory of all files available.	DIR	ls
Deletes a specified file or group of files.	ERASE or DELETE	rm
Lists the contents of a program to the monitor.	TYPE	cat
Prints the contents of a program to the printer.	PRINT	lp or lpr
Copies a file.	COPY	cp
Renames a file.	RENAME	mv
Forces a permanent halt to a running program.	CTRL + Break or CTRL + Z or CTRL + C	Del Key, CTRL+Break, or\
Forces a temporary suspension of the current operation.	CTRL + S or Pause	CTRL + S
Resumes the current operation (recover from a CTRL + S).	CTRL + S	CTRL + S

Most system operations are transparent to the user; that is, they take place internally without user intervention. However, some operating system commands are provided intentionally for you to interact directly with the system. The most common of these commands are those that allow the handling of data files on disk. Some of these are listed in Table 1.1.

Programming Languages

At a very basic level, the purpose of almost all programs is to process data to produce one or more specific results (Figure 1.3). In a procedure-oriented language, a program is constructed as a series of one or more sets of instructions, called procedures, that are individually concerned with producing a result from a set of inputs. Effectively, these languages concentrate on the processing shown in Figure 1.3, with each procedure moving the input data one step closer to the final desired output.

This implies that all computer programming languages that support a procedure orientation must provide essentially the same capabilities for performing these operations. These capabilities are provided either as specific instruction types or prepackaged groups of instructions that can be called to do specific tasks. Prepackaged groups of instructions in procedure-oriented languages are typically supplied in what are referred to as function libraries. Table 1.2 lists the fundamental set of instructions pro- vided by Fortran, COBOL, Pascal, C, and C++ for performing input, processing, and output tasks.

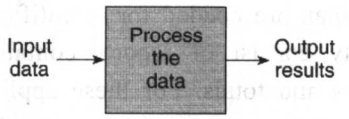


FIGURE 1. 3 Basic Program Operations