PEARSON Prentice Hall

大学计算机教育国外著名教材系列 影印版

Introduction to Programming in Java
An Interdisciplinary Approach

Java程序设计
一种跨学科的方法

Robert Sedgewick
Kevin Wayne 著

PEARSON Education

清华大学出版社

大学计算机教育国外著名教材系列（影印版）

Introduction to Programming in Java

An Interdisciplinary Approach

Java 程序设计

一种跨学科的方法

Robert Sedgewick

Kevin Wayne

Princeton University

清华大学出版社
北京

图书在版编目(CIP)数据

Java 程序设计 = Introduction to Programming in Java: An Interdisciplinary Approach: 英文 / (美) 赛德维克 (Sedgewick, R.), (美) 韦恩 (Wayne, K.) 著. —影印本. —北京: 清华大学出版社, 2009.5

(大学计算机教育国外著名教材系列(影印版))

ISBN 978-7-302-19800-0

Ⅰ. J…　Ⅱ. ①赛…②韦…　Ⅲ. Java 语言－程序设计－高等学校－教材－英文　Ⅳ. TP312

中国版本图书馆 CIP 数据核字 (2009) 第 045366 号

责任印制: 王秀菊

出版发行: 清华大学出版社　　地　址: 北京清华大学学研大厦 A 座
http://www.tup.com.cn　　邮　编: 100084
社 总 机: 010-62770175　　邮　购: 010-62786544
投稿与读者服务: 010-62776969, c-service@tup.tsinghua.edu.cn
质 量 反 馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 刷 者: 清华大学印刷厂
装 订 者: 三河市溧源装订厂
发 行 者: 全国新华书店
开　本: 185×230　　印张: 44.5
版　次: 2009 年 5 月第 1 版　　印　次: 2009 年 5 月第 1 次印刷
印　数: 1~3000
定　价: 69.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话: 010-62770177 转 3103　　产品编号: 025056-01

出 版 说 明

进入 21 世纪，世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的竞争。谁拥有大量高素质的人才，谁就能在竞争中取得优势。高等教育，作为培养高素质人才的事业，必然受到高度重视。目前我国高等教育的教材更新较慢，为了加快教材的更新频率，教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从 1996 年开始，与国外著名出版公司合作，影印出版了“大学计算机教育丛书（影印版）”等一系列引进图书，受到国内读者的欢迎和支持。跨入 21 世纪，我们本着为我国高等教育教材建设服务的初衷，在已有的基础上，进一步扩大选题内容，改变图书开本尺寸，一如既往地请有关专家挑选适用于我国高校本科及研究生计算机教育的国外经典教材或著名教材，组成本套“大学计算机教育国外著名教材系列（影印版）”，以飨读者。深切期盼读者及时将使用本系列教材的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材，以利我们把“大学计算机教育国外著名教材系列（影印版）”做得更好，更适合高校师生的需要。

清华大学出版社

Preface

THE BASIS FOR EDUCATION IN THE last millennium was "reading, writing, and arithmetic;" now it is reading, writing, and *computing*. Learning to program is an essential part of the education of every student in the sciences and engineering. Beyond direct applications, it is the first step in understanding the nature of computer science's undeniable impact on the modern world. This book aims to teach programming to those who need or want to learn it, in a scientific context.

Our primary goal is to *empower* students by supplying the experience and basic tools necessary to use computation effectively. Our approach is to teach students that writing a program is a natural, satisfying, and creative experience (not an onerous task reserved for experts). We progressively introduce essential concepts, embrace classic applications from applied mathematics and the sciences to illustrate the concepts, and provide opportunities for students to write programs to solve engaging problems.

We use the Java programming language for all of the programs in this book—we refer to Java after programming in the title to emphasize the idea that the book is about *fundamental concepts in programming*, not Java per se. This book teaches basic skills for computational problem-solving that are applicable in many modern computing environments, and is a self-contained treatment intended for people with no previous experience in programming.

This book is an *interdisciplinary* approach to the traditional CS1 curriculum, where we highlight the role of computing in other disciplines, from materials science to genomics to astrophysics to network systems. This approach emphasizes for students the essential idea that mathematics, science, engineering, and computing are intertwined in the modern world. While it is a CS1 textbook designed for any first-year college student interested in mathematics, science, or engineering (including computer science), the book also can be used for self-study or as a supplement in a course that integrates programming with another field.

Coverage The book is organized around four stages of learning to program: basic elements, functions, object-oriented programming, and algorithms (with data structures). We provide the basic information readers need to build confidence in writing programs at each level before moving to the next level. An essential feature of our approach is to use example programs that solve intriguing problems, supported with exercises ranging from self-study drills to challenging problems that call for creative solutions.

Basic elements include variables, assignment statements, built-in types of data, flow of control (conditionals and loops), arrays, and input/output, including graphics and sound.

Functions and modules are the student's first exposure to modular programming. We build upon familiarity with mathematical functions to introduce Java static methods, and then consider the implications of programming with functions, including libraries of functions and recursion. We stress the fundamental idea of dividing a program into components that can be independently debugged, maintained, and reused.

Object-oriented programming is our introduction to data abstraction. We emphasize the concepts of a data type (a set of values and a set of operations on them) and an object (an entity that holds a data-type value) and their implementation using Java's class mechanism. We teach students how to *use*, *create*, and *design* data types. Modularity, encapsulation, and other modern programming paradigms are the central concepts of this stage.

Algorithms and data structures combine these modern programming paradigms with classic methods of organizing and processing data that remain effective for modern applications. We provide an introduction to classical algorithms for sorting and searching as well as fundamental data structures (including stacks, queues, and symbol tables) and their application, emphasizing the use of the scientific method to understand performance characteristics of implementations.

Applications in science and engineering are a key feature of the text. We motivate each programming concept that we address by examining its impact on specific applications. We draw examples from applied mathematics, the physical and biological sciences, and computer science itself, and include simulation of physical systems, numerical methods, data visualization, sound synthesis, image processing, financial simulation, and information technology. Specific examples include a treatment in the first chapter of Markov chains for web page ranks and case studies that address the percolation problem, *N*-body simulation, and the small-world phenomenon. These applications are an integral part of the text. They engage students in the material, illustrate the importance of the programming concepts, and provide persuasive evidence of the critical role played by computation in modern science and engineering.

Our primary goal is to teach the specific mechanisms and skills that are needed to develop effective solutions to any programming problem. We work with complete Java programs and encourage readers to use them. We focus on programming by individuals, not library programming or programming in the large (which we treat briefly in an appendix).

Use in the Curriculum This book is intended for a first-year college course aimed at teaching novices to program in the context of scientific applications. Taught from this book, prospective majors in any area of science and engineering will learn to program in a familiar context. Any student completing a course based on this book will be well-prepared to apply their skills in later courses in science and engineering and to recognize when further education in computer science might be beneficial.

Prospective computer science majors, in particular, can benefit from learning to program in the context of scientific applications. A computer scientist needs the same basic background in the scientific method and the same exposure to the role of computation in science as does a biologist, an engineer, or a physicist.

Indeed, our interdisciplinary approach enables colleges and universities to teach prospective computer science majors and prospective majors in other fields of science and engineering in the *same* course. We cover the material prescribed by CS1, but our focus on applications brings life to the concepts and motivates students to learn them. Our interdisciplinary approach exposes students to problems in many different disciplines, helping them to more wisely choose a major.

Whatever the specific mechanism, the use of this book is best positioned early in the curriculum. First, this positioning allows us to leverage familiar material in high school mathematics and science. Second, students who learn to program early in their college curriculum will then be able to use computers more effectively when moving on to courses in their specialty. Like reading and writing, programming is certain to be an essential skill for any scientist or engineer. Students who have grasped the concepts in this book will continually develop that skill through a lifetime, reaping the benefits of exploiting computation to solve or to better understand the problems and projects that arise in their chosen field.

Prerequisites This book is meant to be suitable for typical science and engineering students in their first year of college. That is, we do not expect preparation beyond what is typically required for other entry-level science and mathematics courses.

Mathematical maturity is important. While we do not dwell on mathematical material, we do refer to the mathematics curriculum that students have taken in high school, including algebra, geometry, and trigonometry. Most students in our target audience (those intending to major in the sciences and engineering) automatically meet these requirements. Indeed, we take advantage of their familiarity with the basic curriculum to introduce basic programming concepts.

Scientific curiosity is also an essential ingredient. Science and engineering students bring with them a sense of fascination in the ability of scientific inquiry to help explain what goes on in nature. We leverage this predilection with examples of simple programs that speak volumes about the natural world. We do not assume any specific knowledge beyond that provided by typical high school courses in mathematics, physics, biology, or chemistry.

Programming experience is not necessary, but also is not harmful. Teaching programming is our primary goal, so we assume no prior programming experience. But writing a program to solve a new problem is a challenging intellectual task, so students who have written numerous programs in high school can benefit from taking an introductory programming course based on this book (just as students who have written numerous essays in high school can benefit from an introductory writing course in college). The book can support teaching students with varying backgrounds because the applications appeal to both novices and experts alike.

Experience using a computer is also not necessary, but also is not at all a problem. Every college student nowadays uses a computer regularly, to communicate with friends and relatives, listen to music, process photos, and many other activities. The realization that they can harness the power of their own computer in interesting and important ways is an exciting and lasting lesson for most students.

In summary, virtually all students in science and engineering are prepared to take a course based on this book as a part of their first-semester curriculum.

Goals What can *instructors* of upper-level courses in science and engineering expect of students who have completed a course based on this book?

We cover the CS1 curriculum, but anyone who has taught an introductory programming course knows that expectations of instructors in later courses are typically high: each instructor expects all students to be familiar with the computing environment and approach that he or she wants to use. A physics professor might expect some students to design a program over the weekend to run a simulation; an engineering professor might expect other students to be using a particular package to numerically solve differential equations; or a computer science professor might expect knowledge of the details of a particular programming environment. Is it realistic to meet such diverse expectations? Should there be a different introductory course for each set of students? Colleges and universities have been wrestling with such questions since computers came into widespread use in the latter part of the 20th century. Our answer to them is found in this common introductory treatment of programming, which is analogous to commonly accepted introductory courses in mathematics, physics, biology, and chemistry. *An Introduction to Programming* strives to provide the basic preparation needed by all students in science and engineering, while sending the clear message that there is much more to understand about computer science than programming. Instructors teaching students who have studied from this book can expect that they have the knowledge and experience necessary to enable them to adapt to new computational environments and to effectively exploit computers in diverse applications.

What can *students* who have completed a course based on this book expect to accomplish in later courses?

Our message is that programming is not difficult to learn and that harnessing the power of the computer is rewarding. Students who master the material in this book are prepared to address computational challenges wherever they might appear later in their careers. They learn that modern programming environments, such as the one provided by Java, help open the door to any computational problem they might encounter later, and they gain the confidence to learn, evaluate, and use other computational tools. Students interested in computer science will be well-prepared to pursue that interest; students in science and engineering will be ready to integrate computation into their studies.

Booksite An extensive amount of information that supplements this text may be found on the web at

```
http://www.cs.princeton.edu/IntroProgramming
```

For economy, we refer to this site as the *booksite* throughout. It contains material for instructors, students, and casual readers of the book. We briefly describe this material here, though, as all web users know, it is best surveyed by browsing. With a few exceptions to support testing, the material is all publicly available.

One of the most important implications of the booksite is that it empowers instructors and students to use their own computers to teach and learn the material. Anyone with a computer and a browser can begin learning to program by following a few instructions on the booksite. The process is no more difficult than downloading a media player or a song. As with any website, our booksite is continually evolving. It is an essential resource for everyone who owns this book. In particular, the supplemental materials are critical to our goal of making computer science an integral component of the education of all scientists and engineers.

For *instructors*, the booksite contains information about teaching. This information is primarily organized around a teaching style that we have developed over the past decade, where we offer two lectures per week to a large audience, supplemented by two class sessions per week where students meet in small groups with instructors or teaching assistants. The booksite has presentation slides for the lectures, which set the tone.

For *teaching assistants*, the booksite contains detailed problem sets and programming projects, which are based on exercises from the book but contain much more detail. Each programming assignment is intended to teach a relevant concept in the context of an interesting application while presenting an inviting and engaging challenge to each student. The progression of assignments embodies our approach to teaching programming. The booksite fully specifies all the assignments and provides detailed, structured information to help students complete them in the allotted time, including descriptions of suggested approaches and outlines for what should be taught in class sessions.

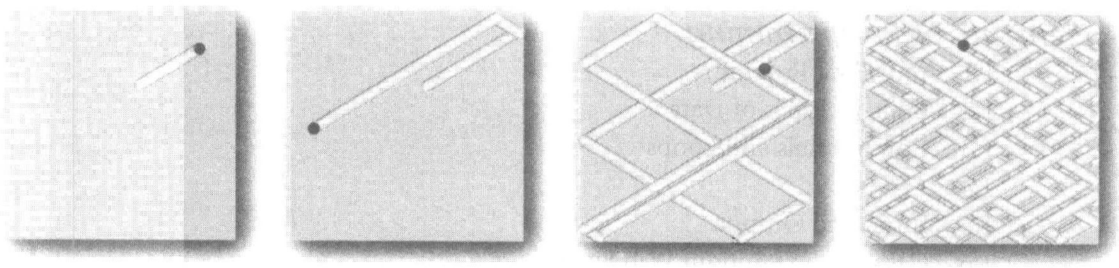
For *students*, the booksite contains quick access to much of the material in the book, including source code, plus extra material to encourage self-learning. Solutions are provided for many of the book's exercises, including complete program code and test data. There is a wealth of information associated with programming assignments, including suggested approaches, checklists, FAQs, and test data.

For *casual readers* (including instructors, teaching assistants, and students!), the booksite is a resource for accessing all manner of extra information associated with the book's content. All of the booksite content provides web links and other routes to pursue more information about the topic under consideration. There is far more information accessible than any individual could fully digest, but our goal is to provide enough to whet any reader's appetite for more information about the book's content.

Acknowledgements This project has been under development since 1992, so far too many people have contributed to its success for us to acknowledge them all here. Special thanks are due to Anne Rogers for helping to start the ball rolling; to Dave Hanson, Andrew Appel, and Chris van Wyk, for their patience in explaining data abstraction; and to Lisa Worthington, for being the first to truly relish the challenge of teaching this material to first-year students. We also gratefully acknowledge the efforts of `/dev/126` (the summer students who have contributed so much of the content); the faculty, graduate students, and teaching staff who have dedicated themselves to teaching this material over the past 15 years here at Princeton; and the thousands of undergraduates who have dedicated themselves to learning it.

Contents

Chapter One



Elements of Programming

OUR GOAL IN THIS CHAPTER IS to convince you that writing a program is easier than writing a piece of text, such as a paragraph or essay. Writing prose is difficult: we spend many years in school to learn how to do it. By contrast, just a few building blocks suffice to enable us to write programs that can help solve all sorts of fascinating, but otherwise unapproachable, problems. In this chapter, we take you through these building blocks, get you started on programming in Java, and study a variety of interesting programs. You will be able to express yourself (by writing programs) within just a few weeks. Like the ability to write prose, the ability to program is a lifetime skill that you can continually refine well into the future.

In this book, you will learn the Java programming language. This task will be much easier for you than, for example, learning a foreign language. Indeed, programming languages are characterized by no more than a few dozen vocabulary words and rules of grammar. Much of the material that we cover in this book could be expressed in the C or C++ languages, or any of several other modern programming languages. But we describe everything specifically in Java so that you can get started creating and running programs right away. On the one hand, we will focus on learning to program, as opposed to learning details about Java. On the other hand, part of the challenge of programming is knowing which details are relevant in a given situation. Java is widely used, so learning to program in this language will enable you to write programs on many computers (your own, for example). Also, learning to program in Java will make it easy for you learn other languages, including lower-level languages such as C and specialized languages such as MATLAB.

1.1 Your First Program

In this section, our plan is to lead you into the world of Java programming by taking you through the basic steps required to get a simple program running. The Java system is a collection of applications, not unlike many of the other applications that you are accustomed to using (such as your word processor, email program, and internet browser). As with any application, you need to be sure that Java is properly installed on your computer. It comes preloaded on many computers, or you can download it easily. You also need a text editor and a terminal application. Your first task is to find the instructions for installing such a Java programming environment on *your* computer by visiting

Programs in this section

`http://www.cs.princeton.edu/IntroProgramming`

We refer to this site as the *booksite*. It contains an extensive amount of supplementary information about the material in this book for your reference and use. You will find it useful to have your browser open to this site while programming.

Programming in Java To introduce you to developing Java programs, we break the process down into three steps. To program in Java, you need to:

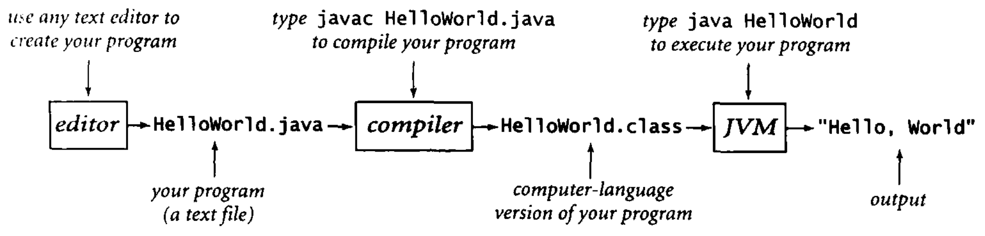
- *Create* a program by typing it into a file named, say, `MyCode.java`.
- *Compile* it by typing `javac MyCode.java` in a terminal window.
- *Run* (or *execute*) it by typing `java MyCode` in the terminal window.

In the first step, you start with a blank screen and end with a sequence of typed characters on the screen, just as when you write an email message or a paper. Programmers use the term *code* to refer to program text and the term *coding* to refer to the act of creating and editing the code. In the second step, you use a system application that *compiles* your program (translates it into a form more suitable for the computer) and puts the result in a file named `MyCode.class`. In the third step, you transfer control of the computer from the system to your program (which returns control back to the system when finished). Many systems have several different ways to create, compile, and execute programs. We choose the sequence described here because it is the simplest to describe and use for simple programs.

Creating a program. A Java program is nothing more than a sequence of characters, like a paragraph or a poem, stored in a file with a `.java` extension. To create one, therefore, you need only define that sequence of characters, in the same way as you do for email or any other computer application. You can use any *text editor* for this task, or you can use one of the more sophisticated program development environments described on the booksite. Such environments are overkill for the sorts of programs we consider in this book, but they are not difficult to use, have many useful features, and are widely used by professionals.

Compiling a program. At first, it might seem that Java is designed to be best understood by the computer. To the contrary, the language is designed to be best understood by the programmer (that's you). The computer's language is far more primitive than Java. A *compiler* is an application that translates a program from the Java language to a language more suitable for executing on the computer. The compiler takes a file with a `.java` extension as input (your program) and produces a file with the same name but with a `.class` extension (the computer-language version). To use your Java compiler, type in a terminal window the `javac` command followed by the file name of the program you want to compile.

Executing a program. Once you compile the program, you can run it. This is the exciting part, where your program takes control of your computer (within the constraints of what the Java system allows). It is perhaps more accurate to say that your computer follows your instructions. It is even more accurate to say that a part of the Java system known as the *Java Virtual Machine* (the *JVM*, for short) directs your computer to follow your instructions. To use the JVM to execute your program, type the `java` command followed by the program name in a terminal window.


use any text editor to create your program
type javac HelloWorld.java to compile your program
type java HelloWorld to execute your program
editor
HelloWorld.java
compiler
HelloWorld.class
JVM
"Hello, World"
your program (a text file)
computer-language version of your program
output

Developing a Java program

Program 1.1.1 Hello, World

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

This code is a Java program that accomplishes a simple task. It is traditionally a beginner's first program. The box below shows what happens when you compile and execute the program. The terminal application gives a command prompt (% in this book) and executes the commands that you type (`javac` and then `java` in the example below). The result in this case is that the program prints a message in the terminal window (the third line).

```
% javac HelloWorld.java
% java HelloWorld
Hello, World
```

Program 1.1.1 is an example of a complete Java program. Its name is `HelloWorld`, which means that its code resides in a file named `HelloWorld.java` (by convention in Java). The program's sole action is to print a message back to the terminal window. For continuity, we will use some standard Java terms to describe the program, but we will not define them until later in the book: Program 1.1.1 consists of a single *class* named `HelloWorld` that has a single *method* named `main()`. This method uses two other methods named `System.out.print()` and `System.out.println()` to do the job. (When referring to a method in the text, we use `()` after the name to distinguish it from other kinds of names.) Until Section 2.1, where we learn about classes that define multiple methods, all of our classes will have this same structure. For the time being, you can think of "class" as meaning "program."

The first line of a method specifies its name and other information; the rest is a sequence of *statements* enclosed in braces and each followed by a semicolon. For the time being, you can think of "programming" as meaning "specifying a class