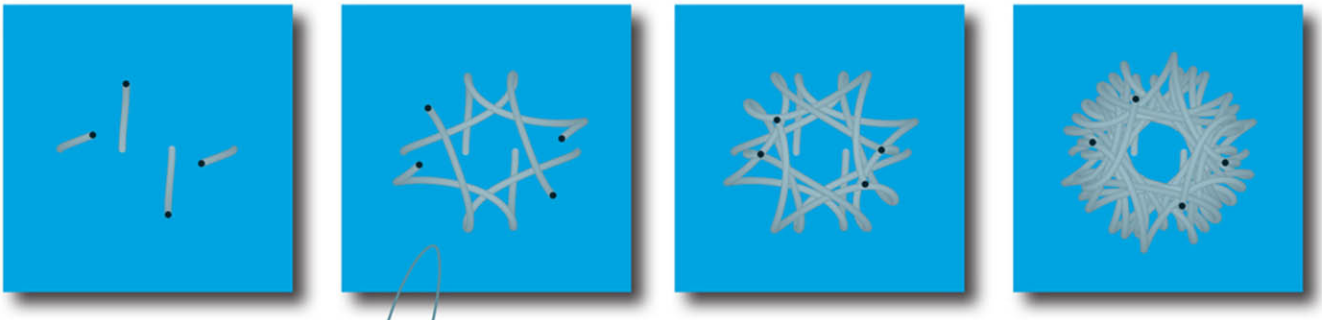
INTRODUCTION TO Programming in Java

SECOND EDITION



An Interdisciplinary Approach

Robert Sedgewick • Kevin Wayne

Introduction to Programming in Java

Second Edition

This page intentionally left blank

Introduction to Programming in Java

An Interdisciplinary Approach

Second Edition

Robert Sedgewick
Kevin Wayne

Princeton University

Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Visit us on the Web: `informit.com/aw`

Library of Congress Control Number: 2017934241

ISBN-13: 978-0-672-33784-0
ISBN-10: 0-672-33784-3

1 17

To Adam, Andrew, Brett, Robbie,

Henry, Iona, Peter, Rose,

and especially Linda

To Jackie, Alex, and Michael

Contents

Programs viii

Preface xi

1—Elements of Programming 1

1.1 Your First Program *2*
1.2 Built-in Types of Data *14*
1.3 Conditionals and Loops *50*
1.4 Arrays *90*
1.5 Input and Output *126*
1.6 Case Study: Random Web Surfer *170*

2—Functions and Modules 191

2.1 Defining Functions *192*
2.2 Libraries and Clients *226*
2.3 Recursion *262*
2.4 Case Study: Percolation *300*

3—Object-Oriented Programming 329

3.1 Using Data Types *330*
3.2 Creating Data Types *382*
3.3 Designing Data Types *428*
3.4 Case Study: N-Body Simulation *478*

4—Algorithms and Data Structures 493

4.1 Performance *494*
4.2 Sorting and Searching *532*
4.3 Stacks and Queues *566*
4.4 Symbol Tables *624*
4.5 Case Study: Small-World Phenomenon *670*

Context 715

Glossary 721

Index 729

APIs 751

Programs

Elements of Programming

Your First Program

1.1.1 Hello, World 4
1.1.2 Using a command-line argument 7

Built-in Types of Data

1.2.1 String concatenation 20
1.2.2 Integer multiplication and division 23
1.2.3 Quadratic formula 25
1.2.4 Leap year 28
1.2.5 Casting to get a random integer . . 34

Conditionals and Loops

1.3.1 Flipping a fair coin 53
1.3.2 Your first while loop. 55
1.3.3 Computing powers of 2 57
1.3.4 Your first nested loops 63
1.3.5 Harmonic numbers 65
1.3.6 Newton's method 66
1.3.7 Converting to binary 68
1.3.8 Gambler's ruin simulation 71
1.3.9 Factoring integers. 73

Arrays

1.4.1 Sampling without replacement . . 98
1.4.2 Coupon collector simulation . . 102
1.4.3 Sieve of Eratosthenes 104
1.4.4 Self-avoiding random walks . . 113

Input and Output

1.5.1 Generating a random sequence 128
1.5.2 Interactive user input 136
1.5.3 Averaging a stream of numbers 138
1.5.4 A simple filter 140
1.5.5 Standard input-to-drawing filter 147
1.5.6 Bouncing ball 153
1.5.7 Digital signal processing 158

Case Study: Random Web Surfer

1.6.1 Computing the transition matrix 173
1.6.2 Simulating a random surfer . . 175
1.6.3 Mixing a Markov chain 182

Functions and Modules

Defining Functions

2.1.1 Harmonic numbers (revisited) 194
2.1.2 Gaussian functions 203
2.1.3 Coupon collector (revisited) . . 206
2.1.4 Play that tune (revisited) 213

Libraries and Clients

2.2.1 Random number library 234
2.2.2 Array I/O library 238
2.2.3 Iterated function systems. . . . 241
2.2.4 Data analysis library. 245
2.2.5 Plotting data values in an array 247
2.2.6 Bernoulli trials 250

Recursion

2.3.1 Euclid's algorithm. 267
2.3.2 Towers of Hanoi 270
2.3.3 Gray code 275
2.3.4 Recursive graphics 277
2.3.5 Brownian bridge 279
2.3.6 Longest common subsequence 287

Case Study: Percolation

2.4.1 Percolation scaffolding. 304
2.4.2 Vertical percolation detection . . 306
2.4.3 Visualization client 309
2.4.4 Percolation probability estimate 311
2.4.5 Percolation detection 313
2.4.6 Adaptive plot client 316

Object-Oriented Programming

Using Data Types

3.1.1 Identifying a potential gene . . 337
3.1.2 Albers squares 342
3.1.3 Luminance library 345
3.1.4 Converting color to grayscale . . 348
3.1.5 Image scaling 350
3.1.6 Fade effect 352
3.1.7 Concatenating files 356
3.1.8 Screen scraping for stock quotes 359
3.1.9 Splitting a file 360

Creating Data Types

3.2.1 Charged particle 387
3.2.2 Stopwatch 391
3.2.3 Histogram 393
3.2.4 Turtle graphics 396
3.2.5 Spira mirabilis 399
3.2.6 Complex number 405
3.2.7 Mandelbrot set 409
3.2.8 Stock account 413

Designing Data Types

3.3.1 Complex number (alternate) . . 434
3.3.2 Counter 437
3.3.3 Spatial vectors 444
3.3.4 Document sketch 461
3.3.5 Similarity detection 463

Case Study: N-Body Simulation

3.4.1 Gravitational body 482
3.4.2 N-body simulation 485

Algorithms and Data Structures

Performance

4.1.1 3-sum problem 497
4.1.2 Validating a doubling hypothesis 499

Sorting and Searching

4.2.1 Binary search (20 questions) . . 534
4.2.2 Bisection search 537
4.2.3 Binary search (sorted array) . . 539
4.2.4 Insertion sort 547
4.2.5 Doubling test for insertion sort 549
4.2.6 Mergesort 552
4.2.7 Frequency counts 557

Stacks and Queues

4.3.1 Stack of strings (array). 570
4.3.2 Stack of strings (linked list). . . 575
4.3.3 Stack of strings (resizing array) 579
4.3.4 Generic stack 584
4.3.5 Expression evaluation 588
4.3.6 Generic FIFO queue (linked list) 594
4.3.7 M/M/1 queue simulation . . . 599
4.3.8 Load balancing simulation . . . 607

Symbol Tables

4.4.1 Dictionary lookup 631
4.4.2 Indexing. 633
4.4.3 Hash table 638
4.4.4 Binary search tree. 646
4.4.5 Dedup filter 653

Case Study: Small-World Phenomenon

4.5.1 Graph data type 677
4.5.2 Using a graph to invert an index 681
4.5.3 Shortest-paths client 685
4.5.4 Shortest-paths implementation 691
4.5.5 Small-world test 696
4.5.6 Performer–performer graph . . 698

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 composing a program is a natural, satisfying, and creative experience. 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, in that 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, 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 their ability to compose programs at each level before moving to the next level. An essential feature of our approach is the use of 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, 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 functions, 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 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 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 programming in the large.

Related texts This book is the second edition of our 2008 text that incorporates hundreds of improvements discovered during another decade of teaching the material, including, for example, a new treatment of hashing algorithms.

The four chapters in this book are identical to the first four chapters of our text *Computer Science: An Interdisciplinary Approach*. That book is a full introductory course on computer science that contains additional chapters on the theory of computing, machine-language programming, and machine architecture. We have published this book separately to meet the needs of people who are interested only in the Java programming content. We also have published a version of this book that is based on Python programming.

The chapters in this volume are suitable preparation for our book *Algorithms, Fourth Edition*, which is a thorough treatment of the most important algorithms in use today.

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

Instructors interested in a full-year course (or a fast-paced one-semester course with broader coverage) should instead consider adopting *Computer Science: An Interdisciplinary Approach.*

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 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 choose a major more wisely.

Whatever the specific mechanism, the use of this book is best positioned early in the curriculum. This positioning allows us to leverage familiar material in high school mathematics and science. Moreover, 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 suitable for typical first-year college students. In other words, 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 automatically meet these requirements. Indeed, we take advantage of familiarity with this curriculum to introduce basic programming concepts.

Scientific curiosity is also an essential ingredient. Science and engineering students bring with them a sense of fascination with the ability of scientific inquiry to help explain what occurs 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. Nevertheless, composing 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. The book can support teaching students with varying backgrounds because the applications appeal to both novices and experts alike.

Experience using a computer is not necessary, but also is not at all a problem. College students use computers regularly—to communicate with friends and relatives, to listen to music, to process photos, and as part of 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.

Goals 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. For example, a physics professor might expect students to design a program over the weekend to run a simulation; a biology professor might expect students to be able to analyze genomes; or a computer science professor might expect knowledge of the details of a particular programming environment. Is it realistic to meet such diverse expectations? Is it realistic to offer a single introductory CS course for all students, as opposed to a different introductory course for each set of students?

With this book, and decades of experience at Princeton and other institutions that have adopted earlier versions, we answer these questions with a resounding *yes*. The most important reason to do so is that this approach encourages diversity. By keeping interesting applications at the forefront, we can keep advanced students engaged, and by avoiding classifying students at the beginning, we can ensure that every student who successfully masters this material is prepared for further study.

What can *teachers* of upper-level college courses expect of students who have completed a course based on this book?

This is a common introductory treatment of programming, which is analogous to commonly accepted introductory courses in mathematics, physics, biology, economics, or chemistry. *An Introduction to Programming in Java* strives to provide the basic preparation needed by all college students, while sending the clear message that there is much more to understand about computer science than just programming. Instructors teaching students who have studied from this book can expect that they will have the knowledge and experience necessary to enable them 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 other fields will be ready to integrate computation into their studies.

Online lectures A complete set of studio-produced videos that can be used in conjunction with this text is available at

```
http://www.informit.com/title/9780134493831
```

As with traditional live lectures, the purpose is to *inform* and *inspire*, motivating students to study and learn from the text. Our experience is that student engagement with such online material is significantly better than with live lectures because of the ability to play the lectures at a chosen speed and to replay and review the lectures at any time.

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

```
http://introcs.cs.princeton.edu/java
```

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.

The booksite contains a condensed version of the text narrative for reference while online, hundreds of exercises and programming problems (some with solutions), hundreds of easily downloadable Java programs, real-world data sets, and our I/O libraries for processing text, graphics, and sound. It is the web presence associated with the book and is a living document that is accessed millions of times per year. It is an essential resource for everyone who owns this book and is critical to our goal of making computer science an integral component of the education of all college students.

One of the most important implications of the booksite is that it empowers teachers 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.

For *teachers*, the booksite contains resources for teaching that (together with the book and the studio-produced videos) are sufficiently flexible to support many of the models for teaching that are emerging as teachers embrace technology in the 21st century. For example, at Princeton, our teaching style was for many years based on offering 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. More recently, we have moved to a model where students watch lectures online and we hold *class meetings* once a week in addition to the two class sessions. Other teachers may work completely online. Still others may use a "flipped" model involving enrichment of the lectures after students watch them.

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*, 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.

Acknowledgments 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 and Donna Gabai, 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 faculty, graduate students, and teaching staff who have dedicated themselves to teaching this material over the past 25 years here at Princeton University; and the thousands of undergraduates who have dedicated themselves to learning it.

Robert Sedgewick
Kevin Wayne

February 2017

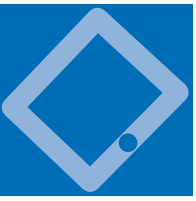
Chapter One

Elements of Programming

1.1 Your First Program 2
1.2 Built-in Types of Data 14
1.3 Conditionals and Loops 50
1.4 Arrays 90
1.5 Input and Output 126
1.6 Case Study: Random Web Surfer . . . 170

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 only a few dozen vocabulary words and rules of grammar. Much of the material that we cover in this book could be expressed in the Python or C++ languages, or any of several other modern programming languages. 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 to 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 platform* (hereafter abbreviated *Java*) 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 web 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

1.1.1 Hello, World 4
1.1.2 Using a command-line argument . . 7
Programs in this section

```
http://introcs.cs.princeton.edu/java
```

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 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:

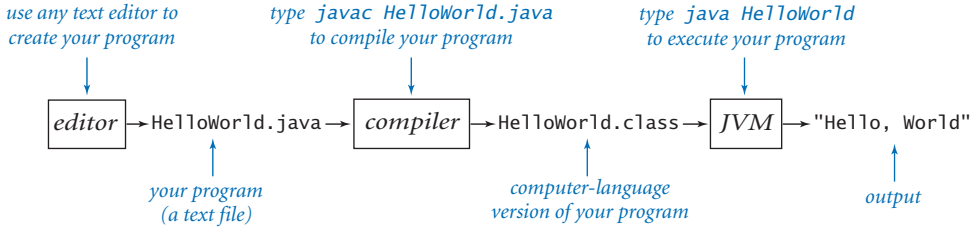
- *Create* a program by typing it into a file named, say, `MyProgram.java`.
- *Compile* it by typing `javac MyProgram.java` in a terminal window.
- *Execute* (or *run*) it by typing `java MyProgram` in the terminal window.

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 compose an email message or an essay. 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 `MyProgram.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 given here because it is the simplest to describe and use for small 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 simply 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 *integrated 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 execution 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 (running) a program. Once you compile the program, you can execute (or run) it. This is the exciting part, where your program takes control of your computer (within the constraints of what Java 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 Java known as the *Java virtual machine* (*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.



Developing a Java program

Program 1.1.1 Hello, World

```
public class HelloWorld
{
   public static void main(String[] args)
   {
      // Prints "Hello, World" in the terminal window.
      System.out.println("Hello, World");
   }
}
```

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). Our convention is to highlight in boldface the text that you type and display the results in regular face. In this case, the result is that the program prints the message `Hello, World` in the terminal window.

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

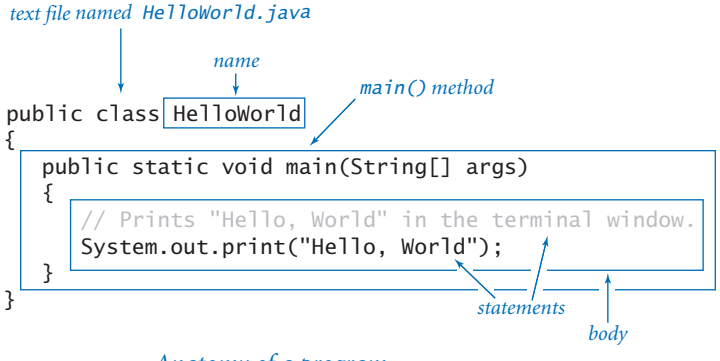
PROGRAM 1.1.1 is an example of a complete Java program. Its name is `HelloWorld`, which means that its code resides in a file named `HelloWorld.java` (by convention in Java). The program's sole action is to print a message to the terminal window. For continuity, we will use some standard Java terms to 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()`. (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, 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 curly braces, with each statement typically followed by a semicolon. For the time being, you can think of "programming" as meaning "specifying a class name and a sequence of statements for its `main()` method," with the heart of the program consisting of the sequence of statements in the `main()` method (its *body*). PROGRAM 1.1.1 contains two such statements:

- The first statement is a *comment*, which serves to document the program. In Java a single-line comment begins with two '/' characters and extends to the end of the line. In this book, we display comments in gray. Java ignores comments—they are present only for human readers of the program.
- The second statement is a *print statement*. It calls the method named `System.out.println()` to print a text message—the one specified between the matching double quotes—to the terminal window.

In the next two sections, you will learn about many different kinds of statements that you can use to make programs. For the moment, we will use only comments and print statements, like the ones in `HelloWorld`.

When you type `java` followed by a class name in your terminal window, the system calls the `main()` method that you defined in that class, and executes its statements in order, one by one. Thus, typing `java HelloWorld` causes the system to call the `main()` method in PROGRAM 1.1.1 and execute its two statements. The first statement is a comment, which Java ignores. The second statement prints the specified message to the terminal window.



Anatomy of a program

Since the 1970s, it has been a tradition that a beginning programmer's first program should print `Hello, World`. So, you should type the code in PROGRAM 1.1.1 into a file, compile it, and execute it. By doing so, you will be following in the footsteps of countless others who have learned how to program. Also, you will be checking that you have a usable editor and terminal application. At first, accomplishing the task of printing something out in a terminal window might not seem very interesting; upon reflection, however, you will see that one of the most basic functions that we need from a program is its ability to tell us what it is doing.

For the time being, all our program code will be just like PROGRAM 1.1.1, except with a different sequence of statements in `main()`. Thus, you do not need to start with a blank page to write a program. Instead, you can

- Copy `HelloWorld.java` into a new file having a new program name of your choice, followed by `.java`.
- Replace `HelloWorld` on the first line with the new program name.
- Replace the comment and print statements with a different sequence of statements.

Your program is characterized by its sequence of statements and its name. Each Java program must reside in a file whose name matches the one after the word `class` on the first line, and it also must have a `.java` extension.

Errors. It is easy to blur the distinctions among editing, compiling, and executing programs. You should keep these processes separate in your mind when you are learning to program, to better understand the effects of the errors that inevitably arise.

You can fix or avoid most errors by carefully examining the program as you create it, the same way you fix spelling and grammatical errors when you compose an email message. Some errors, known as *compile-time* errors, are identified when you compile the program, because they prevent the compiler from doing the translation. Other errors, known as *run-time* errors, do not show up until you execute the program.

In general, errors in programs, also commonly known as *bugs*, are the bane of a programmer's existence: the error messages can be confusing or misleading, and the source of the error can be very hard to find. One of the first skills that you will learn is to identify errors; you will also learn to be sufficiently careful when coding, to avoid making many of them in the first place. You can find several examples of errors in the Q&A at the end of this section.

Program 1.1.2 Using a command-line argument

```
public class UseArgument
{
   public static void main(String[] args)
   {
      System.out.print("Hi, ");
      System.out.print(args[0]);
      System.out.println(". How are you?");
   }
}
```

This program shows the way in which we can control the actions of our programs: by providing an argument on the command line. Doing so allows us to tailor the behavior of our programs.

```
% javac UseArgument.java
% java UseArgument Alice
Hi, Alice. How are you?
% java UseArgument Bob
Hi, Bob. How are you?
```

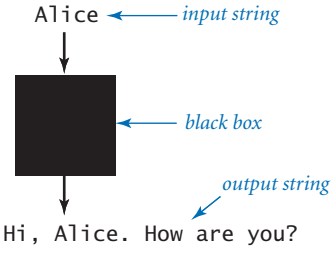
Input and output Typically, we want to provide *input* to our programs—that is, data that they can process to produce a result. The simplest way to provide input data is illustrated in `UseArgument` (PROGRAM 1.1.2). Whenever you execute the program `UseArgument`, it accepts the *command-line argument* that you type after the program name and prints it back out to the terminal window as part of the message. The result of executing this program depends on what you type after the program name. By executing the program with different command-line arguments, you produce different printed results. We will discuss in more detail the mechanism that we use to pass command-line arguments to our programs later, in SECTION 2.1. For now it is sufficient to understand that `args[0]` is the first command-line argument that you type after the program name, `args[1]` is the second, and so forth. Thus, you can use `args[0]` within your program's body to represent the first string that you type on the command line when it is executed, as in `UseArgument`.

In addition to the `System.out.println()` method, `UseArgument` calls the `System.out.print()` method. This method is just like `System.out.println()`, but prints just the specified string (and not a newline character).

Again, accomplishing the task of getting a program to print back out what we type in to it may not seem interesting at first, but upon reflection you will realize that another basic function of a program is its ability to respond to basic information from the user to control what the program does. The simple model that `UseArgument` represents will suffice to allow us to consider Java's basic programming mechanism and to address all sorts of interesting computational problems.

Stepping back, we can see that `UseArgument` does neither more nor less than implement a function that maps a string of characters (the command-line argument) into another string of characters (the message printed back to the terminal window). When using it, we might think of our Java program as a black box that converts our input string to some output string.

This model is attractive because it is not only simple but also sufficiently general to allow completion, in principle, of any computational task. For example, the Java compiler itself is nothing more than a program that takes one string of characters as input (a `.java` file) and produces another string of characters as output (the corresponding `.class` file). Later, you will be able to write programs that accomplish a variety of interesting tasks (though we stop short of programs as complicated as a compiler). For the moment, we will live with various limitations on the size and type of the input and output to our programs; in SECTION 1.5, you will see how to incorporate more sophisticated mechanisms for program input and output. In particular, you will see that we can work with arbitrarily long input and output strings and other types of data such as sound and pictures.



A bird's-eye view of a Java program

Q&A

Q. Why Java?

A. The programs that we are writing are very similar to their counterparts in several other languages, so our choice of language is not crucial. We use Java because it is widely available, embraces a full set of modern abstractions, and has a variety of automatic checks for mistakes in programs, so it is suitable for learning to program. There is no perfect language, and you certainly will be programming in other languages in the future.

Q. Do I really have to type in the programs in the book to try them out? I believe that you ran them and that they produce the indicated output.

A. Everyone should type in and run `HelloWorld`. Your understanding will be greatly magnified if you also run `UseArgument`, try it on various inputs, and modify it to test different ideas of your own. To save some typing, you can find all of the code in this book (and much more) on the booksite. This site also has information about installing and running Java on your computer, answers to selected exercises, web links, and other extra information that you may find useful while programming.

Q. What is the meaning of the words `public`, `static`, and `void`?

A. These keywords specify certain properties of `main()` that you will learn about later in the book. For the moment, we just include these keywords in the code (because they are required) but do not refer to them in the text.

Q. What is the meaning of the `//`, `/*`, and `*/` character sequences in the code?

A. They denote *comments*, which are ignored by the compiler. A comment is either text in between `/*` and `*/` or at the end of a line after `//`. Comments are indispensable because they help other programmers to understand your code and even can help you to understand your own code in retrospect. The constraints of the book format demand that we use comments sparingly in our programs; instead we describe each program thoroughly in the accompanying text and figures. The programs on the booksite are commented to a more realistic degree.