高等学校计算机专业教材精选·算法与程序设计

Foundations of Programming in Java

董东 [澳] Gavin Finnie 编著

清华大学出版社

高等学校计算机专业教材精选·算法与程序设计

Foundations of Programming in Java

董东 [澳] Gavin Finnie 编著

清华大学出版社
北京

内 容 简 介

本书以对象的概念为核心，立足于双语教学的需要，由浅入深、循序渐进地介绍 Java 面向对象程序设计的基本思想、方法和技术，力图使学生直接使用英语掌握 J2SE 的基本内容，在满足专业要求的同时，提高专业英语的阅读理解能力。

全书共分 9 章。第 1 章介绍 Java 程序设计环境并强调了注释的类别和重要作用；第 2 章介绍基本数据类型及运算、简单的控制台输入输出、流程控制语句、数组以及使用 Java 中内置的类来实例化对象完成一定的功能，例如使用大整数类实现大整数运算；第 3 章介绍类与对象的设计；第 4 章介绍继承、接口和多态；第 5 章介绍异常处理；第 6 章介绍 Collection 框架；第 7 章介绍输入输出流；第 8 章介绍图形用户界面设计；第 9 章介绍多线程和并发程序设计。每章后面均附有上机指导、术语表和习题。附录给出了 Java 修饰符、Java 文档注释、Unicode 表和常用 Eclipse 快捷键。

本书可作为计算机类专业 Java 面向对象程序设计的双语教材，也可供专业技术人员参考。

图书在版编目（CIP）数据

Java 基础教程=Foundations of Programming in Java：英文/董东，（澳）芬尼（Finnie, G.）编著. --北京：清华大学出版社，2013

高等学校计算机专业教材精选 • 算法与程序设计

ISBN 978-7-302-32551-2

Ⅰ. ①J…　Ⅱ. ①董… ②芬…　Ⅲ. ①JAVA 语言–程序设计–高等学校–教材–英文　Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2013）第 115108 号

责任编辑：张　玥　战晓雷
封面设计：傅瑞学
责任校对：时翠兰
责任印制：何　芊

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn, http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　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
课 件 下 载：http://www.tup.com.cn,010-62795954

印 刷 者：北京世知印务有限公司
装 订 者：三河市新茂装订有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：26.25　　**字　　数**：639 千字
版　　次：2013 年 8 月第 1 版　　**印　　次**：2013 年 8 月第 1 次印刷
印　　数：1～2000
定　　价：55.00 元

产品编号：050601-01

About This Book

This book is aimed at students majoring in computer science and related programs at college who wish to learn the object-oriented language Java. It is particularly useful for readers who have had programming practice with the C language. The book uses version 6.0 of J2SE. The Java Platform, Standard Edition (Java SE) is popular for developing and deploying Java applications on desktops and servers, as well as in today's demanding embedded environments. Java offers a rich user interface, performance, versatility, portability, and security that applications now require. As the language is in a continual state of development, this book will be updated as new versions appear.

The first three chapters of the book introduce fundamental programming technologies to illustrate the basic ideas and constructions in Java. This includes the object-oriented concept, Java application structure, primitive data types and reference types, literals, variables, expressions and statements, control structures, arrays, and the built-in Java classes such as String, Random Number, and BigInteger. The class and object concepts are the core of this book. It tries to explain clearly how a Java program is organized by classes from the static point view and how a Java program runs with objects and the communication of objects from the dynamic perspective.

The next chapter, inheritance, interface and polymorphism, introduces advanced technologies for the construction of complicated software.

Then the book moves to explore key issues in Java exception handling. Exceptions are objects generated by the platform or your program. The exception handling mechanism makes source code more readable since it separates the code in normal situation and abnormal situations.

The collections framework, Stream I/O and GUI components are the important APIs provided by Java SE. A Java programmer should be familiar with these APIs in order to code efficiently.

The last chapter of this book concentrates on multithreading and concurrency programming. Besides the status and life cycle of a thread, it focuses on the solutions to share resources such as volatile, synchronized methods, and atomic variables. It also discusses Executors for the thread life cycle management.

This book also lists new terminologies, new words and expressions from the English language perspective for the Chinese students at the end of each chapter as well as exercises. Additional online exercises will be on our Online Judge website.

The list of Java modifiers, the style guide and tag conventions for documentation comments, and the Unicode chart (basic Latin) are also included as appendixes.

We warmly welcome any comments and suggestions. Feel free to contact the authors by email dongdong@hebtu.edu.cn.

Dong Dong, Gavin Finnie
Feb. 2013

Preface

Java satisfies current application needs as an object-oriented language in the Web computational platform. The Java language is becoming more and more popular.

This book attempts to satisfy a number of goals:

- Emphasize the importance of the object-oriented paradigm. Students need to think in terms of objects, classes, and message passing as a critical programming paradigm.
- Emphasize the importance of problem-solving skills. Student need to learn how to abstract, design algorithms, and then translate those algorithms into working programs.
- Emphasize good style. The source code is not only read by machine, but also read by humans such as super managers, colleagues and even the programmers themselves. I have been careful to use that style in all the examples and case studies.
- Emphasize entertaining programming. By providing interesting programming questions—often involving stories in our daily life—I've tried to show students that they can have fun at the same time they're learning. I have also tried to use realistic examples to help motivate students and show clearly how programming techniques are used in the real world.
- Emphasize understanding, not memory. Students may forget the syntax structure of the while loop in Java some time after graduation. However, they should not forget the programming paradigm and solution ideas.

Source code for all programs in this book is available via the Web at Tsinghua University Press(http://www.tup.com.cn). Updates, corrections, and news about the book are also available at this site.

The prototype of this textbook is the lecture notes used on my bilingual class for computing students during the past 5 years. I am responsible for the technical issues. Dr. Gavin Finnie, a professor from Bond University at Australia, has focused on the English language details in order to improve the readability. I really appreciate his kindness, sage advice, and excellent cooperation to improve my manuscript.

I would like to thank our publisher, Tsinghua University Press, for the excellent commendation and efficient work conducted throughout the production of this book. Additionally, I would like to thank especially the computing students of the Hebei Normal University for their thoughtful comments and feedback. For support and inspiration, special thanks are due to our families and colleagues. I am also grateful for the talented writers who have posted their ideas and code on blogs and Web pages although I have not had the chance to get to know them. Finally, I want to thank my university for the chance given to me to teach the bilingual course.

Dong Dong
Feb. 2013

CONTENTS

Chapter 1 Introduction to Java Programming

OBJECTIVES

- To become familiar with the phases for a problem solving
- To understand the structure of a Java application
- To be able to follow the coding style

1.1 Abstraction

A car is a wheeled vehicle used for transporting passengers, which also carries its own engine(Fig.1.1). A car typically has seating for five people and four wheels. It has additional features, like gas tank size, sliding doors, logo type, CD changer slot count, Bluetooth, GPS navigator, or other various properties that are specific to the make/model of the car. A car can move forwards and backwards at different speeds and turn left and right. In addition, a car may have certain behaviors like open/close window or sunroof, open/close lights, ABS, and other behaviors that would be specific to a car. When another car is scanned with the reversing radar, the scanning car can take actions such as sounding an alarm.

Now consider simulating the behaviors of the cars in a computer. First, let us consider a simple scenario. For a specific car called `Donald`, let the car move ahead and then move backward.

From an object perspective, a car is an object, which is made up of several components such as motor, wheels, and radar. The object can perform behaviors such as moving forward, moving backward, turning left, turning right, and scaning. Every car object has attributes including price, manufacture date, manufacturer, country-of-assembly, provider, mileage, mileage between services, body style, width, height, length, weight, color, position, driving directions, etc.

Fig. 1.2 shows an abstract representation of a car. For the sake of simplicity, we suppose the car here can only turn left and turn right by 90 degrees.

Fig. 1.1 A car in daily life

Car
name: String height, width: double x, y: int direction: byte
moveAhead(int distance) moveBack(int distance) turnRight() turnLeft()

Fig. 1.2 An abstraction representation for a car

The abstraction is termed as a class. From an object-oriented view, a class is used to describe all objects that share the same behaviors and internal attributes. Each object is an instance of the class to which it belongs. `Car` is the name of the class in Fig. 1.2, attributes and behaviors of the class are listed under the name.

The diagram in Fig. 1.2 is known as a class diagram. A UML (Unified Modelling Language) class diagram describes the types of objects in a problem domain and the various kinds of static relationships that exist among them. The top compartment in a class diagram shows the class's name. The middle compartment lists the class's attributes. Classes have attributes that describe the characteristics of their objects. Attributes are atomic entities with no responsibilities. The class's operations are documented in the third (lowest) compartment of the class diagram's rectangle. Like the attributes, the operations of a class are displayed in a list format, with each operation on its own line. Operations are the processes that objects of a class (or the class itself) know to carry out. Attributes and operations are all optional.

Besides attributes and operations, there is another essential element in a class diagram, relationships, such as inheritance, association, realization and dependency. Inheritance models "is a" and "is like" relationships, enabling you to rather conveniently reuse data and code that already exist. When A inherits from B we say that A is the subclass of B and that B is the superclass of A. In the UML class diagram, the inheritance is indicated by a solid line with a closed, unfilled arrowhead pointing at the superclass. For example, the fact that a car is a vehicle can be depicted in class diagram as Fig. 1.3 shows.

An association is a semantic relationship between two or more classes that specifies connections among their instances. It is indicated by a solid line between the two classes. A realization relationship indicates that one class implements a behavior specified by another class (an interface). A dotted line with a closed, unfilled arrow means realization (or implementation). A dependency is a relation between two classes in which a change in one may force changes in the other although there is no explicit association between them. A dotted line with an open arrowhead that shows one object depends on the behavior of another object.

In order to initialize a `Car` object, the class `Car` should be designed first. `Donald` is an instance of the class. Once instantiated, `Donald` can perform some actions. The phases of a problem solving are expressed diagrammatically in Fig. 1.4.

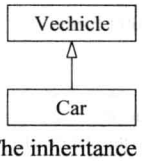

Fig. 1.3 The inheritance relationship

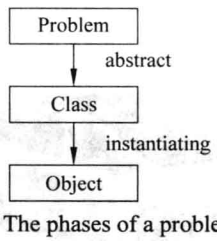

Fig. 1.4 The phases of a problem solving

A class is an abstract representation of an entity in a problem. An object is an instance of a class. The term "instantiating" is the same thing as "creating". When an object is created, an

"instance" of a class is created, therefore "instantiating" a class.

Abstraction is the process or result of generalization by reducing the information content of a concept or an observable phenomenon, typically in order to retain only information that is relevant for a particular purpose. The car abstraction above retains only the information on general car attributes and behavior. We use abstraction to understand and solve problems and communicate our solutions with the computer in Java language.

1.2 Development Environment and Running Environment

There are two types of programs in Java: applications and applets. Applications are the programs that can be run directly by the Java interpreter while applets are programs designed to be run by a Java-enabled browser. The Java Development Kit (JDK) consists of the primary programming tools such as a loader, a compiler, an interpreter, and a debugger to develop Java programs. A Java program could be compiled and run after the JDK is installed successfully. The JDK also comes with a complete Java Runtime Environment (JRE). The Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine(JVM), and other components to run applets and applications written in the Java programming language. The Java virtual machine is the code that ultimately runs Java programs by interpreting the intermediate byte-code format of the Java program. The Java platform consists of a Java virtual machine and all of the class libraries provided in the production environment. Fig. 1.5 depicts the Java platform.

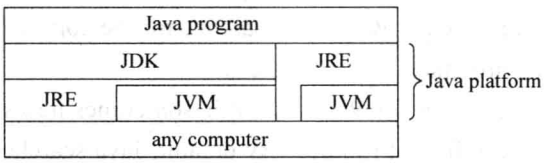

Fig. 1.5 The Java platform

The newest version of JDK can be downloaded from the official web site. Double-click the .exe file downloaded (for Windows) to run the installation. The installation destination folder is referred to as "Java home". After installation, the `bin`, `jre`, `lib` and other folders will be available in your Java home folder as Fig. 1.6 shows. In this case, the destination folder, `c:\Program Files\Java\jdk1.6.0_20`, is Java home. All of the programming tools, such as `javac` and `java`, reside in the `bin` folder. The `jre` folder contains the Java virtual machine (`bin` folder) and runtime library (`lib` folder). `Java` in the `bin` folder is the interpreter to run programs. Note that `jdk/jre/bin` contains two jvm.dll (virtual machines) located in `client` sub-folder and `server` sub-folder respectively. The client virtual machine is normally used.

Next, you turn to set the Windows environment variable `Path`. The environment variable `Path` enables you to issue and execute the compile command `javac` or others anywhere in the command prompt window. You may set `Path` by right clicking on **My Computer** icon on the

Fig. 1.6 Java home folder

Windows desktop and selecting **Properties**. Under the **Advanced** tab, there is a button to set the **Environment variables**. Click on this button and alter the **Path** variable so that it also contains the path to the Java executable. For example, if Java is installed in `C:\Program Files\Java\jdk1.6.0_20` and the environment variable `Path` is currently set to `"C:\WINDOWS\ SYSTEM32"`, then you should change `Path` to `"C:\Program Files\Java\jdk1.6.0_ 20\bin; C:\WINDOWS\SYSTEM32"`. After that, you will be able to run `javac` to compile code in a new command prompt window.

Note that you must use a semicolon (;) to separate the folders in path. The symbol dot (.) stands for the current working folder. Windows searches for the command that you issued by the order of folders listed in environment variable `Path`.

In addition to setting up `Path` for Windows, it is sometimes necessary to set `classpath` for Java to find compiled class files at runtime. By default, Java searches for them in the current working folder and the `lib` folder. Therefore, if you are sure that all the classes needed are in the two default folders, it is unnecessary to set `classpath`. The `classpath` can also contain other folders that may contain compiled class files used by the Java application. Note that if you are using classes that are contained inside a `.jar` file, you will need to specify the full path and name of this file in the classpath, rather than just the name of the folder it is contained.

1.3 Programming in the Command Prompt Window

The basic programming tools include an editor, a compiler, and an interpreter. Initially, you use an editor to type, revise your Java programs, and save them in disk files with `.java` extension. This original code is called **source code**. After editing and saving your source code, you attempt to use command `javac` to translate them into Java **bytecode,** which is a representation of the program in a low-level form similar to machine language code. The

compiler takes a file with a `.java` extension as input and generates a file with the same name but with a `.class` extension. The difference between Java bytecode and true machine language code is that Java bytecode is independent on any particular processor type. This approach makes Java architecture neutral. Therefore, your Java program is easily portable from one machine to another. However, the Java bytecode cannot be executed directly. You have to call the Java interpreter for each processor type to read Java bytecode and execute it on a specific machine. The process is pictured in Fig. 1.7.

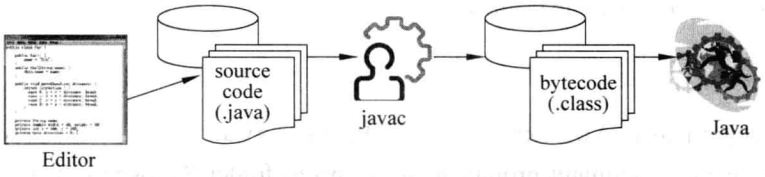

Fig. 1.7 The process for Java programming

The compiler, javac, converts source code into Java bytecode while Java, the loader, is also an interpreter and can interpret the class files generated by the javac compiler. The Java class file is portable, while the Java compiler and JVM are not portable. Therefore, a Java program is "write once run anywhere."

Here is an example to show the process. First, compose the following source code in a text editor such as Windows Notepad, or UltraEdit. The following code implements the `Car` class.

Code 1.1 Car.java

```
public class Car {

    public Car() {
        name = "N/A";
    }
    public Car(String name) {
        this.name = name;
    }

    public void moveAhead(int distance) {
        switch (direction ) {
          case 0: y = y + distance; break;
          case 1: x = x + distance; break;
          case 2: y = y - distance; break;
          case 3: x = x - distance; break;
        }
    }

    private String name;
    private double width = 40, height = 30;
    private int x = 400, y = 300;
```

```
    private byte direction = 0;      //0: top; 1: right; 2: bottom; 3: left
}
```

Save the program as `E:\workspace\Car.java`. Assuming device E is your usual workspace device. Secondly, create another new file and compose the following source code:

```
public class CarRace {
    public static void main(String[] args) {
        Car donald = new Car("Donald");
        donald.moveAhead(100);
    }
}
```

Save the program as `E:\workspace\CarRace.java`.

After that, open a command prompt window, go to folder `E:\workspace`, and type the following commands as shown in Fig. 1.8.

```
javac Car.java
javac -cp . CarRace.java
```

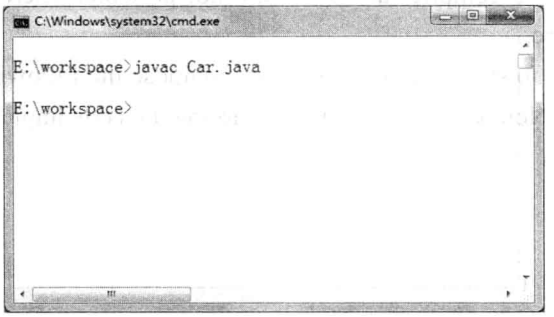

Fig. 1.8 Compiling in a command prompt window

The command `javac` will compile the source code. If there is any error in the source code, return to the editor to modify the source code and save it again. Use `javac` to compile again. Repeat the above procedure until the compiler completes its work without any error message.

Finally, you type in the following command to run your program:

```
java -cp . CarRace
```

In this case, the Java program consists of two class declarations, `Car` and `CarRace`. A Java program is a set of class declarations, in fact. The body of a class declaration starts from the first opening brace "{" and ends with the final closing brace "}". Inside the body, there are a few method declarations. Like a class declaration, braces also delimit a method declaration. All Java programs have at least one `main()` method, which is the entrance of a program. There are two lines of code in the `main()` method. There is only one statement on each line. The first statement declares a reference variable `donald` to refer to the instance of `Car` created. The second statement sends a `moveAhead` message to the object, `donald`. Alternatively, we can