PEARSON Prentice Hall

大学计算机教育国外著名教材系列

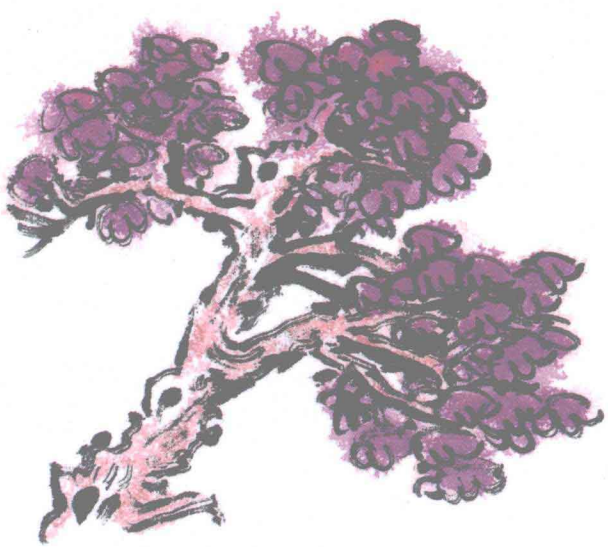
影印版

The Art and Science of Java

An Introduction to Computer Science

Java语言艺术与科学

计算机科学导论



Eric S. Roberts 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）

The Art and Science of Java

An Introduction to Computer Science

Java 语言艺术与科学

计算机科学导论

Eric S. Roberts

Stanford University

清华大学出版社

北 京

图书在版编目(CIP)数据

Java 语言艺术与科学：计算机科学导论 = The Art and Science of Java: An Introduction to Computer Science: 英文 /（美）罗伯特（Roberts, E. S.）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 978-7-302-19805-5

Ⅰ. J… Ⅱ. 罗… 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 045643 号

责任印制： 杨 艳

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： 北京市密云县京文制本装订厂

发 行 者： 全国新华书店

开 本： 185×230 **印张：** 38.25

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： 49.00 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：025055-01

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

Dedicated to the memory of Anita Borg (1949–2003)
for her tireless efforts to bring the excitement
of computing to a wider audience.

To the Student

Welcome! By picking up this book, you have taken a step into the world of computer science—a field of study that has grown from almost nothing half a century ago to one of the most vibrant and active disciplines of our time.

Over that time, the computer has opened up extraordinary possibilities in almost every area of human endeavor. Business leaders today are able to manage global enterprises on an unprecedented scale because computers enable them to transfer data anywhere in a fraction of a second. Scientists can now solve problems that were beyond their reach in the days before computers made the necessary calculations possible. The World Wide Web puts a vast amount of information at your fingertips and lies at the foundation of a vibrant industry that did not exist a decade ago. Filmmakers can use computer technology to create animated features that would have been unthinkable in Walt Disney's time. Modern computation has revolutionized many fields, enabling biologists to sequence the human genome, economists to model international markets, and literary scholars to assess whether an unattributed Elizabethan manuscript might have been penned by Shakespeare.

Computing is a profoundly empowering technology. The computing industry continues to grow, with more jobs available today than at the height of the Internet boom of the 1990s. But the advances we have seen up to now are small compared to what we will experience in this century. Those of you who are students today will soon inherit the responsibility of guiding that progress. No matter what field you choose, understanding how to use computing effectively will be of enormous value.

Like most skills that are worth knowing, learning how computers work and how to control their enormous power takes time. You will not understand it all at once. You must start somewhere. Twenty-five centuries ago, the Chinese philosopher Lao-tzu observed that the longest journey begins with a single step. This book can be your beginning.

For many of you, however, the first step can be the hardest to take. Many students find computers overwhelming and imagine that computer science is beyond their reach. Learning the basics of programming, however, does not require advanced mathematics or a detailed understanding of electronics. What matters in programming is whether you can progress from the statement of a problem to its solution. To do so, you must be able to think logically. You must have the necessary discipline to express your logic in a form that the computer can understand. Perhaps most importantly, you must be able to see the task through to its completion without getting discouraged by difficulties and setbacks. If you stick with the process, you will discover that reaching the solution is so exhilarating that it more than makes up for any frustrations you encounter along the way.

I wish you a pleasant journey along that road.

Eric Roberts
Stanford University
January 2007

To the Instructor

This text is intended for use in the first programming course in a typical college or university curriculum. It covers the material in a traditional CS1 course, as defined in the *Curriculum '78* report prepared by the Association for Computing Machinery (ACM). It also includes the full set of topics specified for the $CS101_O$ or $CS111_O$ courses in the computer science volume of the more recent *Computing Curriculum 2001* report.

The Art and Science of Java uses a similar approach to that of my 1995 textbook, *The Art and Science of C*. Each of these texts uses libraries to make programming less complex and consequently more accessible to the novice. In the C-based version, those libraries proved to be extremely successful with students, not only at Stanford, but at many other institutions as well. This book uses the ACM Java Libraries to achieve the same goals.

In the years since its initial release in 1995, the Java programming language has become increasingly important as an instructional language, to the point that it is now something of a standard in introductory computing courses. On the positive side, Java offers many advantages over earlier teaching languages, primarily by making it possible for students to write highly interactive programs that capture their interest and imagination. At the same time, Java is far more sophisticated than languages that have traditionally filled the role of teaching languages, such as BASIC and Pascal. The complexity that accompanies Java's sophistication can be a significant barrier to both teachers and students as they try to understand the structure of the language.

To address the problems that introductory instructors encountered using Java, in 2004 the ACM established the Java Task Force and gave it the following charge:

> To review the Java language, APIs, and tools from the perspective of introductory computing education and to develop a stable collection of pedagogical resources that will make it easier to teach Java to first-year computing students without having those students overwhelmed by its complexity.

Over the next two years, the Java Task Force developed a new set of libraries intended to support the use of Java at the introductory level. After releasing two preliminary drafts to obtain community feedback, the Java Task Force published its final report in the summer of 2006. The ACM Java Libraries described in that report are available on the following web site:

`http://jtf.acm.org/`

In addition to the ACM Java Libraries themselves, the web site contains a large collection of demo programs, a tutorial guide, an extensive discussion of the rationale behind the design of the various packages, and an executive summary that identifies the following as the greatest strengths of the libraries. The ACM Java Libraries provide

- *A simple object-oriented model for programs.* The **Program** class defined in the **acm.program** package offers an easy-to-use model for writing simple programs. In addition to hiding the static **main** method, the **Program** class and its standard subclasses provide a highly intuitive example of object-oriented class hierarchies.
- *A model for input and output that treats traditional console I/O and dialog I/O symmetrically.* The **acm.io** package defines the classes **IOConsole** and **IODialog** that share a common interface for all input/output operations. This design addresses Java's lack of a simple input mechanism in a way that emphasizes the value of interface-based design.
- *An extensive library of graphical objects.* The **acm.graphics** package implements a simple but extremely powerful model for creating graphical pictures based on the metaphor of a felt board in which students construct graphical objects of various types and place them on a canvas. This design emphasizes the use of objects and frees the student from having to respond explicitly to repaint requests.
- *A minimal set of new classes to support development of graphical user interfaces.* The **acm.gui** package includes a small set of classes to bring Java's extensive GUI-development resources within the reach of novice programmers.
- *Backward compatibility for applets.* Unlike most Java code today, programs developed using the ACM Java Libraries can typically be executed as applets even on older web browsers. This flexibility makes these libraries an ideal foundation for web-based teaching tools and lecture demonstrations.

Given the enormous number of classes and methods that are available in Java's own libraries, it is inevitable that a book of this sort will leave out some particular feature of Java. It is impossible to cover everything and still produce a book that is accessible to the average student. In terms of the choice of topics, this book seeks to offer a coherent presentation that teaches the fundamentals of computer science rather than to cover everything there is to know about Java. In many ways, the guiding principle for the choice of topics in this book comes from the following observation from Antoine de Saint-Exupéry in his 1942 memoire *Pilote de Guerre* (later quoted by Tony Hoare in his 1980 Turing Award lecture):

> Perfection is attained, not when there is nothing left to add, but when there is nothing left to take away.

While this book certainly makes no claims to perfection, it would probably have been better to leave more topics out than to put more topics in.

Although the book covers topics in an order that has proven successful here at Stanford, you may want to vary the order of presentation to suit your audience and the goals of your course. The following notes provide an overview of the chapters and indicate some of the more important dependencies.

Chapter 1 traces the history of computing and describes the programming process. The chapter requires no programming *per se* but provides the contextual background for the rest of the text.

I have designed Chapter 2 for students with little or no computing background. This chapter is conceptual in its approach and focuses on developing a holistic understanding of object-oriented programming rather than on the details of the Java language. When new students are faced with detailed rules of syntax and structure, they concentrate on learning the rules instead of the underlying concepts, which are vastly more important at this stage. If your students already know some programming, you can probably move quickly through this material.

Chapters 3, 4, and 5 offer relatively traditional introductions to expressions, statements, and methods so that students understand these basic concepts.

Chapters 6 and 7 then go on to introduce the fundamentals of objects and classes. Chapter 6 provides the high-level view, focusing on how to use objects and classes rather than on their underlying structure. Chapter 7 then turns to the low-level details involved of how objects are represented in memory. Although Chapter 7 is not absolutely required, students who can picture the internal structure of an object are much more likely to understand the essential concept of a reference.

The next three chapters introduce particular classes from either Java's standard libraries or the ACM packages. Chapter 8 covers the **`String`** class, which is presumably an important topic in any introductory course. Chapter 9 describes the **`acm.graphics`** package in detail, which makes it possible for students to write far more exciting programs. Parts of the **`acm.graphics`** package are included in earlier chapters, and it is certainly possible to cover only some of the topics in Chapter 9. Chapter 10 offers a similar overview of event-driven programming. The first several sections in Chapter 10 focus on mouse and keyboard events; the remaining sections provide an introduction to graphical user interfaces (GUIs) and the standard interactor classes from **`javax.swing`**. This second part of the chapter is valuable if you want to have students design GUI-based programs, but it is not necessary to understand subsequent chapters.

Chapters 11 and 12 address the idea of arrays, but do so from different perspectives. Chapter 11 introduces both the built-in array type and the **`ArrayList`** utility class from the **`java.util`** package. Both of these topics seem essential to any Java-based introductory course. Chapter 12 focuses on algorithms for searching and sorting arrays. The chapter also includes a brief discussion of computational complexity that will help students understand the importance of algorithmic design.

Chapter 13 describes the Java Collections Framework, which is perhaps more often presented in a second programming course. However, because Java takes care of so many of the underlying details, it is actually quite reasonable to teach introductory students how to *use* these classes, even if they can't understand their implementation. The only essential class from the Java Collections Framework is **`ArrayList`**, which has already appeared in Chapter 11.

Chapter 14 includes four important topics that sometimes appear in a first programming course: recursion, concurrency, networking, and programming patterns. At Stanford, which is on the quarter system, we teach these topics in the second course. If you decide to teach recursion in the first course, I strongly recommend that you do so early enough to allow students time to assimilate the material. One possibility is to discuss recursive functions immediately after Chapter 5 and recursive algorithms at the end of Chapter 12.

Supplemental Resources

For students

The following items are available to all readers of this book at the Addison-Wesley web site (**`http://www.aw.com/cssupport/`**):

- Source code files for each example program in the book
- Full-color PDF versions of sample runs
- Answers to review questions

For instructors

The following items are available to qualified instructors from Addison-Wesley's Instructor Resource Center (**`http://www.aw.com/irc/`**):

- Source code files for each example program in the book
- Full-color PDF versions of sample runs
- Answers to review questions
- Solutions to programming exercises
- Applet-based lecture slides that include animations of the program examples

For adopters of the ACM Java Libraries

The Association for Computing Machinery maintains an extensive web site on the ACM Java Libraries developed by the Java Task Force (**`http://jtf.acm.org/`**). That site includes the following resources:

- An executive summary outlining the purpose of the ACM Java Libraries
- Downloadable copies of ACM libraries in both source and compiled form
- An extensive demo gallery including source code for the examples
- An introductory tutorial to using the ACM libraries
- A comprehensive discussion of the rationale behind the design

Acknowledgments

Writing a textbook is never the work of a single individual. In putting this book together, I have been extremely fortunate to have the help of many talented and dedicated people. I particularly want to thank my colleagues on the ACM Java Task Force—Kim Bruce, James H. Cross II, Robb Cutler, Scott Grissom, Karl Klee, Susan Rodger, Fran Trees, Ian Utting, and Frank Yellin—for their hard work on the project, as well as the National Science Foundation, the ACM Education Board, and the SIGCSE Special Projects Fund for their financial support. I also want to thank everyone who responded to the Java Task Force's call for proposals in 2004: Alyce Brady, Kim Bruce, Pam Cutter, Ken Lambert, Robert McCartney, Dave Musicant, Martin Osborne, Nick Parlante, Viera Proulx, Richard Rasala, Juris Reinfelds, Dean Sanders, Kathryn Sanders, Ruth Ungar, and Andries van Dam. As the design document for the Java Task Force makes clear, these suggestions were of enormous value even if the task force did not adopt the designs in their original form.

Here at Stanford, thanks are due to many people. In many ways, the people who have shaped the book as much as anyone have been my students, who have had to learn the material from a series of preliminary drafts over the last year and a half. For one thing, the students in Stanford's introductory course have certainly risen to the challenge of using a new approach to the material and have demonstrated beyond my expectations how much students can accomplish using the ACM Java Libraries. For another, my students have proven to be vigilant readers of the various drafts, never hesitating to point out opportunities for improvement. I also want to thank the entire team of undergraduate teaching assistants, who have had to explain to students all the concepts I left out of the earlier versions.

I want to express my gratitude to my editor, Michael Hirsch, and the other members of the team at Addison-Wesley for their support on this book as well as its predecessor.

As always, the greatest thanks are due to my wife Lauren Rusk, who has again worked her magic as my developmental editor. Lauren's expertise has added considerable clarity and polish to the text. As I said in the preface to my 1995 book, without her, nothing would ever come out as well as it should.

Contents