

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

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

English reprint edition copyright © 2009 by PEARSON EDUCATION ASIA LIMITED and TSINGHUA UNIVERSITY PRESS.

Original English language title from Proprietor's edition of the Work.

Original English language title: The Art and Science of Java: An Introduction to Computer Science by Eric S. Roberts, Copyright © 2009

All Rights Reserved.

Published by arrangement with the original publisher, Pearson Education, Inc., publishing as Addison-Wesley, Inc.

This edition is authorized for sale and distribution only in the People's Republic of China (excluding the Special Administrative Region of Hong Kong, Macao SAR and Taiwan).

本书影印版由 Pearson Education(培生教育出版集团)授权给清华大学出版社出版发行。

**For sale and distribution in the People's Republic of China exclusively
(except Taiwan, Hong Kong SAR and Macao SAR).**

**仅限于中华人民共和国境内(不包括中国香港、澳门特别行政区和
中国台湾地区)销售发行。**

本书封面贴有 Pearson Education(培生教育出版集团)激光防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

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

<http://www.tup.com.cn>

社 总 机: 010-62770175

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

地 址: 北京清华大学学研大厦 A 座

邮 编: 100084

邮 购: 010-62786544

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 CS101O or CS111O 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 memoir *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 **javaw.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

1	Introduction	1
1.1	A brief history of computing	2
1.2	What is computer science?	4
1.3	A brief tour of computer hardware	5
	The CPU 6, Memory 7, Secondary storage 7, Input/output devices 7, Network 8	
1.4	Algorithms	8
1.5	Stages in the programming process	9
	Creating and editing programs 9, Programming errors and debugging 12, Software maintenance 13	
1.6	Java and the object-oriented paradigm	15
	The history of object-oriented programming 15, The Java programming language 16	
1.7	Java and the World Wide Web	17
	Summary	21
	Review questions	22
2	Programming by Example	23
2.1	The "Hello world" program	24
	Comments 25, Imports 26, The main class 26,	
2.2	Perspectives on the programming process	28
2.3	A program to add two numbers	30
2.4	Programming idioms and patterns	34
2.5	Classes and objects	36
	Class hierarchies 36, The Program class hierarchy 38	
2.6	Graphical programs	39
	The HelloProgram example revisited 40, Sending messages to GObject s 40, The GObject class hierarchy 43, The GRect class 44, The G Oval class 48, The GLine class 49	
	Summary	51
	Review questions	52
	Programming exercises	53
3	Expressions	57
3.1	Primitive data types	59
3.2	Constants and variables	61
	Constants 61, Variables 62, Declarations 63, Named constants 64	

3.3 Operators and operands 65

Combining integers and floating-point numbers 66, Integer division and the remainder operator 67, Precedence 68, Applying rules of precedence 70, Type conversion 71

3.4 Assignment statements 73

Shorthand assignment operators 76, The increment and decrement operators 77

3.5 Boolean expressions 77

Relational operators 78, Logical operators 78, Short-circuit evaluation 81, Flags 82, An example of Boolean calculation 82

3.6 Designing for change 83

The importance of readability 83, Using named constants to support program maintenance 84, Using named constants to support program development 85

Summary 88

Review questions 89

Programming exercises 91

4 Statement Forms

95

4.1 Statement types in Java 96

Simple statements 96, Compound statements 98, Control statements 98

4.2 Control statements and problem solving 99

Generalizing the **Add2Integers** program 100, The repeat-N-times pattern 101, The read-until-sentinel pattern 102

4.3 The **if** statement 105

Single-line **if** statements 107, Multiline **if** statements 107, The **if-else** statement 107, Cascading **if** statements 107, The **?:** operator 108

4.4 The **switch** statement 110

4.5 The **while** statement 112

Using the **while** loop 113, Infinite loops 114, Solving the loop-and-a-half problem 115

4.6 The **for** statement 118

The relationship between **for** and **while** 120, Using **for** with floating-point data 120, Nested **for** statements 121, Simple graphical animation 122

Summary 126

Review questions 126

Programming exercises 127

5 Methods

133

5.1 A quick overview of methods 134

Methods as mechanisms for hiding complexity 135, Methods as tools for programmers rather than users 135, Method calls as expressions 136, Method calls as messages 137

5.2 Writing your own methods 138

The format of a method 139, The **return** statement 139, Methods involving internal control structures 141, Methods that return nonnumeric values 142, Predicate methods 144

5.3 Mechanics of the method-calling process 147

Parameter passing 148, Calling methods from within other methods 152

5.4 Decomposition 158

Stepwise refinement 159, Specifying parameters 161, Designing from the top down 163, Looking for common features 164, Completing the decomposition 165

5.5 Algorithmic methods 166

The “brute force” approach 166, Euclid’s algorithm 168, Defending the correctness of Euclid’s algorithm 168, Comparing the efficiency of the two algorithms 170

Summary 170

Review questions 171

Programming exercises 172

6 Objects and Classes

177

6.1 Using the **RandomGenerator** class 178

Pseudorandom numbers 179, Using the **RandomGenerator** class 180, The role of the random number seed 183

6.2 The **javadoc** documentation system 185

6.3 Defining your own classes 188

The structure of a class definition 188, Controlling the visibility of entries 189, Encapsulation 189

6.4 Representing student information 190

Declaring instance variables 190, Completing the class definition 191, Writing **javadoc** comments 194, Writing the constructor 194, Getters and setters 196, The **toString** method 196, Defining named constants in a class 197, Using the **Student** class 198

6.5 Rational numbers 199

6.6 Extending existing classes 204

Creating a class to represent filled rectangles 204, Rules for inherited constructors 208, Rules for inherited methods 209

Summary 211

Review questions 213

Programming exercises 214

7 Objects and Memory 221

- 7.1 The structure of memory 222
 - Bits, bytes, and words 222, Binary and hexadecimal representations 223, Memory addresses 225
- 7.2 The allocation of memory to variables 226
 - Memory diagrams for the **Rational** class 227, Garbage collection 232
- 7.3 Primitive types versus objects 232
 - Parameter passing 234, Wrapper classes 236, Boxing and unboxing 238
- 7.4 Linking objects together 240
 - Message passing in linked structures: The beacons of Gondor 240, The internal representation of linked structures 243
 - Summary 244
 - Review questions 245
 - Programming exercises 247

8 Strings and Characters 249

- 8.1 The principle of enumeration 250
 - Representing enumerated types inside the machine 251, Representing enumerated types as integers 252, Defining new enumerated types 253
- 8.2 Characters 254
 - The data type **char** 254, The ASCII and Unicode coding systems 254, Character constants 257, Important properties of the Unicode representation 257, Special characters 258, Character arithmetic 259, Useful methods in the **Character** class 261, Control statements involving characters 263
- 8.3 Strings as an abstract idea 263
 - Holistic and reductionist views of strings 264, The notion of an abstract type 264
- 8.4 Using the methods in the **String** class 265
 - Determining the length of a string 265, Selecting characters from a string 267, Concatenation 267, Extracting parts of a string 269, Comparing one string with another 270, Searching within a string 271, Case conversion 273
- 8.5 A case study in string processing 273
 - Applying top-down design 273, Implementing **translateLine** 275, Taking spaces and punctuation into account 276, The **StringTokenizer** class 278, Completing the implementation 280
 - Summary 284
 - Review questions 284
 - Programming exercises 286

9 Object-oriented Graphics 295

- 9.1 The **acm.graphics** model 296
- 9.2 Structure of the **acm.graphics** package 297
 - The **GCanvas** class 297, Further details about the **Color** class 300, The **GPoint**, **GDimension**, and **GRectangle** classes 301, The **GMath** class 302, The **GObject** class and its subclasses 303
- 9.3 Using the shape classes 306
 - The **GLabel** class 306, The **GRect** class and its subclasses (**GRoundRect** and **G3DRect**) 311, The **G Oval** class 311, The **GLine** class 312, The **GArc** class 313, The **GImage** class 316, The **GPolygon** class 321
- 9.4 Creating compound objects 328
 - A simple **GCompound** example 328, The **GCompound** coordinate system 331, Object decomposition using **GCompound** 331, Nesting **GCompound** objects 336
 - Summary 338
 - Review questions 339
 - Programming exercises 340

10 Event-driven Programs 349

- 10.1 The Java event model 350
- 10.2 A simple event-driven program 351
- 10.3 Responding to mouse events 354
 - The **MouseListener** and **MouseMotionListener** interfaces 355, Overriding listener methods 355, A line-drawing program 356, Dragging objects on the canvas 358
- 10.4 Responding to keyboard events 360
- 10.5 Creating a simple GUI 362
- 10.6 The Swing interactor hierarchy 365
 - The **JButton** class 365, The **JToggleButton** class 367, The **JCheckBox** class 368, The **JRadioButton** and **ButtonGroup** classes 369, The **JSlider** and **JLabel** classes 371, The **JComboBox** class 372, The **JTextField**, **IntField**, and **DoubleField** classes 374
- 10.7 Managing component layout 380
 - The Java windowing hierarchy 380, Layout managers 382, The **BorderLayout** manager 383, The **FlowLayout** manager 385, The **GridLayout** manager 386, The inadequacy of the standard layout managers 387
- 10.8 Using the **TableLayout** class 388
 - Comparing **GridLayout** and **TableLayout** 388, Using **TableLayout** to create a temperature converter 389, Specifying

constraints 390, Using **TableLayout** to create a simple calculator 390

Summary 399

Review questions 401

Programming exercises 402

11 Arrays and ArrayLists

409

11.1 Introduction to arrays 410

Array declaration 411, Array selection 412, An example of a simple array 413, Changing the index range 414, Arrays of objects 415, Using arrays in graphical programs 415, A digression on the ++ and -- operators 417

11.2 Internal representation of arrays 419

11.3 Passing arrays as parameters 421

11.4 Using arrays for tabulation 427

11.5 Initialization of arrays 428

11.6 Multidimensional arrays 430

Passing multidimensional arrays to methods 431, Initializing multidimensional arrays 432

11.7 Image processing 432

Representation of images 433, Using the **GImage** class to manipulate images 433, Bit manipulation 434, Using bit operations to isolate components of a pixel 437, Creating a grayscale image 438, Smoothing an image through averaging 439, Hiding complexity 439

11.8 The **ArrayList** class 442

Summary 448

Review questions 449

Programming exercises 450

12 Searching and Sorting

461

12.1 Searching 462

Searching in an integer array 462, Searching a table 463, Binary search 466, The relative efficiency of the search algorithms 468

12.2 Sorting 470

Sorting an integer array 470, The selection sort algorithm 471, Evaluating the efficiency of selection sort 473, Measuring the running time of a program 475, Analyzing the selection sort algorithm 476, The radix sort algorithm 478

12.3 Assessing algorithmic efficiency 481

Big-O notation 481, Standard simplifications of big-O 482, The computational complexity of selection sort 482, Predicting