

GLOBAL  
EDITION



# Building Java™ Programs

## *A Back to Basics Approach*

FOURTH EDITION

Stuart Reges • Marty Stepp



**Fourth Edition**  
**Global Edition**

# **Building Java Programs**

## **A Back to Basics Approach**

Stuart Reges  
*University of Washington*

Marty Stepp  
*Stanford University*



**Pearson**

---

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

**Vice President, Editorial Director:** Marcia Horton  
**Acquisitions Editor:** Matt Goldstein  
**Editorial Assistant:** Kristy Alaura  
**Acquisitions Editor, Global Editions:** Sourabh Maheshwari  
**VP of Marketing:** Christy Lesko  
**Director of Field Marketing:** Tim Galligan  
**Product Marketing Manager:** Bram Van Kempen  
**Field Marketing Manager:** Demetrius Hall  
**Marketing Assistant:** Jon Bryant  
**Director of Product Management:** Erin Gregg  
**Team Lead, Program and Project Management:**  
Scott Disanno  
**Program Manager:** Carole Snyder  
**Project Editor, Global Editions:** K.K. Neelakantan

**Project Manager:** Lakeside Editorial Services L.L.C.  
**Senior Specialist, Program Planning and Support:**  
Maura Zaldivar-Garcia  
**Senior Manufacturing Controller, Global Editions:** Kay  
Holman  
**Media Production Manager, Global Editions:** Vikram  
Kumar  
**Cover Design:** Lumina Datamatics  
**R&P Manager:** Rachel Youdelman  
**R&P Project Manager:** Timothy Nicholls  
**Inventory Manager:** Meredith Maresca  
**Cover Art:** © Westend61 Premium/Shutterstock.com  
**Full-Service Project Management:**  
Apoorva Goel/Cenveo® Publisher Services

---

The authors and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in this book. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

Acknowledgements of third-party content appear on pages 1219–1220, which constitute an extension of this copyright page.

PEARSON, and MYPROGRAMMINGLAB are exclusive trademarks in the U.S. and/or other countries owned by Pearson Education, Inc. or its affiliates.

Pearson Education Limited  
Edinburgh Gate  
Harlow  
Essex CM20 2JE  
England

and Associated Companies throughout the world

Visit us on the World Wide Web at:  
[www.pearsonglobaleditions.com](http://www.pearsonglobaleditions.com)

© Pearson Education Limited 2018

The rights of Stuart Reges and Marty Stepp to be identified as the authors of this work have been asserted by them in accordance with the Copyright, Designs and Patents Act 1988.

*Authorized adaptation from the United States edition, entitled Building Java Programs: A Back to Basics Approach, 4th Edition, ISBN 978-0-13-432276-6, by Stuart Reges and Marty Stepp published by Pearson Education © 2017.*

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without either the prior written permission of the publisher or a license permitting restricted copying in the United Kingdom issued by the Copyright Licensing Agency Ltd, Saffron House, 6–10 Kirby Street, London EC1N 8TS.

All trademarks used herein are the property of their respective owners. The use of any trademark in this text does not vest in the author or publisher any trademark ownership rights in such trademarks, nor does the use of such trademarks imply any affiliation with or endorsement of this book by such owners.

British Library Cataloguing-in-Publication Data  
A catalogue record for this book is available from the British Library  
10 9 8 7 6 5 4 3 2 1

ISBN 10: 1-292-16168-X  
ISBN 13: 978-1-292-16168-6

Typeset in Monotype by Cenveo Publisher Services  
Printed and bound in Malaysia.

The newly revised fourth edition of our *Building Java Programs* textbook is designed for use in a two-course introduction to computer science. We have class-tested it with thousands of undergraduates, most of whom were not computer science majors, in our CS1-CS2 sequence at the University of Washington. These courses are experiencing record enrollments, and other schools that have adopted our textbook report that students are succeeding with our approach.

Introductory computer science courses are often seen as “killer” courses with high failure rates. But as Douglas Adams says in *The Hitchhiker’s Guide to the Galaxy*, “Don’t panic.” Students can master this material if they can learn it gradually. Our textbook uses a layered approach to introduce new syntax and concepts over multiple chapters.

Our textbook uses an “objects later” approach where programming fundamentals and procedural decomposition are taught before diving into object-oriented programming. We have championed this approach, which we sometimes call “back to basics,” and have seen through years of experience that a broad range of scientists, engineers, and others can learn how to program in a procedural manner. Once we have built a solid foundation of procedural techniques, we turn to object-oriented programming. By the end of the course, students will have learned about both styles of programming.

Here are some of the changes that we have made in the fourth edition:

- **New chapter on functional programming with Java 8.** As explained below, we have introduced a chapter that uses the new language features available in Java 8 to discuss the core concepts of functional programming.
- **New section on images and 2D pixel array manipulation.** Image manipulation is becoming increasingly popular, so we have expanded our `DrawingPanel` class to include features that support manipulating images as two-dimensional arrays of pixel values. This extra coverage will be particularly helpful for students taking an AP/CS A course because of the heavy emphasis on two-dimensional arrays on the AP exam.
- **Expanded self-checks and programming exercises.** Many chapters have received new self-check problems and programming exercises. There are roughly fifty total problems and exercises per chapter, all of which have been class-tested with real students and have solutions provided for instructors on our web site.

Since the publication of our third edition, Java 8 has been released. This new version supports a style of programming known as functional programming that is gaining in

popularity because of its ability to simply express complex algorithms that are more easily executed in parallel on machines with multiple processors. ACM and IEEE have released new guidelines for undergraduate computer science curricula, including a strong recommendation to cover functional programming concepts.

We have added a new Chapter 19 that covers most of the functional concepts from the new curriculum guidelines. The focus is on concepts, not on language features. As a result, it provides an introduction to several new Java 8 constructs but not a comprehensive coverage of all new language features. This provides flexibility to instructors since functional programming features can be covered as an advanced independent topic, incorporated along the way, or skipped entirely. Instructors can choose to start covering functional constructs along with traditional constructs as early as Chapter 6. See the dependency chart at the end of this section.

The following features have been retained from previous editions:

- **Focus on problem solving.** Many textbooks focus on language details when they introduce new constructs. We focus instead on problem solving. What new problems can be solved with each construct? What pitfalls are novices likely to encounter along the way? What are the most common ways to use a new construct?
- **Emphasis on algorithmic thinking.** Our procedural approach allows us to emphasize algorithmic problem solving: breaking a large problem into smaller problems, using pseudocode to refine an algorithm, and grappling with the challenge of expressing a large program algorithmically.
- **Layered approach.** Programming in Java involves many concepts that are difficult to learn all at once. Teaching Java to a novice is like trying to build a house of cards. Each new card has to be placed carefully. If the process is rushed and you try to place too many cards at once, the entire structure collapses. We teach new concepts gradually, layer by layer, allowing students to expand their understanding at a manageable pace.
- **Case studies.** We end most chapters with a significant case study that shows students how to develop a complex program in stages and how to test it as it is being developed. This structure allows us to demonstrate each new programming construct in a rich context that can't be achieved with short code examples. Several of the case studies were expanded and improved in the second edition.
- **Utility as a CS1+CS2 textbook.** In recent editions, we added chapters that extend the coverage of the book to cover all of the topics from our second course in computer science, making the book usable for a two-course sequence. Chapters 12–19 explore recursion, searching and sorting, stacks and queues, collection implementation, linked lists, binary trees, hash tables, heaps, and more. Chapter 12 also

received a section on recursive backtracking, a powerful technique for exploring a set of possibilities for solving problems such as 8 Queens and Sudoku.

## Layers and Dependencies

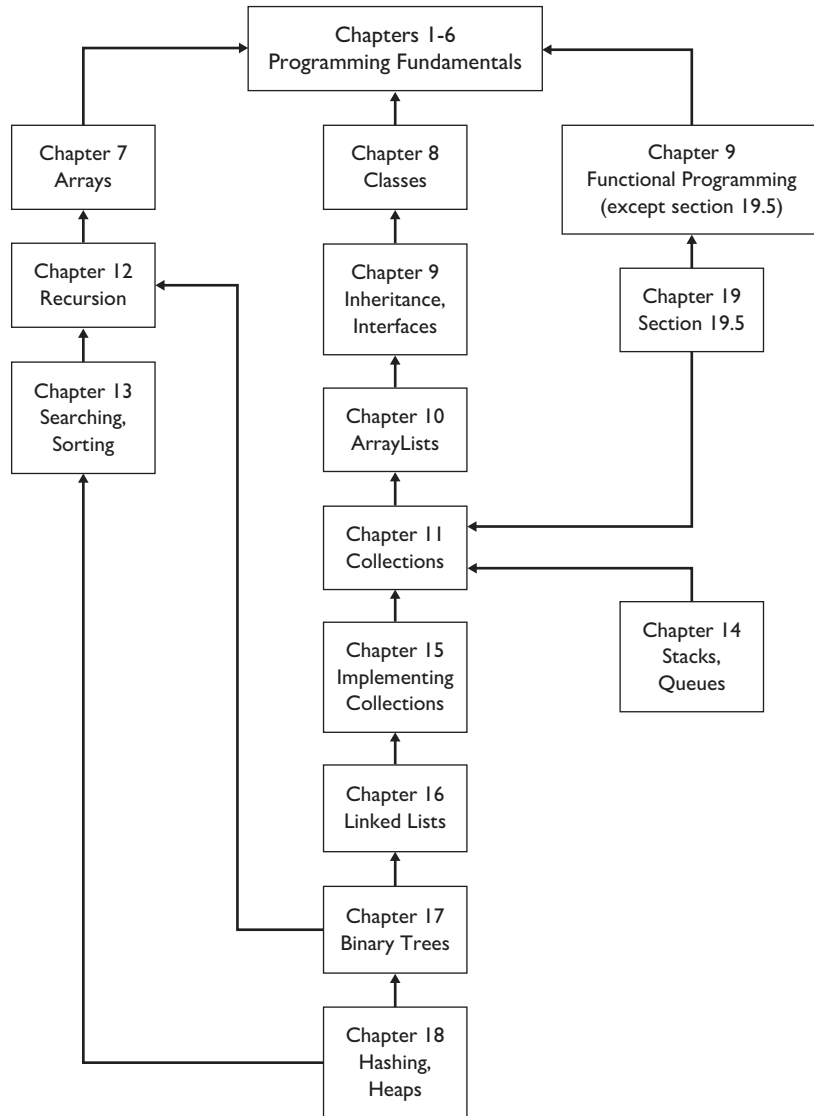
Many introductory computer science books are language-oriented, but the early chapters of our book are layered. For example, Java has many control structures (including for-loops, while-loops, and if/else-statements), and many books include all of these control structures in a single chapter. While that might make sense to someone who already knows how to program, it can be overwhelming for a novice who is learning how to program. We find that it is much more effective to spread these control structures into different chapters so that students learn one structure at a time rather than trying to learn them all at once.

The following table shows how the layered approach works in the first six chapters:

Chapter	Control Flow	Data	Programming Techniques	Input/Output
1	methods	String literals	procedural decomposition	println, print
2	definite loops (for)	variables, expressions, int, double	local variables, class constants, pseudocode	
3	return values	using objects	parameters	console input, 2D graphics (optional)
4	conditional (if/else)	char	pre/post conditions, throwing exceptions	printf
5	indefinite loops (while)	boolean	assertions, robust programs	
6		Scanner	token/line-based file processing	file I/O

Chapters 1–6 are designed to be worked through in order, with greater flexibility of study then beginning in Chapter 7. Chapter 6 may be skipped, although the case study in Chapter 7 involves reading from a file, a topic that is covered in Chapter 6.

The following is a dependency chart for the book:



## Supplements

Answers to all self-check problems appear on the web site and are accessible to anyone. Our web site has the following additional resources for students:

- **Online-only supplemental chapters**, such as a chapter on creating Graphical User Interfaces

- **Source code and data files** for all case studies and other complete program examples
- The **DrawingPanel** class used in the optional graphics Supplement 3G

Our web site has the following additional resources for teachers:

- **PowerPoint slides** suitable for lectures
- **Solutions** to exercises and programming projects, along with homework specification documents for many projects
- **Sample exams** and solution keys
- **Additional lab exercises** and **programming exercises** with solution keys
- **Closed lab creation tools** to produce lab handouts with the instructor's choice of problems integrated with the textbook

The materials are available at [www.pearsonglobaleditions.com/reges](http://www.pearsonglobaleditions.com/reges).

## MyProgrammingLab

MyProgrammingLab is an online practice and assessment tool that helps students fully grasp the logic, semantics, and syntax of programming. Through practice exercises and immediate, personalized feedback, MyProgrammingLab improves the programming competence of beginning students who often struggle with basic concepts and paradigms of popular high-level programming languages. A self-study and homework tool, the MyProgrammingLab course consists of hundreds of small practice exercises organized around the structure of this textbook. For students, the system automatically detects errors in the logic and syntax of code submissions and offers targeted hints that enable students to figure out what went wrong, and why. For instructors, a comprehensive grade book tracks correct and incorrect answers and stores the code inputted by students for review.

For a full demonstration, to see feedback from instructors and students, or to adopt MyProgrammingLab for your course, visit the following web site: <http://www.myprogramminglab.com/>

## VideoNotes



We have recorded a series of instructional videos to accompany the textbook. They are available at the following web site: [www.pearsonglobaleditions.com/reges](http://www.pearsonglobaleditions.com/reges).

Roughly 3–4 videos are posted for each chapter. An icon in the margin of the page indicates when a VideoNote is available for a given topic. In each video, we spend



5–15 minutes walking through a particular concept or problem, talking about the challenges and methods necessary to solve it. These videos make a good supplement to the instruction given in lecture classes and in the textbook. Your new copy of the textbook has an access code that will allow you to view the videos.

## Acknowledgments

First, we would like to thank the many colleagues, students, and teaching assistants who have used and commented on early drafts of this text. We could not have written this book without their input. Special thanks go to H el ene Martin, who pored over early versions of our first edition chapters to find errors and to identify rough patches that needed work. We would also like to thank instructor Benson Limketkai for spending many hours performing a technical proofread of the second edition.

Second, we would like to thank the talented pool of reviewers who guided us in the process of creating this textbook:

- Greg Anderson, Weber State University
- Delroy A. Brinkerhoff, Weber State University
- Ed Brunjes, Miramar Community College
- Tom Capaul, Eastern Washington University
- Tom Cortina, Carnegie Mellon University
- Charles Dierbach, Towson University
- H.E. Dunsmore, Purdue University
- Michael Eckmann, Skidmore College
- Mary Anne Egan, Siena College
- Leonard J. Garrett, Temple University
- Ahmad Ghafarian, North Georgia College & State University
- Raj Gill, Anne Arundel Community College
- Michael Hostetler, Park University
- David Hovemeyer, York College of Pennsylvania
- Chenglie Hu, Carroll College
- Philip Isenhour, Virginia Polytechnic Institute
- Andree Jacobson, University of New Mexico
- David C. Kamper, Sr., Northeastern Illinois University
- Simon G.M. Koo, University of San Diego
- Evan Korth, New York University
- Joan Krone, Denison University
- John H.E.F. Lasseter, Fairfield University

- Eric Matson, Wright State University
- Kathryn S. McKinley, University of Texas, Austin
- Jerry Mead, Bucknell University
- George Medelinskas, Northern Essex Community College
- John Neitzke, Truman State University
- Dale E. Parson, Kutztown University
- Richard E. Pattis, Carnegie Mellon University
- Frederick Pratter, Eastern Oregon University
- Roger Priebe, University of Texas, Austin
- Dehu Qi, Lamar University
- John Rager, Amherst College
- Amala V.S. Rajan, Middlesex University
- Craig Reinhart, California Lutheran University
- Mike Scott, University of Texas, Austin
- Alexa Sharp, Oberlin College
- Tom Stokke, University of North Dakota
- Leigh Ann Sudol, Fox Lane High School
- Ronald F. Taylor, Wright State University
- Andy Ray Terrel, University of Chicago
- Scott Thede, DePauw University
- Megan Thomas, California State University, Stanislaus
- Dwight Tuinstra, SUNY Potsdam
- Jeannie Turner, Sayre School
- Tammy VanDeGrift, University of Portland
- Thomas John VanDrunen, Wheaton College
- Neal R. Wagner, University of Texas, San Antonio
- Jiangping Wang, Webster University
- Yang Wang, Missouri State University
- Stephen Weiss, University of North Carolina at Chapel Hill
- Laurie Werner, Miami University
- Dianna Xu, Bryn Mawr College
- Carol Zander, University of Washington, Bothell

Finally, we would like to thank the great staff at Pearson who helped produce the book. Michelle Brown, Jeff Holcomb, Maurene Goo, Patty Mahtani, Nancy Kotary, and Kathleen Kenny did great work preparing the first edition. Our copy editors and the staff of Aptara Corp, including Heather Sisan, Brian Baker, Brendan Short,

**Preface**

and Rachel Head, caught many errors and improved the quality of the writing. Marilyn Lloyd and Chelsea Bell served well as project manager and editorial assistant respectively on prior editions. For their help with the third edition we would like to thank Kayla Smith-Tarbox, Production Project Manager, and Jenah Blitz-Stoehr, Computer Science Editorial Assistant. Mohinder Singh and the staff at Aptara, Inc., were also very helpful in the final production of the third edition. For their great work on production of the fourth edition, we thank Louise Capulli and the staff of Lakeside Editorial Services, along with Carole Snyder at Pearson. Special thanks go to our lead editor at Pearson, Matt Goldstein, who has believed in the concept of our book from day one. We couldn't have finished this job without all of their hard work and support.

Stuart Reges  
Marty Stepp

## Acknowledgments for the Global Edition

Pearson would like to thank and acknowledge the following people for their contributions to the Global Edition.

**Contributor**

Ankur Saxena, Amity University

**Reviewers**

Arup Bhattacharya, RCC Institute of Technology

Soumen Mukherjee, RCC Institute of Technology

Khyat Sharma

# MyProgrammingLab™

Through the power of practice and immediate personalized feedback, MyProgrammingLab helps improve your students' performance.

## PROGRAMMING PRACTICE

With MyProgrammingLab, your students will gain first-hand programming experience in an interactive online environment.

## IMMEDIATE, PERSONALIZED FEEDBACK

MyProgrammingLab automatically detects errors in the logic and syntax of their code submission and offers targeted hints that enables students to figure out what went wrong and why.

## GRADUATED COMPLEXITY

MyProgrammingLab breaks down programming concepts into short, understandable sequences of exercises. Within each sequence the level and sophistication of the exercises increase gradually but steadily.

## DYNAMIC ROSTER

Students' submissions are stored in a roster that indicates whether the submission is correct, how many attempts were made, and the actual code submissions from each attempt.

## PEARSON eTEXT

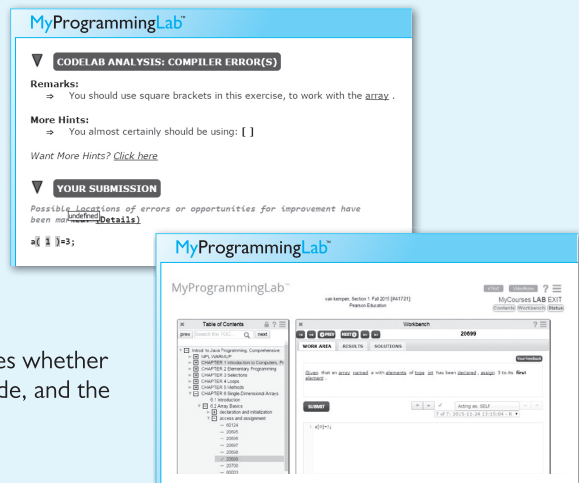
The Pearson eText gives students access to their textbook anytime, anywhere

## STEP-BY-STEP VIDEONOTE TUTORIALS

These step-by-step video tutorials enhance the programming concepts presented in select Pearson textbooks.

For more information and titles available with **MyProgrammingLab**, please visit [www.myprogramminglab.com](http://www.myprogramminglab.com).

Copyright © 2016 Pearson Education, Inc. or its affiliate(s). All rights reserved. HELO88173 • 11/15



## LOCATION OF VIDEO NOTES IN THE TEXT

[www.pearsonglobaleditions.com/reges](http://www.pearsonglobaleditions.com/reges)



VideoNote

<b>Chapter 1</b>	Pages 57, 66
<b>Chapter 2</b>	Pages 91, 100, 115, 123, 136
<b>Chapter 3</b>	Pages 167, 182, 187, 193
<b>Chapter 3G</b>	Pages 223, 241
<b>Chapter 4</b>	Pages 269, 277, 304
<b>Chapter 5</b>	Pages 350, 353, 355, 359, 382
<b>Chapter 6</b>	Pages 422, 435, 449
<b>Chapter 7</b>	Pages 484, 491, 510, 531
<b>Chapter 8</b>	Pages 561, 573, 581, 594
<b>Chapter 9</b>	Pages 623, 636, 652
<b>Chapter 10</b>	Pages 698, 703, 712
<b>Chapter 11</b>	Pages 742, 755, 763
<b>Chapter 12</b>	Pages 790, 798, 835
<b>Chapter 13</b>	Pages 860, 863, 869
<b>Chapter 14</b>	Pages 915, 922
<b>Chapter 15</b>	Pages 956, 962, 966
<b>Chapter 16</b>	Pages 998, 1005, 1018
<b>Chapter 17</b>	Pages 1063, 1064, 1074
<b>Chapter 18</b>	Pages 1099, 1118

# Brief Contents

<b>Chapter 1</b>	Introduction to Java Programming	27
<b>Chapter 2</b>	Primitive Data and Definite Loops	89
<b>Chapter 3</b>	Introduction to Parameters and Objects	163
<b>Supplement 3G</b>	Graphics (Optional)	222
<b>Chapter 4</b>	Conditional Execution	264
<b>Chapter 5</b>	Program Logic and Indefinite Loops	341
<b>Chapter 6</b>	File Processing	413
<b>Chapter 7</b>	Arrays	469
<b>Chapter 8</b>	Classes	556
<b>Chapter 9</b>	Inheritance and Interfaces	613
<b>Chapter 10</b>	ArrayLists	688
<b>Chapter 11</b>	Java Collections Framework	741
<b>Chapter 12</b>	Recursion	780
<b>Chapter 13</b>	Searching and Sorting	858
<b>Chapter 14</b>	Stacks and Queues	910
<b>Chapter 15</b>	Implementing a Collection Class	948
<b>Chapter 16</b>	Linked Lists	991
<b>Chapter 17</b>	Binary Trees	1043
<b>Chapter 18</b>	Advanced Data Structures	1097
<b>Chapter 19</b>	Functional Programming with Java 8	1133
<b>Appendix A</b>	Java Summary	1175
<b>Appendix B</b>	The Java API Specification and Javadoc Comments	1190
<b>Appendix C</b>	Additional Java Syntax	1196

This page intentionally left blank



# Contents

<b>Chapter 1 Introduction to Java Programming</b>	<b>27</b>
<b>1.1 Basic Computing Concepts</b>	<b>28</b>
Why Programming?	28
Hardware and Software	29
The Digital Realm	30
The Process of Programming	32
Why Java?	33
The Java Programming Environment	34
<b>1.2 And Now—Java</b>	<b>36</b>
String Literals (Strings)	40
<code>System.out.println</code>	41
Escape Sequences	41
<code>print</code> versus <code>println</code>	43
Identifiers and Keywords	44
A Complex Example: <code>DrawFigures1</code>	46
Comments and Readability	47
<b>1.3 Program Errors</b>	<b>50</b>
Syntax Errors	50
Logic Errors (Bugs)	54
<b>1.4 Procedural Decomposition</b>	<b>54</b>
Static Methods	57
Flow of Control	60
Methods That Call Other Methods	62
An Example Runtime Error	65
<b>1.5 Case Study: <code>DrawFigures</code></b>	<b>66</b>
Structured Version	67
Final Version without Redundancy	69
Analysis of Flow of Execution	70
<b>Chapter 2 Primitive Data and Definite Loops</b>	<b>89</b>
<b>2.1 Basic Data Concepts</b>	<b>90</b>
Primitive Types	90



Expressions	91
Literals	93
Arithmetic Operators	94
Precedence	96
Mixing Types and Casting	99
<b>2.2 Variables</b>	<b>100</b>
Assignment/Declaration Variations	105
String Concatenation	108
Increment/Decrement Operators	110
Variables and Mixing Types	113
<b>2.3 The for Loop</b>	<b>115</b>
Tracing for Loops	117
for Loop Patterns	121
Nested for Loops	123
<b>2.4 Managing Complexity</b>	<b>125</b>
Scope	125
Pseudocode	131
Class Constants	134
<b>2.5 Case Study: Hourglass Figure</b>	<b>136</b>
Problem Decomposition and Pseudocode	137
Initial Structured Version	139
Adding a Class Constant	140
Further Variations	143
<b>Chapter 3 Introduction to Parameters and Objects</b>	<b>163</b>
<b>3.1 Parameters</b>	<b>164</b>
The Mechanics of Parameters	167
Limitations of Parameters	171
Multiple Parameters	174
Parameters versus Constants	177
Overloading of Methods	177
<b>3.2 Methods That Return Values</b>	<b>178</b>
The Math Class	179
Defining Methods That Return Values	182
<b>3.3 Using Objects</b>	<b>186</b>
String Objects	187
Interactive Programs and Scanner Objects	193
Sample Interactive Program	196

<b>Contents</b>	<b>17</b>
<b>3.4 Case Study: Projectile Trajectory</b>	<b>199</b>
Unstructured Solution	203
Structured Solution	205
<b>Supplement 3G Graphics (Optional)</b>	<b>222</b>
<b>3G.1 Introduction to Graphics</b>	<b>223</b>
DrawingPanel	223
Drawing Lines and Shapes	224
Colors	229
Drawing with Loops	232
Text and Fonts	236
Images	239
<b>3G.2 Procedural Decomposition with Graphics</b>	<b>241</b>
A Larger Example: DrawDiamonds	242
<b>3G.3 Case Study: Pyramids</b>	<b>245</b>
Unstructured Partial Solution	246
Generalizing the Drawing of Pyramids	248
Complete Structured Solution	249
<b>Chapter 4 Conditional Execution</b>	<b>264</b>
<b>4.1 if/else Statements</b>	<b>265</b>
Relational Operators	267
Nested if/else Statements	269
Object Equality	276
Factoring if/else Statements	277
Testing Multiple Conditions	279
<b>4.2 Cumulative Algorithms</b>	<b>280</b>
Cumulative Sum	280
Min/Max Loops	282
Cumulative Sum with if	286
Roundoff Errors	288
<b>4.3 Text Processing</b>	<b>291</b>
The char Type	291
char versus int	292
Cumulative Text Algorithms	293
System.out.printf	295
<b>4.4 Methods with Conditional Execution</b>	<b>300</b>
Preconditions and Postconditions	300
Throwing Exceptions	300

	Revisiting Return Values	304
	Reasoning about Paths	309
<b>4.5</b>	<b>Case Study: Body Mass Index</b>	<b>311</b>
	One-Person Unstructured Solution	312
	Two-Person Unstructured Solution	315
	Two-Person Structured Solution	317
	Procedural Design Heuristics	321
<b>Chapter 5 Program Logic and Indefinite Loops</b>		<b>341</b>
<b>5.1</b>	<b>The while Loop</b>	<b>342</b>
	A Loop to Find the Smallest Divisor	343
	Random Numbers	346
	Simulations	350
	do/while Loop	351
<b>5.2</b>	<b>Fencepost Algorithms</b>	<b>353</b>
	Sentinel Loops	355
	Fencepost with if	356
<b>5.3</b>	<b>The boolean Type</b>	<b>359</b>
	Logical Operators	361
	Short-Circuited Evaluation	364
	boolean Variables and Flags	368
	Boolean Zen	370
	Negating Boolean Expressions	373
<b>5.4</b>	<b>User Errors</b>	<b>374</b>
	Scanner Lookahead	375
	Handling User Errors	377
<b>5.5</b>	<b>Assertions and Program Logic</b>	<b>379</b>
	Reasoning about Assertions	381
	A Detailed Assertions Example	382
<b>5.6</b>	<b>Case Study: NumberGuess</b>	<b>387</b>
	Initial Version without Hinting	387
	Randomized Version with Hinting	389
	Final Robust Version	393
<b>Chapter 6 File Processing</b>		<b>413</b>
<b>6.1</b>	<b>File-Reading Basics</b>	<b>414</b>
	Data, Data Everywhere	414

<b>Contents</b>	<b>19</b>
Files and File Objects	414
Reading a File with a Scanner	417
<b>6.2 Details of Token-Based Processing</b>	<b>422</b>
Structure of Files and Consuming Input	424
Scanner Parameters	429
Paths and Directories	430
A More Complex Input File	433
<b>6.3 Line-Based Processing</b>	<b>435</b>
String Scanners and Line/Token Combinations	436
<b>6.4 Advanced File Processing</b>	<b>441</b>
Output Files with <code>PrintStream</code>	441
Guaranteeing That Files Can Be Read	446
<b>6.5 Case Study: Zip Code Lookup</b>	<b>449</b>
<b>Chapter 7 Arrays</b>	<b>469</b>
<b>7.1 Array Basics</b>	<b>470</b>
Constructing and Traversing an Array	470
Accessing an Array	474
A Complete Array Program	477
Random Access	481
Arrays and Methods	484
The For-Each Loop	487
Initializing Arrays	489
The <code>Arrays</code> Class	490
<b>7.2 Array-Traversal Algorithms</b>	<b>491</b>
Printing an Array	492
Searching and Replacing	494
Testing for Equality	497
Reversing an Array	498
String Traversal Algorithms	503
Functional Approach	504
<b>7.3 Reference Semantics</b>	<b>505</b>
Multiple Objects	507
<b>7.4 Advanced Array Techniques</b>	<b>510</b>
Shifting Values in an Array	510
Arrays of Objects	514
Command-Line Arguments	516
Nested Loop Algorithms	516

<b>7.5</b>	<b>Multidimensional Arrays</b>	<b>518</b>
	Rectangular Two-Dimensional Arrays	518
	Jagged Arrays	520
<b>7.6</b>	<b>Arrays of Pixels</b>	<b>525</b>
<b>7.7</b>	<b>Case Study: Benford's Law</b>	<b>530</b>
	Tallying Values	531
	Completing the Program	535
 <b>Chapter 8 Classes</b>		 <b>556</b>
<b>8.1</b>	<b>Object-Oriented Programming</b>	<b>557</b>
	Classes and Objects	558
	Point Objects	560
<b>8.2</b>	<b>Object State and Behavior</b>	<b>561</b>
	Object State: Fields	562
	Object Behavior: Methods	564
	The Implicit Parameter	567
	Mutators and Accessors	569
	The <code>toString</code> Method	571
<b>8.3</b>	<b>Object Initialization: Constructors</b>	<b>573</b>
	The Keyword <code>this</code>	578
	Multiple Constructors	580
<b>8.4</b>	<b>Encapsulation</b>	<b>581</b>
	Private Fields	582
	Class Invariants	588
	Changing Internal Implementations	592
<b>8.5</b>	<b>Case Study: Designing a Stock Class</b>	<b>594</b>
	Object-Oriented Design Heuristics	595
	<code>Stock</code> Fields and Method Headers	597
	<code>Stock</code> Method and Constructor Implementation	599
 <b>Chapter 9 Inheritance and Interfaces</b>		 <b>613</b>
<b>9.1</b>	<b>Inheritance Basics</b>	<b>614</b>
	Nonprogramming Hierarchies	615
	Extending a Class	617
	Overriding Methods	621

<b>9.2</b>	<b>Interacting with the Superclass</b>	<b>623</b>
	Calling Overridden Methods	623
	Accessing Inherited Fields	624
	Calling a Superclass's Constructor	626
	DividendStock Behavior	628
	The Object Class	630
	The equals Method	631
	The instanceof Keyword	634
<b>9.3</b>	<b>Polymorphism</b>	<b>636</b>
	Polymorphism Mechanics	639
	Interpreting Inheritance Code	641
	Interpreting Complex Calls	643
<b>9.4</b>	<b>Inheritance and Design</b>	<b>646</b>
	A Misuse of Inheritance	646
	Is-a Versus Has-a Relationships	649
	Graphics2D	650
<b>9.5</b>	<b>Interfaces</b>	<b>652</b>
	An Interface for Shapes	653
	Implementing an Interface	655
	Benefits of Interfaces	658
<b>9.6</b>	<b>Case Study: Financial Class Hierarchy</b>	<b>660</b>
	Designing the Classes	661
	Redundant Implementation	665
	Abstract Classes	668

## **Chapter 10 ArrayLists** **688**

<b>10.1</b>	<b>ArrayLists</b>	<b>689</b>
	Basic ArrayList Operations	690
	ArrayList Searching Methods	693
	A Complete ArrayList Program	696
	Adding to and Removing from an ArrayList	698
	Using the For-Each Loop with ArrayLists	702
	Wrapper Classes	703
<b>10.2</b>	<b>The Comparable Interface</b>	<b>706</b>
	Natural Ordering and compareTo	708
	Implementing the Comparable Interface	712
<b>10.3</b>	<b>Case Study: Vocabulary Comparison</b>	<b>718</b>
	Some Efficiency Considerations	718
	Version 1: Compute Vocabulary	721

Version 2: Compute Overlap	724
Version 3: Complete Program	729

## Chapter 11 Java Collections Framework **741**

<b>11.1 Lists</b>	<b>742</b>
Collections	742
LinkedList versus ArrayList	743
Iterators	746
Abstract Data Types (ADTs)	750
LinkedList Case Study: Sieve	752
<b>11.2 Sets</b>	<b>755</b>
Set Concepts	756
TreeSet versus HashSet	758
Set Operations	759
Set Case Study: Lottery	761
<b>11.3 Maps</b>	<b>763</b>
Basic Map Operations	764
Map Views (keySet and values)	766
TreeMap versus HashMap	767
Map Case Study: WordCount	768
Collection Overview	771

## Chapter 12 Recursion **780**

<b>12.1 Thinking Recursively</b>	<b>781</b>
A Nonprogramming Example	781
An Iterative Solution Converted to Recursion	784
Structure of Recursive Solutions	786
<b>12.2 A Better Example of Recursion</b>	<b>788</b>
Mechanics of Recursion	790
<b>12.3 Recursive Functions and Data</b>	<b>798</b>
Integer Exponentiation	798
Greatest Common Divisor	801
Directory Crawler	807
Helper Methods	811
<b>12.4 Recursive Graphics</b>	<b>814</b>

<b>12.5 Recursive Backtracking</b>	<b>818</b>
A Simple Example: Traveling North/East	819
8 Queens Puzzle	824
Solving Sudoku Puzzles	831
<b>12.6 Case Study: Prefix Evaluator</b>	<b>835</b>
Infix, Prefix, and Postfix Notation	835
Evaluating Prefix Expressions	836
Complete Program	839

## **Chapter 13 Searching and Sorting** **858**

<b>13.1 Searching and Sorting in the Java Class Libraries</b>	<b>859</b>
Binary Search	860
Sorting	863
Shuffling	864
Custom Ordering with Comparators	865
<b>13.2 Program Complexity</b>	<b>869</b>
Empirical Analysis	870
Complexity Classes	876
<b>13.3 Implementing Searching and Sorting Algorithms</b>	<b>878</b>
Sequential Search	879
Binary Search	880
Recursive Binary Search	883
Searching Objects	886
Selection Sort	877
<b>13.4 Case Study: Implementing Merge Sort</b>	<b>890</b>
Splitting and Merging Arrays	891
Recursive Merge Sort	894
Complete Program	897

## **Chapter 14 Stacks and Queues** **910**

<b>14.1 Stack/Queue Basics</b>	<b>911</b>
Stack Concepts	911
Queue Concepts	914
<b>14.2 Common Stack/Queue Operations</b>	<b>915</b>
Transferring Between Stacks and Queues	917
Sum of a Queue	918
Sum of a Stack	919



<b>14.3</b>	<b>Complex Stack/Queue Operations</b>	<b>922</b>
	Removing Values from a Queue	922
	Comparing Two Stacks for Similarity	924
<b>14.4</b>	<b>Case Study: Expression Evaluator</b>	<b>926</b>
	Splitting into Tokens	927
	The Evaluator	932
<b>Chapter 15 Implementing a Collection Class</b>		<b>948</b>
<b>15.1</b>	<b>Simple <code>ArrayIntList</code></b>	<b>949</b>
	Adding and Printing	949
	Thinking about Encapsulation	955
	Dealing with the Middle of the List	956
	Another Constructor and a Constant	961
	Preconditions and Postconditions	962
<b>15.2</b>	<b>A More Complete <code>ArrayIntList</code></b>	<b>966</b>
	Throwing Exceptions	966
	Convenience Methods	969
<b>15.3</b>	<b>Advanced Features</b>	<b>972</b>
	Resizing When Necessary	972
	Adding an Iterator	974
<b>15.4</b>	<b><code>ArrayList&lt;E&gt;</code></b>	<b>980</b>
<b>Chapter 16 Linked Lists</b>		<b>991</b>
<b>16.1</b>	<b>Working with Nodes</b>	<b>992</b>
	Constructing a List	993
	List Basics	995
	Manipulating Nodes	998
	Traversing a List	1001
<b>16.2</b>	<b>A Linked List Class</b>	<b>1005</b>
	Simple <code>LinkedIntList</code>	1005
	Appending <code>add</code>	1007
	The Middle of the List	1011
<b>16.3</b>	<b>A Complex List Operation</b>	<b>1018</b>
	Inchworm Approach	1023
<b>16.4</b>	<b>An <code>IntList</code> Interface</b>	<b>1024</b>

<b>16.5</b>	<b>LinkedList&lt;E&gt;</b>	<b>1027</b>
	Linked List Variations	1028
	Linked List Iterators	1031
	Other Code Details	1033
 <b>Chapter 17 Binary Trees</b>		<b>1043</b>
<b>17.1</b>	<b>Binary Tree Basics</b>	<b>1044</b>
	Node and Tree Classes	1047
<b>17.2</b>	<b>Tree Traversals</b>	<b>1048</b>
	Constructing and Viewing a Tree	1054
<b>17.3</b>	<b>Common Tree Operations</b>	<b>1063</b>
	Sum of a Tree	1063
	Counting Levels	1064
	Counting Leaves	1066
<b>17.4</b>	<b>Binary Search Trees</b>	<b>1067</b>
	The Binary Search Tree Property	1068
	Building a Binary Search Tree	1070
	The Pattern $x = \text{change}(x)$	1074
	Searching the Tree	1077
	Binary Search Tree Complexity	1081
<b>17.5</b>	<b>SearchTree&lt;E&gt;</b>	<b>1082</b>
 <b>Chapter 18 Advanced Data Structures</b>		<b>1097</b>
<b>18.1</b>	<b>Hashing</b>	<b>1098</b>
	Array Set Implementations	1098
	Hash Functions and Hash Tables	1099
	Collisions	1101
	Rehashing	1106
	Hashing Non-Integer Data	1109
	Hash Map Implementation	1112
<b>18.2</b>	<b>Priority Queues and Heaps</b>	<b>1113</b>
	Priority Queues	1113
	Introduction to Heaps	1115
	Removing from a Heap	1117
	Adding to a Heap	1118
	Array Heap Implementation	1120
	Heap Sort	1124

<b>Chapter 19 Functional Programming with Java 8</b>	<b>1133</b>
<b>19.1 Effect-Free Programming</b>	<b>1134</b>
<b>19.2 First-Class Functions</b>	<b>1137</b>
Lambda Expressions	1140
<b>19.3 Streams</b>	<b>1143</b>
Basic Idea	1143
Using Map	1145
Using Filter	1146
Using Reduce	1148
Optional Results	1149
<b>19.4 Function Closures</b>	<b>1150</b>
<b>19.5 Higher-Order Operations on Collections</b>	<b>1153</b>
Working with Arrays	1154
Working with Lists	1155
Working with Files	1159
<b>19.6 Case Study: Perfect Numbers</b>	<b>1160</b>
Computing Sums	1161
Incorporating Square Root	1164
Just Five and Leveraging Concurrency	1167
<b>Appendix A Java Summary</b>	<b>1175</b>
<b>Appendix B The Java API Specification and Javadoc Comments</b>	<b>1190</b>
<b>Appendix C Additional Java Syntax</b>	<b>1196</b>
<b>Index</b>	<b>1205</b>
<b>Credits</b>	<b>1219</b>

# Introduction to Java Programming

## Introduction

---

This chapter begins with a review of some basic terminology about computers and computer programming. Many of these concepts will come up in later chapters, so it will be useful to review them before we start delving into the details of how to program in Java.

We will begin our exploration of Java by looking at simple programs that produce output. This discussion will allow us to explore many elements that are common to all Java programs, while working with programs that are fairly simple in structure.

After we have reviewed the basic elements of Java programs, we will explore the technique of procedural decomposition by learning how to break up a Java program into several methods. Using this technique, we can break up complex tasks into smaller subtasks that are easier to manage and we can avoid redundancy in our program solutions.

### 1.1 Basic Computing Concepts

- Why Programming?
- Hardware and Software
- The Digital Realm
- The Process of Programming
- Why Java?
- The Java Programming Environment

### 1.2 And Now—Java

- String Literals (Strings)
- `System.out.println`
- Escape Sequences
- `print` versus `println`
- Identifiers and Keywords
- A Complex Example: `DrawFigures1`
- Comments and Readability

### 1.3 Program Errors

- Syntax Errors
- Logic Errors (Bugs)

### 1.4 Procedural Decomposition

- Static Methods
- Flow of Control
- Methods That Call Other Methods
- An Example Runtime Error

### 1.5 Case Study: `DrawFigures`

- Structured Version
- Final Version without Redundancy
- Analysis of Flow of Execution

## 1.1 Basic Computing Concepts

Computers are pervasive in our daily lives, and, thanks to the Internet, they give us access to nearly limitless information. Some of this information is essential news, like the headlines at [cnn.com](http://cnn.com). Computers let us share photos with our families and map directions to the nearest pizza place for dinner.

Lots of real-world problems are being solved by computers, some of which don't much resemble the one on your desk or lap. Computers allow us to sequence the human genome and search for DNA patterns within it. Computers in recently manufactured cars monitor each vehicle's status and motion. Digital music players such as Apple's iPod actually have computers inside their small casings. Even the Roomba vacuum-cleaning robot houses a computer with complex instructions about how to dodge furniture while cleaning your floors.

But what makes a computer a computer? Is a calculator a computer? Is a human being with a paper and pencil a computer? The next several sections attempt to address this question while introducing some basic terminology that will help prepare you to study programming.

### Why Programming?

At most universities, the first course in computer science is a programming course. Many computer scientists are bothered by this because it leaves people with the impression that computer science is programming. While it is true that many trained computer scientists spend time programming, there is a lot more to the discipline. So why do we study programming first?

A Stanford computer scientist named Don Knuth answers this question by saying that the common thread for most computer scientists is that we all in some way work with *algorithms*.

#### Algorithm

A step-by-step description of how to accomplish a task.

Knuth is an expert in algorithms, so he is naturally biased toward thinking of them as the center of computer science. Still, he claims that what is most important is not the algorithms themselves, but rather the thought process that computer scientists employ to develop them. According to Knuth,

It has often been said that a person does not really understand something until after teaching it to someone else. Actually a person does not *really* understand something until after teaching it to a *computer*, i.e., expressing it as an algorithm.<sup>1</sup>

<sup>1</sup>Knuth, Don. *Selected Papers on Computer Science*. Stanford, CA: Center for the Study of Language and Information, 1996.

Knuth is describing a thought process that is common to most of computer science, which he refers to as *algorithmic thinking*. We study programming not because it is the most important aspect of computer science, but because it is the best way to explain the approach that computer scientists take to solving problems.

The concept of algorithms is helpful in understanding what a computer is and what computer science is all about. The Merriam-Webster dictionary defines the word “computer” as “one that computes.” Using that definition, all sorts of devices qualify as computers, including calculators, GPS navigation systems, and children’s toys like the Furby. Prior to the invention of electronic computers, it was common to refer to humans as computers. The nineteenth-century mathematician Charles Peirce, for example, was originally hired to work for the U.S. government as an “Assistant Computer” because his job involved performing mathematical computations.

In a broad sense, then, the word “computer” can be applied to many devices. But when computer scientists refer to a computer, we are usually thinking of a universal computation device that can be programmed to execute any algorithm. Computer science, then, is the study of computational devices and the study of computation itself, including algorithms.

Algorithms are expressed as computer programs, and that is what this book is all about. But before we look at how to program, it will be useful to review some basic concepts about computers.

## Hardware and Software

A computer is a machine that manipulates data and executes lists of instructions known as *programs*.

### Program

A list of instructions to be carried out by a computer.

One key feature that differentiates a computer from a simpler machine like a calculator is its versatility. The same computer can perform many different tasks (playing games, computing income taxes, connecting to other computers around the world), depending on what program it is running at a given moment. A computer can run not only the programs that exist on it currently, but also new programs that haven’t even been written yet.

The physical components that make up a computer are collectively called *hardware*. One of the most important pieces of hardware is the central processing unit, or *CPU*. The CPU is the “brain” of the computer: It is what executes the instructions. Also important is the computer’s *memory* (often called random access memory, or *RAM*, because the computer can access any part of that memory at any time). The computer uses its memory to store programs that are being executed, along with their data. RAM is limited in size and does not retain its contents when the computer is turned off. Therefore, computers generally also use a *hard disk* as a larger permanent storage area.

Computer programs are collectively called *software*. The primary piece of software running on a computer is its operating system. An *operating system* provides an environment in which many programs may be run at the same time; it also provides a bridge between those programs, the hardware, and the *user* (the person using the computer). The programs that run inside the operating system are often called *applications*.

When the user selects a program for the operating system to run (e.g., by double-clicking the program's icon on the desktop), several things happen: The instructions for that program are loaded into the computer's memory from the hard disk, the operating system allocates memory for that program to use, and the instructions to run the program are fed from memory to the CPU and executed sequentially.

## The Digital Realm

In the last section, we saw that a computer is a general-purpose device that can be programmed. You will often hear people refer to modern computers as *digital* computers because of the way they operate.

### Digital

Based on numbers that increase in discrete increments, such as the integers 0, 1, 2, 3, etc.

Because computers are digital, everything that is stored on a computer is stored as a sequence of integers. This includes every program and every piece of data. An MP3 file, for example, is simply a long sequence of integers that stores audio information. Today we're used to digital music, digital pictures, and digital movies, but in the 1940s, when the first computers were built, the idea of storing complex data in integer form was fairly unusual.

Not only are computers digital, storing all information as integers, but they are also *binary*, which means they store integers as *binary numbers*.

### Binary Number

A number composed of just 0s and 1s, also known as a base-2 number.

Humans generally work with *decimal* or base-10 numbers, which match our physiology (10 fingers and 10 toes). However, when we were designing the first computers, we wanted systems that would be easy to create and very reliable. It turned out to be simpler to build these systems on top of binary phenomena (e.g., a circuit being open or closed) rather than having 10 different states that would have to be distinguished from one another (e.g., 10 different voltage levels).

From a mathematical point of view, you can store things just as easily using binary numbers as you can using base-10 numbers. But since it is easier to construct a physical device that uses binary numbers, that's what computers use.

This does mean, however, that people who aren't used to computers find their conventions unfamiliar. As a result, it is worth spending a little time reviewing how binary