

Java

Learning to Program with Robots

Byron Weber Becker, University of Waterloo

Java: Learning to Program with Robots

by Byron Weber Becker

Managing Editor:

Mary Franz

Senior Product Manager:

Alyssa Pratt

Production Editor:

Kelly Robinson

Developmental Editor:

Lisa Ruffolo

Associate Product Manager:

Jennifer Smith

Senior Marketing Manager:

Karen Seitz

Senior Manufacturing Coordinator:

Justin Palmeiro

Marketing Coordinator:

Suelaine Frongello

Cover Artist:

Joel Weber Becker

Cover Designer:

Deborah van Rooyen

Composer:

GEX Publishing Services

Copyeditor:

Lori Cavanaugh

Proofreader:

Green Pen Quality Assurance

Indexer:

Alexandra Nickerson

COPYRIGHT 2007 Thumbody's Thinking Inc.

ALL RIGHTS RESERVED. No part of this work may be reproduced, transcribed, or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, Web distribution, or information storage and retrieval systems—without prior written permission of the publisher.

An exception to the above is made for instructors and students. They are permitted to make copies at cost, including printed copies, for their own use.

Any additional questions about permissions can be submitted by e-mail to info@learningwithrobots.com

The Web addresses in this book are subject to change from time to time as necessary without notice.

Disclaimer

Thumbody's Thinking reserves the right to revise this publication and make changes from time to time in its content without notice.

For more information, contact Thumbody's Thinking, 211 Simeon Street, Kitchener, ON N2H 1S9 Canada or email info@learningwithrobots.com

This work was originally published by Thomson Course Technology, a division of Thomson Learning. The rights to this work have subsequently reverted back to the author's company, Thumbody's Thinking Inc. This copy, including the credits listed above, is identical except for information related to the original publisher.

ISBN 0-619-21724-3

Photo Credits

Figure 1-5: Courtesy of NASA/JPL-Caltech
Figure 1-22: Courtesy of the U.S. Navy
Figure 3-3: Cartoon © 2005 ScienceCartoonsPlus.com. Used with permission.
Cover: Drawing © 2001 by Joel Weber Becker. Used with permission.

Some portions of this work are based on *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming* by Joseph Bergin, Mark Stehlik, Jim Roberts, and Richard Pattis. Copyright © 1997 by John Wiley & Sons, Inc. Used with permission of John Wiley & Sons, Inc.

Contents

Chapter 1	Programming with Objects	1
1.1	Modeling with Objects	2
1.1.1	Using Models	2
1.1.2	Using Software Objects to Create Models	4
1.1.3	Modeling Robots	7
1.2	Understanding Karel's World	9
1.2.1	Avenues, Streets, and Intersections	9
1.2.2	Walls and (other) Things	10
1.2.3	Robots	10
1.3	Modeling Robots with Software Objects	12
1.3.1	Attributes	13
1.3.2	Constructors	13
1.3.3	Services	14
1.4	Two Example Programs	15
1.4.1	Situations	15
1.4.2	Program Listing	17
1.4.3	Setting up the Initial Situation	19
1.4.4	Sending Messages	20
1.4.5	Tracing a Program	20
1.4.6	Another Example Program	23
1.4.7	The Form of a Java Program	25
1.4.8	Reading Documentation to Learn More	26
1.5	Compiling and Executing Programs	29
1.5.1	Compile-Time Errors	30
1.5.2	Run-Time Errors	32
1.5.3	Intent Errors	33
1.5.4	A Brief History of Bugs and Debugging	34
1.6	GUI: Creating a Window	34
1.6.1	Displaying a Frame	35
1.6.2	Adding User Interface Components	37
1.7	Patterns	39
1.7.1	The Java Program Pattern	40
1.7.2	The Object Instantiation Pattern	41
1.7.3	The Command Invocation Pattern	42
1.7.4	The Sequential Execution Pattern	43
1.7.5	The Display a Frame Pattern	44

1.8	Summary and Concept Map	44
1.8.1	Concept Maps	45
1.9	Problem Set	46
Chapter 2	Extending Classes with Services	53
2.1	Understanding Programs: An Experiment	54
2.2	Extending the <code>Robot</code> Class	56
2.2.1	The Vocabulary of Extending Classes	58
2.2.2	The Form of an Extended Class	59
2.2.3	Implementing the Constructor	59
2.2.4	Adding a Service	62
2.2.5	Implementing <code>move3</code>	64
2.2.6	Implementing <code>turnRight</code>	65
2.2.7	<code>RobotSE</code>	66
2.2.8	Extension vs. Modification	67
2.3	Extending the <code>Thing</code> Class	67
2.3.1	Exploring the <code>Thing</code> Class	68
2.3.2	Implementing a Simple <code>Lamp</code> Object	69
2.3.3	Completely Initializing Lamps	74
2.3.4	Fine-Tuning the <code>Lamp</code> Class (optional)	75
2.3.5	Other Subclasses of <code>Thing</code>	76
2.3.6	Fun with Lights (optional)	77
2.4	Style	78
2.4.1	White Space and Indentation	78
2.4.2	Identifiers	79
2.4.3	Comments	81
2.4.4	External Documentation (advanced)	84
2.5	Meaning and Correctness	85
2.6	Modifying Inherited Methods	87
2.6.1	Overriding a Method Definition	87
2.6.2	Method Resolution	90
2.6.3	Side Effects	92
2.7	GUI: Extending GUI Components	92
2.7.1	Extending <code>JComponent</code>	94
2.7.2	Overriding <code>paintComponent</code>	97
2.7.3	How <code>paintComponent</code> Is Invoked	100
2.7.4	Extending <code>Icon</code>	100
2.8	Patterns	102
2.8.1	The Extended Class Pattern	102
2.8.2	The Constructor Pattern	103
2.8.3	The Parameterless Command Pattern	105
2.8.4	The Draw a Picture Pattern	105
2.9	Summary and Concept Map	106
2.10	Problem Set	107

Chapter 3	Developing Methods	115
3.1	Solving Problems	116
3.2	Stepwise Refinement	117
3.2.1	Identifying the Required Services	118
3.2.2	Refining <code>harvestField</code>	120
3.2.3	Refining <code>harvestTwoRows</code>	126
3.2.4	Refining <code>harvestOneRow</code>	127
3.2.5	Refining <code>goToNextRow</code>	129
3.2.6	Refining <code>positionForNextHarvest</code>	129
3.2.7	The Complete Program	130
3.2.8	Summary of Stepwise Refinement	132
3.3	Advantages of Stepwise Refinement	133
3.3.1	Understandable Programs	133
3.3.2	Avoiding Errors	134
3.3.3	Testing and Debugging	135
3.3.4	Future Modifications	136
3.4	Pseudocode	138
3.5	Variations on the Theme	139
3.5.1	Using Multiple Robots	140
3.5.2	Multiple Robots with Threads (advanced)	142
3.5.3	Factoring Out Differences	146
3.6	Private and Protected Methods	147
3.7	GUI: Using Helper Methods	151
3.7.1	Declaring Parameters	153
3.7.2	Using Parameters	153
3.8	Patterns	155
3.8.1	The Helper Method Pattern	155
3.8.2	The Multiple Threads Pattern	156
3.8.3	The Template Method Pattern	157
3.8.4	The Parameterized Method Pattern	158
3.9	Summary and Concept Map	159
3.10	Problem Set	160
Chapter 4	Making Decisions	167
4.1	Understanding Two Kinds of Decisions	168
4.1.1	Flowcharts for <code>if</code> and <code>while</code> Statements	169
4.1.2	Examining an <code>if</code> Statement	169
4.1.3	Examining a <code>while</code> Statement	171
4.1.4	The General Forms of the <code>if</code> and <code>while</code> Statements	173
4.2	Questions Robots Can Ask	174
4.2.1	Built-In Queries	174
4.2.2	Negating Predicates	175
4.2.3	Testing Integer Queries	176

4.3	Reexamining Harvesting a Field	177
4.3.1	Putting a Missing Thing	178
4.3.2	Picking Up a Pile of Things	179
4.3.3	Improving <code>goToNextRow</code>	181
4.4	Using the <code>if-else</code> Statement	183
4.4.1	An Example Using <code>if-else</code>	184
4.5	Writing Predicates	186
4.5.1	Writing <code>frontIsBlocked</code>	187
4.5.2	Predicates Using Non-Boolean Queries	188
4.6	Using Parameters	189
4.6.1	Using a <code>while</code> Statement with a Parameter	190
4.6.2	Using an Assignment Statement with a Loop	191
4.6.3	Revisiting Stepwise Refinement	193
4.7	GUI: Scaling Images	196
4.7.1	Using Size Queries	198
4.7.2	Scaling an Image	199
4.8	Patterns	201
4.8.1	The Once or Not at All Pattern	201
4.8.2	The Zero or More Times Pattern	202
4.8.3	The Either This or That Pattern	202
4.8.4	The Simple Predicate Pattern	203
4.8.5	The Count-Down Loop Pattern	204
4.8.6	The Scale an Image Pattern	205
4.9	Summary and Concept Map	205
4.10	Problem Set	207
Chapter 5	More Decision Making	211
5.1	Constructing <code>while</code> Loops	212
5.1.1	Avoiding Common Errors	212
5.1.2	A Four-Step Process for Constructing <code>while</code> Loops	214
5.2	Temporary Variables	218
5.2.1	Counting the <code>Things</code> on an Intersection	219
5.2.2	Tracing with a Temporary Variable	221
5.2.3	Storing the Result of a Query	222
5.2.4	Writing a Query	223
5.2.5	Using a <code>boolean</code> Temporary Variable	224
5.2.6	Scope	224
5.3	Nesting Statements	225
5.3.1	Examples Using <code>if</code> and <code>while</code>	225
5.3.2	Nesting with Helper Methods	227
5.3.3	Cascading- <code>if</code> Statements	227

5.4	Boolean Expressions	231
5.4.1	Combining Boolean Expressions	231
5.4.2	Simplifying Boolean Expressions	236
5.4.3	Short-Circuit Evaluation	238
5.5	Exploring Loop Variations	239
5.5.1	Using a <code>for</code> Statement	239
5.5.2	Using a <code>do-while</code> Loop (optional)	242
5.5.3	Using a <code>while-true</code> Loop (optional)	243
5.5.4	Choosing an Appropriate Looping Statement	246
5.6	Coding with Style	246
5.6.1	Use Stepwise Refinement	247
5.6.2	Use Positively Stated Simple Expressions	247
5.6.3	Visually Structure Code	250
5.7	GUI: Using Loops to Draw	251
5.7.1	Using the Loop Counter	252
5.7.2	Nesting Selection and Repetition	253
5.8	Patterns	257
5.8.1	The Loop-and-a-Half Pattern	257
5.8.2	The Temporary Variable Pattern	258
5.8.3	The Counting Pattern	259
5.8.4	The Query Pattern	259
5.8.5	The Predicate Pattern	260
5.8.6	The Cascading- <code>if</code> Pattern	261
5.8.7	The Counted Loop Pattern	262
5.9	Summary and Concept Map	262
5.10	Problem Set	264
 Chapter 6 Using Variables		 273
6.1	Instance Variables in the Robot Class	274
6.1.1	Implementing Attributes with Instance Variables	275
6.1.2	Declaring Instance Variables	276
6.1.3	Accessing Instance Variables	278
6.1.4	Modifying Instance Variables	280
6.1.5	Testing the <code>SimpleBot</code> Class	282
6.1.6	Adding Another Instance Variable: <code>direction</code>	283
6.1.7	Providing Accessor Methods	286
6.1.8	Instance Variables versus Parameter and Temporary Variables	289
6.2	Temporary and Parameter Variables	289
6.2.1	Reviewing Temporary Variables	290
6.2.2	Reviewing Parameter Variables	296

6.3	Extending a Class with Variables	300
6.3.1	Declaring and Initializing the Variables	302
6.3.2	Maintaining and Using Instance Variables	303
6.3.3	Blank Final Instance Variables	305
6.4	Modifying vs. Extending Classes	305
6.5	Comparing Kinds of Variables	306
6.5.1	Similarities and Differences	306
6.5.2	Rules of Thumb for Selecting a Variable	307
6.5.3	Temporary versus Instance Variables	308
6.6	Printing Expressions	310
6.6.1	Using <code>System.out</code>	310
6.6.2	Using a Debugger	311
6.7	GUI: Repainting	312
6.7.1	Instance Variables in Components	314
6.7.2	Triggering a Repaint	317
6.7.3	Animating the <code>Thermometer</code>	317
6.8	Patterns	318
6.8.1	The Named Constant Pattern	318
6.8.2	The Instance Variable Pattern	319
6.8.3	The Accessor Method Pattern	320
6.9	Summary and Concept Map	321
6.10	Problem Set	322
Chapter 7	More on Variables and Methods	329
7.1	Using Queries to Test Classes	330
7.1.1	Testing a Command	330
7.1.2	Testing a Query	332
7.1.3	Using Multiple Main Methods	335
7.2	Using Numeric Types	337
7.2.1	Integer Types	337
7.2.2	Floating-Point Types	338
7.2.3	Converting Between Numeric Types	340
7.2.4	Formatting Numbers	341
7.2.5	Taking Advantage of Shortcuts	344
7.3	Using Non-Numeric Types	344
7.3.1	The <code>boolean</code> Type	344
7.3.2	The Character Type	345
7.3.3	Using Strings	347
7.3.4	Understanding Enumerations	355
7.4	Example: Writing a Gas Pump Class	358
7.4.1	Implementing Accessor Methods	359
7.4.2	Implementing a Command/Query Pair	361
7.5	Understanding Class Variables and Methods	366
7.5.1	Using Class Variables	366
7.5.2	Using Class Methods	368

7.6	GUI: Using Java Interfaces	374
7.6.1	Specifying Methods with Interfaces	375
7.6.2	Implementing an Interface	377
7.6.3	Developing Classes to a Specified Interface	378
7.6.4	Informing the User Interface of Changes	379
7.7	Patterns	381
7.7.1	The Test Harness Pattern	381
7.7.2	The <code>toString</code> Pattern	381
7.7.3	The Enumeration Pattern	382
7.7.4	The Assign a Unique ID Pattern	383
7.8	Summary and Concept Map	384
7.9	Problem Set	386
Chapter 8	Collaborative Classes	391
8.1	Example: Modeling a Person	392
8.1.1	Using a Single Class	392
8.1.2	Using Multiple Classes	394
8.1.3	Diagramming Collaborating Classes	399
8.1.4	Passing Arguments	401
8.1.5	Temporary Variables	401
8.1.6	Returning Object References	401
8.1.7	Section Summary	403
8.2	Reference Variables	403
8.2.1	Memory	404
8.2.2	Aliases	406
8.2.3	Garbage Collection	409
8.2.4	Testing for Equality	410
8.3	Case Study: An Alarm Clock	412
8.3.1	Step 1: Identifying Objects and Classes	412
8.3.2	Step 2: Identifying Services	414
8.3.3	Step 3: Solving the Problem	423
8.4	Introducing Exceptions	424
8.4.1	Throwing Exceptions	424
8.4.2	Reading a Stack Trace	425
8.4.3	Handling Exceptions	426
8.4.4	Propogating Exceptions	428
8.4.5	Enhancing the Alarm Clock with Sound (optional)	429
8.5	Java's Collection Classes	431
8.5.1	A List Class: <code>ArrayList</code>	432
8.5.2	A Set Class: <code>HashSet</code>	439
8.5.3	A Map Class: <code>TreeMap</code>	441
8.5.4	Wrapper Classes	446

8.6	GUIs and Collaborating Classes	447
8.6.1	Using Libraries of Components	448
8.6.2	Introducing the Model-View-Controller Pattern	448
8.7	Patterns	449
8.7.1	The Has-a (Composition) Pattern	449
8.7.2	The Equivalence Test Pattern	450
8.7.3	The Throw an Exception Pattern	451
8.7.4	The Catch an Exception Pattern	452
8.7.5	The Process All Elements Pattern	452
8.8	Summary and Concept Map	453
8.9	Problem Set	454

Chapter 9 Input and Output 459

9.1	Basic File Input and Output	460
9.1.1	Reading from a File	461
9.1.2	Writing to a File	464
9.1.3	The Structure of Files	466
9.2	Representing Records as Objects	472
9.2.1	Reading Records as Objects	472
9.2.2	File Formats	475
9.3	Using the File Class	477
9.3.1	Filenames	477
9.3.2	Specifying File Locations	478
9.3.3	Manipulating Files	479
9.4	Interacting with Users	480
9.4.1	Reading from the Console	480
9.4.2	Checking Input for Errors	481
9.5	Command Interpreters	486
9.5.1	Using a Command Interpreter	486
9.5.2	Implementing a Command Interpreter	487
9.5.3	Separating the User Interface from the Model	492
9.6	Constructing a Library	495
9.6.1	Compiling without an IDE	495
9.6.2	Creating and Using a Package	497
9.6.3	Using <code>.jar</code> Files	499
9.7	Streams (advanced)	500
9.7.1	Character Input Streams	501
9.7.2	Character Output Streams	502
9.7.3	Byte Streams	503
9.8	GUI: File Choosers and Images	503
9.8.1	Using <code>JFileChooser</code>	504
9.8.2	Displaying Images from a File	506

9.9	Patterns	508
9.9.1	The Open File for Input Pattern	508
9.9.2	The Open File for Output Pattern	508
9.9.3	The Process File Pattern	508
9.9.4	The Construct Record from File Pattern	509
9.9.5	The Error-Checked Input Pattern	509
9.9.6	The Command Interpreter Pattern	510
9.10	Summary and Concept Map	511
9.11	Problem Set	512
<hr/>		
Chapter 10	Arrays	519
10.1	Using Arrays	520
10.1.1	Visualizing an Array	521
10.1.2	Accessing One Array Element	522
10.1.3	Swapping Array Elements	525
10.1.4	Processing All the Elements in an Array	527
10.1.5	Processing Matching Elements	528
10.1.6	Searching for a Specified Element	529
10.1.7	Finding an Extreme Element	533
10.1.8	Sorting an Array	534
10.1.9	Comparing Arrays and Files	540
10.2	Creating an Array	541
10.2.1	Declaration	542
10.2.2	Allocation	543
10.2.3	Initialization	544
10.3	Passing and Returning Arrays	547
10.4	Dynamic Arrays	551
10.4.1	Partially Filled Arrays	551
10.4.2	Resizing Arrays	554
10.4.3	Combining Approaches	557
10.5	Arrays of Primitive Types	558
10.5.1	Using an Array of <code>double</code>	558
10.5.2	Meaningful Indices	560
10.6	Multi-Dimensional Arrays	562
10.6.1	2D Array Algorithms	563
10.6.2	Allocating and Initializing a 2D Array	565
10.6.3	Arrays of Arrays	566
10.7	GUI: Animation	569
10.8	Patterns	572
10.8.1	The Process All Elements Pattern	573
10.8.2	The Linear Search Pattern	573
10.9	Summary and Concept Map	574
10.10	Problem Set	575

Chapter 11	Building Quality Software	583
11.1	Defining Quality Software	584
11.1.1	Quality from a User's Perspective	584
11.1.2	Quality from a Programmer's Perspective	585
11.2	Using a Development Process	586
11.2.1	Defining Requirements	587
11.2.2	Designing the Architecture	588
11.2.3	Iterative Development	595
11.3	Designing Classes and Methods	599
11.3.1	Rules of Thumb for Writing Quality Code	606
11.3.2	Managing Complexity	615
11.4	Programming Defensively	621
11.4.1	Exceptions	621
11.4.2	Design by Contract	622
11.4.3	Assertions	624
11.5	GUIs: Quality Interfaces	624
11.5.1	Iterative User Interface Design	625
11.5.2	User Interface Design Principles	626
11.6	Summary and Concept Map	628
11.7	Problem Set	629
Chapter 12	Polymorphism	633
12.1	Introduction to Polymorphism	634
12.1.1	Dancing Robots	634
12.1.2	Polymorphism via Inheritance	635
12.1.3	Examples of Polymorphism	640
12.1.4	Polymorphism via Interfaces	643
12.1.5	The Substitution Principle	645
12.1.6	Choosing between Interfaces and Inheritance	646
12.2	Case Study: Invoices	647
12.2.1	Step 1: Identifying Objects and Classes	648
12.2.2	Step 2: Identifying Services	652
12.2.3	Step 3: Solving the Problem	655
12.3	Polymorphism without Arrays	661
12.4	Overriding Methods in <code>object</code>	662
12.4.1	<code>toString</code>	662
12.4.2	<code>equals</code>	663
12.4.3	<code>clone</code> (advanced)	665
12.5	Increasing Flexibility with Interfaces	669
12.5.1	Using an Interface	671
12.5.2	Using the Strategy Pattern	674
12.5.3	Flexibility in Choosing Implementations	679

12.6	GUI: Layout Managers	680
12.6.1	The <code>FlowLayout</code> Strategy	680
12.6.2	The <code>GridLayout</code> Strategy	681
12.6.3	The <code>BorderLayout</code> Strategy	683
12.6.4	Other Layout Strategies	683
12.6.5	Nesting Layout Strategies	684
12.7	Patterns	686
12.7.1	The Polymorphic Call Pattern	686
12.7.2	The Strategy Pattern	687
12.7.3	The Equals Pattern	688
12.7.4	The Factory Method Pattern	689
12.8	Summary and Concept Map	689
12.9	Problem Set	690
Chapter 13 Graphical User Interfaces		697
13.1	Overview	698
13.1.1	Models, Views, and Controllers	698
13.1.2	Using a Pattern	700
13.2	Setting up the Model and View	700
13.2.1	The Model's Infrastructure	701
13.2.2	The View's Infrastructure	703
13.2.3	The <code>main</code> Method	704
13.3	Building and Testing the Model	705
13.4	Building the View and Controllers	709
13.4.1	Designing the Interface	709
13.4.2	Laying Out the Components	711
13.4.3	Updating the View	713
13.4.4	Writing and Registering Controllers	716
13.4.5	Refining the View	721
13.4.6	View Pattern	725
13.5	Using Multiple Views	726
13.5.1	Implementing <code>NimView</code>	728
13.5.2	Implementing <code>NimPileView</code>	729
13.5.3	Implementing <code>NimPlayerView</code>	730
13.5.4	Sequence Diagrams	733
13.6	Controller Variations	735
13.6.1	Using Inner Classes	735
13.6.2	Using Event Objects	737
13.6.3	Integrating the Controller and View	739
13.7	Other Components	740
13.7.1	Discover Available Components	740
13.7.2	Identify Listeners	741
13.7.3	Skim the Documentation	743
13.7.4	Begin with Sample Code	744
13.7.5	Work Incrementally	744

13.8	Graphical Views	747
13.8.1	Painting a Component	747
13.8.2	Making a Graphical Component Interactive	750
13.9	Patterns	756
13.9.1	The Model-View-Controller Pattern	756
13.10	Summary and Concept Map	757
13.11	Problem Set	758
<hr/> <u>Epilogue</u>		<u>765</u>
<hr/> <u>Appendix A Glossary</u>		<u>771</u>
<hr/> <u>Appendix B Precedence Rules</u>		<u>787</u>
<hr/> <u>Appendix C Variable Initialization Rules</u>		<u>791</u>
<hr/> <u>Appendix D Unicode Character Set</u>		<u>793</u>
<hr/> <u>Appendix E Selected Robot Documentation</u>		<u>797</u>
<hr/> <u>Index</u>		<u>815</u>

Preface

The preface includes:

- Why this book exists
- The approach it takes to teaching object-oriented programming
- The advantages of this approach
- A section for students describing the software they need and the features of this book that they will find particularly helpful
- A section for instructors describing the author's *Use, Then Write* object-oriented pedagogy, the organization and coverage of topics, and supplemental resources
- Who helped the author along the way

How It All Started

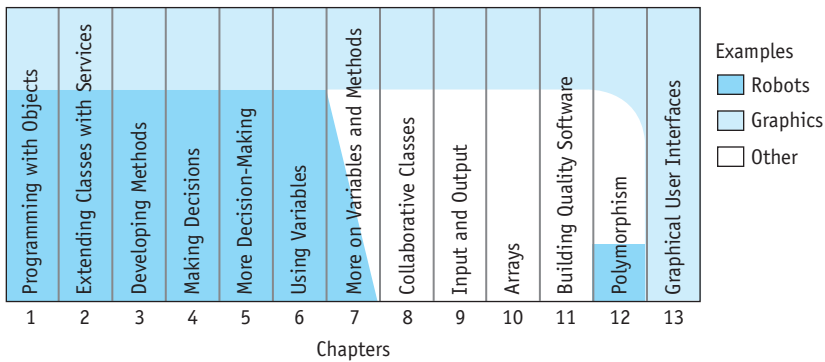
As often happens, this book exists because the author was unhappy with the alternatives. When I was first asked to develop a Java version of our introductory programming course for 1,000 students a year, I naturally collected all the relevant Java textbooks I could find. They all left me with a vague sense of uneasiness. Yes, the programming language had changed from Pascal to Java, but the approach had not. A second change was necessary: a change in pedagogy.

The first term of my course did not go well. I had chosen what I considered to be the best textbook available, but the experience of teaching with it only confirmed that the pedagogical paradigm shift had not been made. Shortly thereafter I discovered a small book, *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming* (Wiley, 1997). It was an “Aha!” experience for me. The pedagogy of this book felt right to me. In addition, I knew its metaphor of programming robots would appeal to my students, it had an obvious appeal for visual learners, and I could imagine having lots of fun acting out programs with students. Unfortunately, *Karel++* is a C++ textbook, not Java. Furthermore, at only 175 pages and lacking many language-specific details, it forms the first several weeks of an introductory course. After that, a different textbook is required—a textbook that did not exist.

Discussions with the publisher of *Karel++* led to them granting me permission to translate it to Java for use at the University of Waterloo, Ontario, Canada. After experiencing the joys of teaching with the approach—and the difficulties of changing to an unrelated text after a few weeks—I began to write the textbook I really wanted. *Java: Learning to Program with Robots* combines the wonderful pedagogy of *Karel++* with the full and complete treatment required by an introductory object-oriented programming textbook.

Approach

This text begins with programming virtual robots to teach object-oriented programming in general (dark green in Figure 1). Once students are comfortable with many aspects of objects and classes, the examples shift from robots to a much broader set of examples (white). Each chapter ends with a section on graphics and graphical user interfaces (light green), applying the concepts learned to a different context. Transferring the knowledge gained using robots to another problem (graphics) is an important part of mastering the material. The graphics sections at the end of each chapter should be viewed as an integral part of the curriculum.

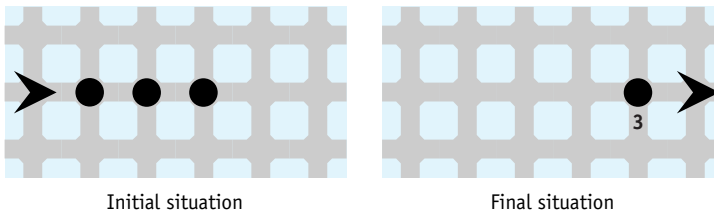


(figure 1)

Distribution of example programs

Starting with Robots

Robots are objects in an object-oriented program that can receive messages telling them to move, turn, pick things up, carry things, and put things down again. We all have a mental image of robots and can easily direct them to perform a task, such as picking up three things in a row and putting them in a pile. This task can be clarified with a pair of diagrams, as shown in Figure 2. The first diagram shows how the task begins: with the robot (an arrowhead) and three things (circles) in front of it. The second diagram shows how the task should end: with the three things all in the same place.



(figure 2)

Robot picking up three things and putting them in a pile

We can easily “program” a student or instructor to complete this task with the following instructions. Assume the person’s name is “Karl.”


```
Karl, move  
Karl, pick up a thing  
Karl, move  
Karl, pick up a thing  
Karl, move  
Karl, pick up a thing  
Karl, move  
Karl, put down a thing  
Karl, put down a thing  
Karl, put down a thing  
Karl, move
```

After verbally directing Karl, it is easy to introduce a simple program that does the same thing where `karl` is the name of a robot object, as follows:

```
karl.move();  
karl.pickThing();  
karl.move();  
karl.pickThing();  
karl.move();  
karl.pickThing();  
karl.move();  
karl.putThing();  
karl.putThing();  
karl.putThing();  
karl.move();
```

There are additional details to cover before this Java fragment can be executed as a complete program. However, these details form an easily learned pattern, leaving the focus on using robot objects to accomplish tasks.

Other kinds of objects can be included in robot programs, including walls that can block a robot from moving and lights that can be turned on and off. We can also create new kinds of objects to use.

The fundamental object-oriented concepts learned with robot objects can all be transferred to programs that have nothing to do with robots. Each chapter includes a section focusing on graphics to help with the conceptual transfer. The latter part of the book includes many examples that have nothing to do with robots.

Advantages of Using Robots

Using robots to learn object-oriented programming offers significant advantages. I have used this approach in my classes for half a dozen years, and find that the following qualities are the most important advantages.

Visualization: The visual qualities of robots make it easy to specify a problem using pictures and a few lines of text. They provide visual feedback about the correctness of the program. Watching the robot traverse the screen makes debugging easier. This text makes the most of the human brain's highly optimized processing of visual input.

Ease of Programming: Object-oriented programs are easier to write when programmers can imagine what they would do if they were the objects in the program. Robot objects make this easy. Because moving, turning, picking things up, and putting them down again are activities that we do every day, it is easy for us to give directions to one another or to a robot object. Even though this method is easier to grasp, we still learn important object-oriented programming concepts.

Fun: Robots are fun! I have never had so much fun with a classroom of students as the day we worked with a “paranoid” robot that “looked” to the right and to the left before it moved forward. People who acted it out adopted a hunched, uptight look with shifty eyes that generated much laughter among the students. Later in the same period, we turned this into a paranoid thief that went up the aisle swiping small objects from student desks, all the while looking both ways before it would move. It was fun, but it also taught students about inheritance, one of the three hallmarks of object-oriented programming.

Quick Startup: The robot microworld allows students to begin object-oriented programming immediately using real objects in a real programming environment. Similar approaches often use graphics alone, but robots are more intuitive than graphics and have many more interesting algorithmic aspects.

Pedagogy: Finally, I believe that the largest benefit of using robots is that they lend themselves to a superior pedagogy for teaching object-oriented programming. This ultimate benefit is more fully explained in a later section of this Preface, For Instructors.

For Students

You are about to embark on an exciting journey of learning to program using Java. Before we begin, let’s take a few moments to orient ourselves to this textbook and to the software you will need to complete all the exercises in the book.

Textbook Features

This textbook includes a number of features to make your life as a student easier. They include the following:

Objectives: A brief list of objectives appears at the beginning of each chapter to provide an overview of the chapter contents. Knowing your destination helps you make the most of your journey through the chapter.

Program listings: Each chapter contains many examples of working code demonstrating the principles under discussion. The code is often shown as a complete listing that is available for you to download, modify, and run yourself.

Figures: Each chapter provides a rich collection of figures to help illustrate the concepts. Figures include UML diagrams, illustrations of robot programs, flowcharts, screen shots of program output, and many others illustrating program features, object-oriented concepts, and the principles of effective program design.

Key terms and glossary: Every discipline has its own vocabulary, including computer science. When a term is used for the first time, it's highlighted. A complete glossary in Appendix A is a handy reference for those times that you need a reminder.

Margin notes: The margin of each chapter contains four types of notes. *Find the Code* notes direct you to files containing sample code. *Key Idea* notes summarize key ideas discussed on the page and help you review. *Looking Back* notes link current discussions with ideas covered earlier in the book. *Looking Ahead* notes preview concepts or techniques introduced in later chapters.

Pattern icons and discussion: In addition to margin notes, each chapter includes pattern icons to highlight code or to explain common programming patterns. Learning to recognize these patterns is an important part of becoming a good programmer. A section named “Patterns” near the end of each chapter summarizes the patterns and generalizes them so that they're more broadly applicable.

Graphical user interface sections: Each chapter includes a section presenting the chapter's topics in the context of graphical user interfaces, helping you transfer your understanding to new situations. In addition, many of the problems in each chapter have a graphical user interface to make your homework look more like the programs you use every day. In the early chapters, the interface is provided by the robot world. In the middle chapters, graphical user interfaces are often provided to work with the code you write. In the last chapter, you will write the interfaces yourself.

Concept maps and summaries: Each chapter concludes with a brief written summary of the important concepts, followed by a concept map. The concept map gives a visual representation of the ideas discussed and how they are related to each other.

Obtaining and Installing Software

Writing programs requires tools. A minimal set of tools is a text editor and the Java Development Kit (JDK) from Sun Microsystems. The JDK is included in the CD-ROM that accompanies this textbook. Updates can be downloaded from www.java.sun.com/j2se/. The software you will be using with this textbook requires Java 5 or higher (also known as JDK 1.5).

Another approach is to use an Integrated Development Environment (IDE). It integrates the text editor and development tools such as the JDK into one environment that is optimized specifically for writing programs. The CD-ROM includes two such IDEs, JCreator and jGrasp. Others include Dr. Java (www.drjava.org/), BlueJ (www.bluej.org/), and Eclipse (www.eclipse.org/). Of these, JCreator and Eclipse are aimed at programmers; the others are developed specifically for students. All of the IDEs listed here have a free version.

In addition to the JDK or an IDE, the introductory programs in this textbook require software implementing the robots. This software and documentation is available on the Robots Web site, www.learningwithrobots.com, and on the CD-ROM.

Instructions for installing the software and documentation is available on the CD-ROM (open `InstallationInstructions.html` with your Web browser) and on the Robots Web site (www.learningwithrobots.com/InstallationInstructions.html).

For Instructors

Robots uses objects to their fullest extent from day one, but doesn't overwhelm the students. How? It provides a rich set of classes that students use to learn about objects *before* they are asked to write their own classes. Let's explore this *Use, Then Write* pedagogy further by comparing it with the alternatives.

Object-Oriented Pedagogies

The concepts of object and class are intimately related. Each kind of object in a student's program is created from a class that a programmer writes to define the objects' characteristics. Given that students need to master both using objects and writing the classes that define them, a crucial question is how to order these topics. There are three possibilities for *writing* classes and *using* the resulting objects:

Write and use: In this approach students are asked to master the basics of writing a class at the same time they are learning how to use objects. One author, for example, introduces classes and objects by describing how to use a bank account object in only two pages. The author then delves into the details of writing the class to define it. This requires introducing students to the distinction between class and object, declaring objects, object instantiation, invoking methods, the structure of a class, defining methods, declaring parameters and passing arguments, return values, and instance variables. This presents an incredible cognitive load for students. The author chose a wonderful example to convey all these concepts, but it is still difficult to understand all the concepts all at once, even at an introductory level.

Write, then use: When actually writing a program, programmers first write the required classes and then use the objects they define. I am aware of only one textbook that has chosen to follow this same ordering. It includes a light treatment on the idea of an object, but then delves into the details of writing classes with very few examples of how the objects they define would be used. This lessens the cognitive load on the students by focusing on just one of the two aspects, but leaves students wondering how these classes are used. Much of the instruction on writing classes is lost because students don't have practical experience in using the resulting objects.

Use, then write: A third possibility is to first use objects and then learn how to write classes defining new kinds of objects. *Robots* uses this approach. Students make extensive use of robot objects, learning how to declare objects, instantiate objects, and invoke their methods. All the details of writing their own classes come later, after they are comfortable with using objects.

Robots provides a gentle but thorough introduction to object-oriented programming using the *Use, Then Write* pedagogy. It's an approach that helps students write interesting, object-oriented programs right away. It uses objects early and consistently, even with the traditional subjects of selection and repetition. Furthermore, it has been classroom tested with over 6,000 students at the University of Waterloo.

Organization and Coverage

Chapter 1, “Programming with Objects,” introduces students to instantiating and using objects.

Chapter 2, “Extending Classes with Services,” discusses extending an existing class with new parameterless methods.

Chapter 3, “Developing Methods,” continues the theme of writing methods, but with a focus on strategies for writing complex methods—pseudocode and stepwise refinement.

Chapter 4, “Making Decisions,” explores how to alter a program’s flow with repetition and selection, and includes the basics of the Boolean expressions used in such constructs. Introducing parameters adds even more flexibility to the methods students write.

Chapter 5, “More Decision Making,” continues exploring decision-making constructs with a process for writing correct loops, additional control statements, and manipulating Boolean expressions. Temporary (local) variables are introduced to simplify some algorithms.

Chapter 6, “Using Variables,” introduces integer instance variables and constants, and expands on using temporary variables and parameter variables.

Chapter 7, “More on Variables and Methods,” examines using variables with types other than `int`, including strings. Queries are used to examine the state of an object and to test it using a test harness. This chapter also includes the first large case study that does not involve robots or graphics.

Chapter 8, “Collaborative Classes,” presents classes that use references to another class and thoroughly explores the differences between reference types and primitive types. Exceptions are introduced, as well as Java collections to collaborate with many objects.

Chapter 9, “Input and Output,” covers reading information from files, writing information to files, and interacting with users via the console.

Chapter 10, “Arrays,” explains how to work with arrays. A number of algorithms are discussed, including a careful treatment of Selection Sort. Handling changing numbers of elements and multi-dimensional arrays are also covered.

Chapter 11, “Building Quality Software,” identifies characteristics of quality software and explains how to follow a development process that promotes quality.

Chapter 12, “Polymorphism,” explores writing polymorphic programs using inheritance and interfaces. It also discusses building an inheritance hierarchy and using the strategy and factory method patterns to make programs more flexible.

Chapter 13, “Graphical User Interfaces,” examines how to write a graphical user interface using existing Java components, structure a graphical user interface using the model-view-controller pattern and multiple views, and write new components for use in graphical user interfaces.

Dependencies

This text is, of necessity, printed in a particular order. You may find that a different organization suits you and your students better. The dependency chart shown in Figure 3 serves as a guide to reordering the material. The core material is shown with heavy lines and should be presented in the order shown. Other material can be rearranged around it at your discretion.

Textbook Features

Most of the textbook's features are listed in the section for students. Three features that instructors are more likely than students to appreciate are listed here:

Written exercises: The problem set at the end of each chapter includes written exercises, which provide an opportunity for students to synthesize the ideas and techniques they have learned in the chapter.

Programming exercises: The problem sets also include programming exercises, which prompt students to write, improve, or experiment with smaller programs.

Programming projects: Finally, the problem sets present projects that encourage students to create complete classes or programs.

Supplemental Resources

The following ancillary materials are available when this book is used in a classroom setting. All of the teaching tools available with this book are provided to the instructor on a single CD.

Instructor's Manual: Additional instructional material to assist in class preparation, including suggested syllabi for 14 and 16 week courses, and complete lecture notes.

PowerPoint Presentations®: This book comes with Microsoft PowerPoint slides for each chapter. In addition to reviewing the chapter, they contain examples and case studies illustrating the current topics. The slides are included as a teaching aid for classroom presentation, to make available to students on the network for chapter review, or to be printed for classroom distribution. Instructors can add their own slides for additional topics they may introduce to the class.

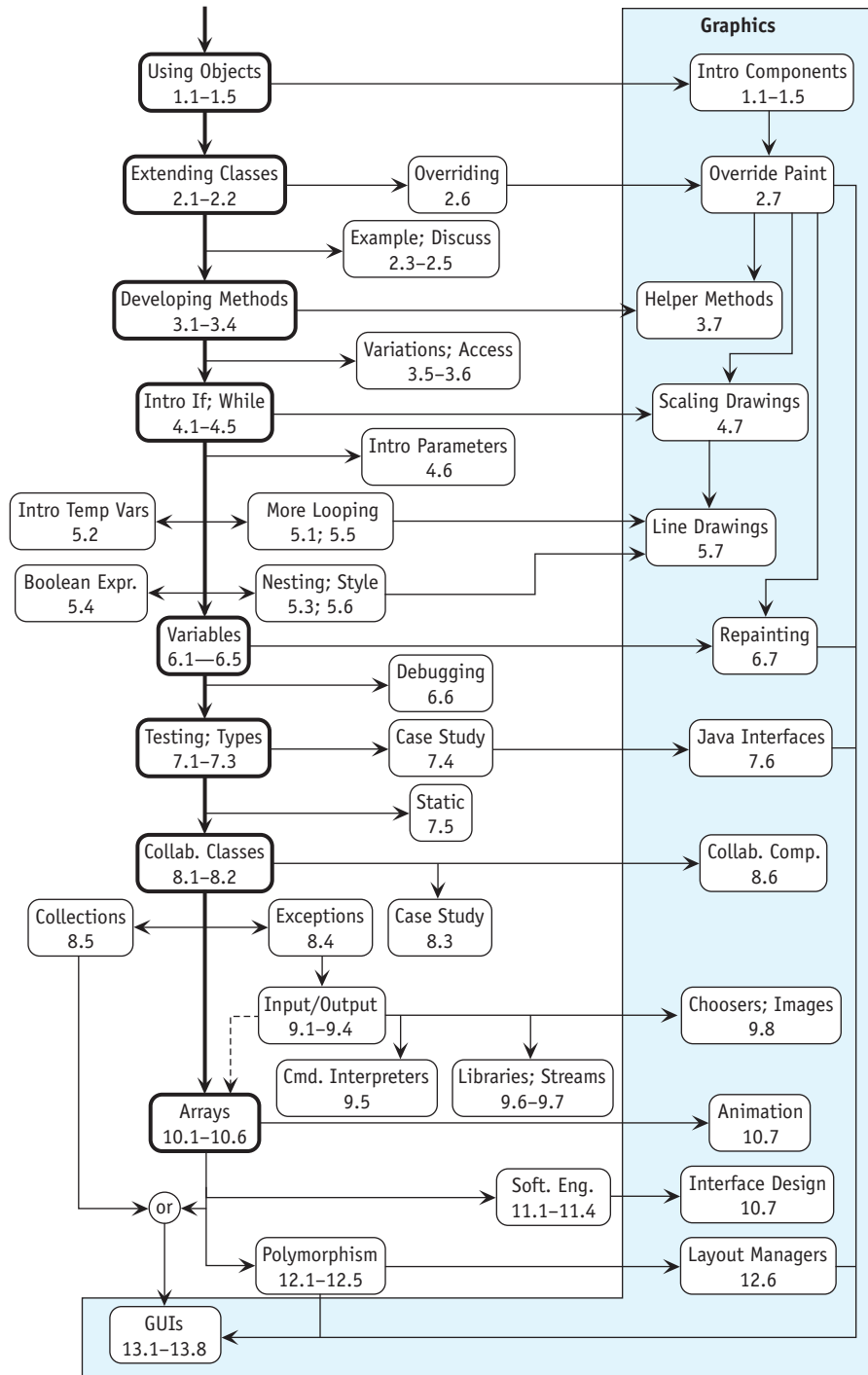
Solution Files: Sample solutions to most exercises.

Example Programs: The source code to almost all of the Java programs listed in this book are easily available to you and your students. They are on the CD accompanying each copy of the book, the Instructor Resources CD, the book's Web site (www.learningwithrobots.com), and the Thomson Course Technology Web site.

ExamView Test Bank: This assessment tool can help instructors design and administer tests.

Software: JDK 5.0, jGRASP, and JCreator are included with each copy of this book. Also provided are the libraries containing the robot classes. These libraries work with any Java development environment (JDK 5.0 and above) and permit you to write, run, and animate robot programs. Because a regular development environment is used, students do

(figure 3)
Dependency chart



not experience a transition in technology from writing robot programs to any other kind of program. Complete graphical user interfaces are also provided in the supporting libraries for use in a number of homework problems.

Web site: *www.learningwithrobots.com* makes many of these resources available to you and your students wherever you have an Internet connection.

Acknowledgements

In recalling those who have helped this book become a reality, I think of five groups of people.

Originators: Rich Pattis developed the idea of using robots to teach programming in the early 1980s. The idea was later adapted to an object-oriented style by Joe Bergin. These are the giants upon whose shoulders this work stands. Without them, this text and the core ideas it builds on would not exist. Thank you to Rich, in particular, who has been very encouraging of my attempts to adapt his ideas to a full CS1 textbook.

Facilitators: Bruce Spatz, Bill Zobrist, Paul Crockett, and all of John Wiley and Sons were flexible with their intellectual property rights to the original Karel the Robot book. Thank you.

Brainstormers: Jack Rehder, Judene Pretti, and Arnie Dyck are all wonderful colleagues of mine at the University of Waterloo. Much of the text has been shaped and improved by brainstorming sessions with them in the course of teaching this material together. Thank you for the ideas, the clarifications, and the suggestions. A large group of other instructors and tutors also contributed in countless smaller ways.

Polishers: Many people helped put the finishing touches on this book to get it ready for publication. They include the team at Course Technology: Lisa Ruffolo, Alyssa Pratt, Kelly Robinson, Mary Franz, and Mac Mendelsohn. Thank you for all your hard work and willingness to listen to my views on the design. Carrie Howells, a colleague at University of Waterloo, did a wonderful job of proofreading and critiquing many chapters. Michael Diramio, one of our former tutors, rescued my sanity by writing some of the solutions to problem sets. Finally, a huge thank you to the reviewers: John Ridgeway (Wesleyan University), Mary Goodwin (Illinois State University), Noel LeJeune (Metropolitan State College of Denver), and especially Rich Pattis (Carnegie Mellon University). Their insightful comments caused me to rework many sections that I had thought were finished.

Cheerleaders: My two sons, Luke and Joel, who can hardly wait to learn to program with “Dad’s robots,” cheered me on. Joel’s artwork graces the cover. A colleague, Sandy Graham, was a wonderful evangelist for the approach.

Finally, the biggest thank you is to Ann, the most wonderful woman a man could ever marry, for her indulgence as I wrote.

—Byron Weber Becker