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

Compositor:
GEX Publishing Services

Copyeditor:
Lori Cavanaugh

Proofreader:
Green Pen Quality Assurance

Indexer:
Alexandra Nickerson

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

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

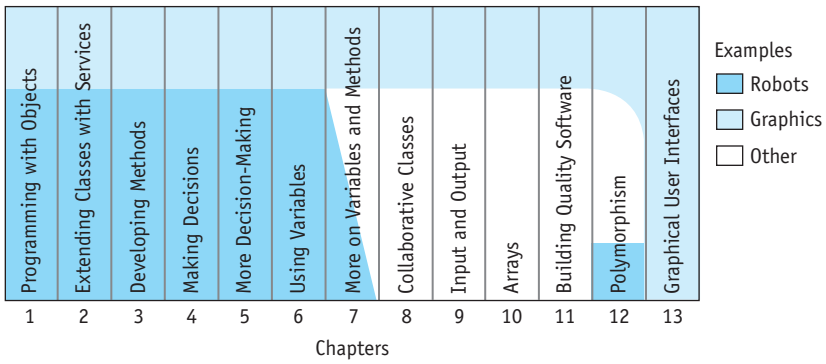
As often happens, this book exists because the author was unhappy with the alternatives. When I was first asked to develop a Java version of our introductory programming course for 1,000 students a year, I naturally collected all the relevant Java textbooks I could find. They all left me with a vague 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.

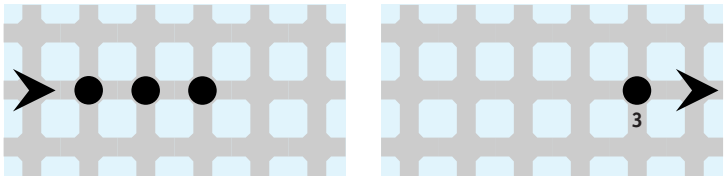


(figure 1)

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.

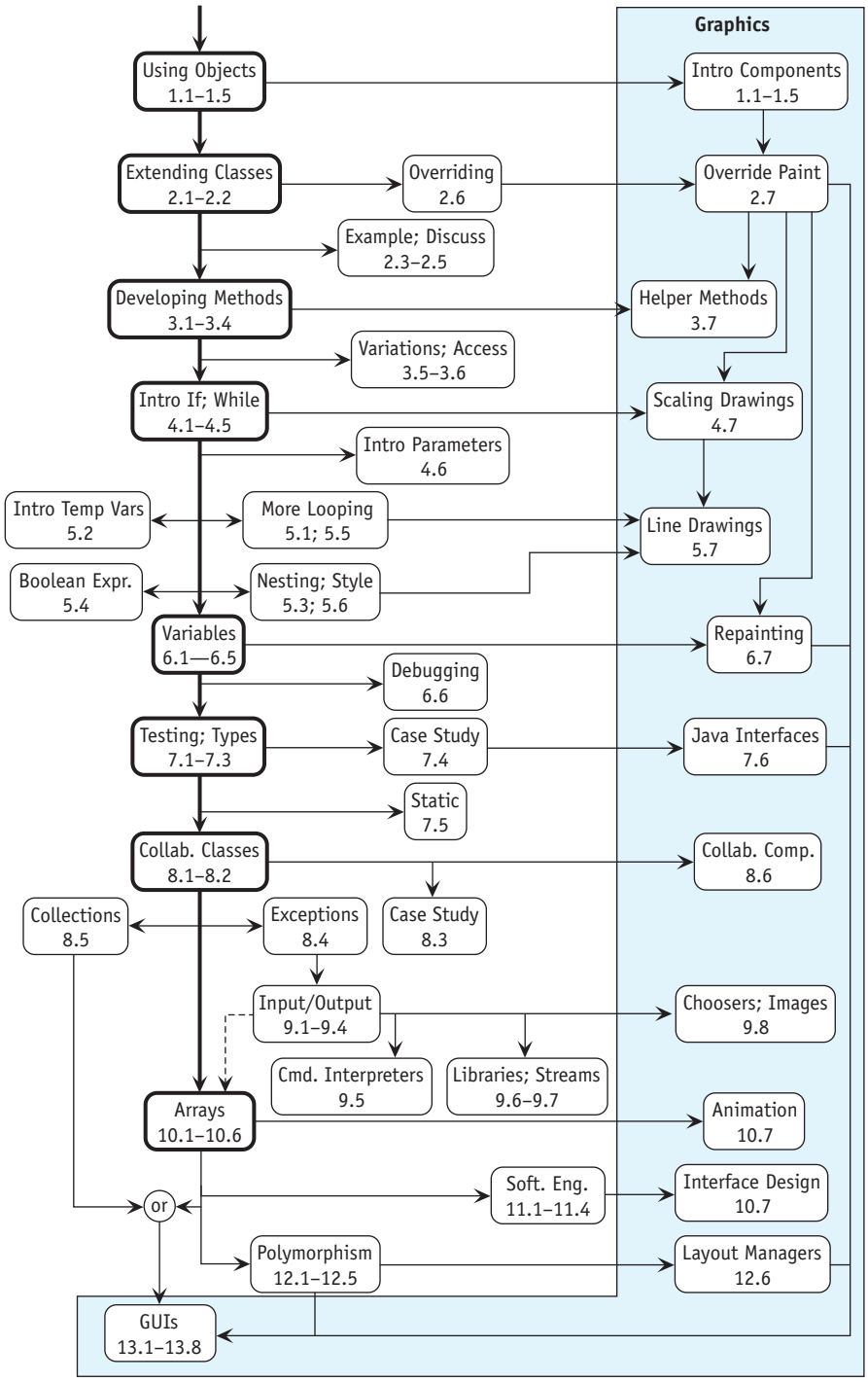
Example Programs: The source code to almost all of the Java programs listed in this book are easily available to you and your students. They are on the CD accompanying each copy of the book, the Instructor Resources CD, the book's Web site (*www.learningwithrobots.com*), and the Thomson Course Technology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