Teaching CS1 with Karel the Robot in Java

Byron Weber Becker
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
bwbecker@uwaterloo.ca

Abstract

Most current Java textbooks for CS1 (and thus most current courses) begin either with fundamentals from the procedural paradigm (assignment, iteration, selection) or with a brief introduction to using objects followed quickly with writing objects. We have found a third way to be most satisfying for both teachers and students: using interesting predefined classes to introduce the fundamentals of object-oriented programming (object instantiation, method calls, inheritance) followed quickly by the traditional fundamentals of iteration and selection, also taught using the same predefined classes.

Karel the Robot, developed by Richard Pattis [6] and well-known to many computer science educators, has aged gracefully and is a vital part of our CS1 curriculum. This paper explains how Karel may be used and the advantages of doing so.

1 Introduction

We began the development of our Java-based introductory programming course for CS majors and Math students in 1998. At the outset we realized the obvious: we would have to make a paradigm shift from procedural programming to object-oriented programming. What surprised us was the second paradigm shift required – the paradigm for *teaching* an OO language (Java) is different from teaching a procedural language such as Pascal, C or even C++.

The first time we taught the course we used a textbook which was very well-written but placed objects relatively late – about where a Pascal text would place records. We were uneasy with this, feeling that objects really ought to be introduced and used from day 1, but as we could not find a satisfactory text which supported this we made an attempt to provide our own examples parallel to the text.

It was a disaster. The fundamental object-oriented concepts (object instantiation, method calls, inheritance) came too late in the course. Students didn't have enough time to master the topics. This was worsened by the fact that most of our students come with some (procedural) programming experience from high school. When the early part of the course focused on iteration and selection (without objects, except for the little we provided ourselves), most students thought they already knew it all.

After reviewing textbooks again, our standard rant went like this: "Textbooks start with a vague, airy-fairy descriptions of objects (Java *is* an object-oriented language after all), but then go into primitive types. *No objects.* Chapter 3 is selection using payroll as an example. *No objects*. Chapter 4 is iteration – *no objects*. Suddenly, in chapter 5, students are expected to use and write objects. And they say, 'Remind me what an object is… What are they good for? How do I use them? What are the characteristics of a good object?'."

Shortly thereafter we discovered Karel the Robot in *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming* [2], the most recent incarnation of Rich Pattis' idea from the early 1980's. It was a revelation to us. *This* was how we wanted to teach Java. There were just two problems. First, the book is really oriented towards C++ rather than Java. Second, it's only the introduction to an introductory course, covering the first 4-5 weeks.

After convincing the publisher to allow us to translate *Karel++* to Java, adding a second textbook, writing software to support Karel, and rewriting all our lectures, we have a course we are very happy with. The remainder of this paper discusses Karel the Robot and the advantages we see in starting the course by using an interesting object.

2 Karel and Karel's World

Karel inhabits a very simple world. There are avenues running north and south numbered one to infinity and streets running east and west, also numbered one to infinity. Walls may block avenues or streets. Beepers may be placed on the intersections of the avenues and streets. Several robots may exist within the same world.

Within this world, robots may move forward from one intersection to an adjacent intersection unless the way is blocked by a wall. They may turn left 90 degrees to face a

different direction. Robots may pick up a beeper from their current intersection (if one or more are present), or place a beeper on their current intersection (provided they are carrying at least one beeper). Robots may carry beepers between intersections in their "beeper bag" and may detect whether or not their beeper bag is empty.

Robots may also determine if they are facing north (or any of the other 3 compass positions), if there is a beeper on the current intersection or if there is another robot on the current intersection.

The world and the actions of any robots within it are shown visually on the computer monitor. A sample is shown in Figure 1. If the goal of the robot is to pick up all of the beepers, then one possible solution is shown in Figure 2. This example may be seen running as an applet at http://www.math.uwaterloo.ca/~bwbecker/papers/sigcse2001/samples.html. A second on-line example shows four robots solving the same problem. A third example uses threads so that four robots work simultaneously (instead of taking turns) to solve the problem.

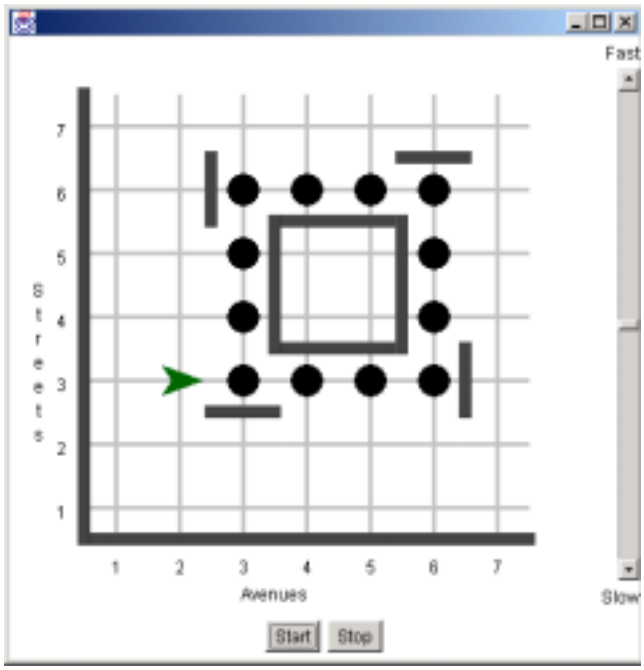

Figure 1: An initial situation with one robot (an arrowhead), twelve beepers (circles) and ten walls (rectangles).

```
package HarvestSquare;
import cs1Lib.karel.Robot;
import cs1Lib.karel.World;

public class Main extends Object
{
  public static void
      main(String[] args)
  { // Instantiate a new world, initialized with walls and
    // beepers as specified in a file.
    World square = new
              World("HarvestAroundSquare.txt");

    // Instantiate a new Robot and add it to the world at
    // the corner of 2nd avenue and 3rd street facing east.
    HarvesterRobot karel =
              new HarvesterRobot();
    square.addRobot(karel, 2, 3,
              World.EAST);

    // Move into the intial position, then pick the beepers
    // and turnoff.
    karel.move();
    karel.pickBoundedRectangle();
    karel.turnOff();
  }
}

/** A new kind of robot which knows how to harvest beepers
laid out in a rectangle with a wall marking the end of each
side. */
class HarvesterRobot extends Robot
{
  /** Pick the beepers in a rectangle where the end of each
  side is bounded by a wall. */
  public void pickBoundedRectangle()
  { for(int side=0; side<4; side++)
    { this.pickUntilWall();
      this.turnLeft();
    }
  }

  /** Pick one beeper from each intersection between the
  current location and the nearest wall in the direction faced
  by the robot. */
  private void pickUntilWall()
  { while (this.frontIsClear())
    { this.move();
      this.pickBeeper();
    }
  }
}
```

Figure 2: One solution to pick up all the beepers in the world shown in Figure 1.

3 Differences from Previous Work

Our implementation of Karel differs from *Karel++* and the original Karel in a number of significant ways. The most obvious is that we use Java while the previous two use their own languages. Karel uses a Pascal-like language while Karel++ uses a language similar to C++. Neither language, however, supports parameters or non-Robot variables while our approach allows the full power of the Java programming language – including objects which do not extend `Robot`, local and instance variables of any type and threads.

The previous implementations were programs which integrated a development environment with a simulator.

The robot and world classes were built right into the combined program. Our approach uses a standard Java development environment. When students write a robot program they simply `import` the required classes from a library. This removes the need to switch environments mid-way through the course, but is somewhat more complex at the beginning.

Because the previous implementations built the definition of the robot's world into the simulator, the behaviors of the world couldn't change. In our implementation the world is a fully extendable class. We encourage students to extend it with new behaviors for placing walls and beepers, for instance.

4 Course Outline

A typical CS1 course would use Karel for the first four to five weeks to introduce objects, inheritance, selection, iteration, and related concepts. After this introduction, robots would be left behind in favor of a wide range of examples.

4.1 Week 01: Instantiating and Using Objects

In the first week we describe robots and the world they inhabit. After reviewing several simple programs, students understand how to instantiate objects, invoke an object's methods and that it is possible to have several objects belonging to the same class, each with its own state. A typical homework problem is to instruct one or more robots to perform a fixed task such as retrieve a beeper from a given intersection.

4.2 Week 02: Extending Existing Classes

During the first week students often ask "Can a robot turn right?" (or move forward more than one intersection at a time or …) – which they can't. This is the perfect opportunity to extend the Robot class with new behaviors. For instance, to create a new class of robots capable of turning right, we write

```
import cs1Lib.karel.Robot;

public class RightRobot extends Robot
{ public void turnRight()
  { this.turnLeft();
    this.turnLeft();
    this.turnLeft();
  }
}
```

This basic idea is quickly expanded into step-wise refinement to solve a more complex problem such as picking up all the beepers in a fixed-size rectangular area. A natural extension, parameterized methods, is deferred to leave sufficient time for discussing inheritance and step-wise refinement thoroughly.

4.3 Week 03: Selection and Iteration

In the third week we tackle selection and iteration. The Robot class includes several methods which return Boolean values, including `frontIsClear` (is there a wall immediately in front of the robot?), `nextToABeeper` (is there a beeper on the same intersection as the robot?), `facingNorth` (is the robot facing North?), and `anyBeepersInBeeperBag` (is the robot carrying one or more beepers?). These allow a number of interesting problems such as the one shown in Figure 1, constructing a histogram out of beepers, or having the robot run a hurdles race where hurdles are represented by walls. Value-returning methods and Boolean expressions fit naturally with the discussion of the provided predicates.

4.4 Week 04: Methods with Parameters

Methods can be made much more general, of course, with parameters. Simpler problems include a `move` method which takes an integer distance parameter or a `harvestPlot` method which collects all the beepers from an area defined by the parameters. A more complex example is a `Contractor` robot which has several subcontractors to help build a house. The subcontractors are passed via parameters and give a good opportunity to compare passing primitive types with passing reference types.

4.5 Week 05: Instance Variables

In the last week spent with robots, we extend the `Robot` class with additional instance variables. We start with a `DeliveryRobot` which keeps track of how many moves it makes so that a customer can be charged for its services. After this relatively brief example we quickly move to other, non-robot, examples such as a date or bank account class.

After introducing the fundamentals of object-oriented programming, the remainder of the course focuses on a wide range of topics including arrays, graphical user interfaces, and object-oriented design.

4.6 Discussion

Taking five weeks to cover these topics seems slow, but it's not. Topics such as procedural decomposition, parameters, iteration and selection are hard for most beginners. They were hard back in the days of Pascal – and that hasn't changed just because we have moved to object-oriented languages. In the Pascal predecessor to this course we spent two weeks on control structures, two weeks on subprograms, and another week on records.

Many courses may be able to move faster. In fact, we do. By 1995 we observed that most of our incoming students had some previous programming experience. After making this a prerequisite for our "first" course (and providing a remedial course for those who didn't have it) we shaved a week off the schedule suggested above.

Some people assume robots are over-used and that students soon tire of them. This is not the case. We could not use them longer, but just over four weeks is not too long. Students enjoy the visual aspects and the anthropomorphism they make possible. In the course evaluations many students say they thoroughly enjoyed the robots; few mention them negatively.

We *do* need to mention often in class that we are only using robots as a tool to learn about programming. It's not a course about robots – it's a course about programming! We reinforce this with frequent allusions to other possible classes such as `Employee` or `Date` or `Account`, indicating that the concept under discussion applies to those situations, too.

5 Advantages

There are a number of advantages to using Karel the Robot to introduce objects. A number have been alluded to already. They are more carefully enumerated here.

5.1 Object-Oriented Fundamentals First

Karel the Robot emphasizes the fundamental concepts in OO programming (instantiating objects, invoking methods, extending existing classes) from the beginning. It's important that students be able to practice these concepts as long as possible and that they not be left to the middle or end of the course where they might be skimmed over lightly if time is tight.

Of course the foundations of procedural programming (selection, iteration, procedural decomposition) are also very important. The `Robot` class allows them to be naturally introduced with the OO fundamentals so that they, too, can be practiced throughout the course.

Many courses reverse these sets of fundamentals – introducing selection and iteration before using objects, sometimes treating objects as fancy Pascal records. There are at least two problems with this approach: a paradigm shift and missed opportunities.

Students are often asked to write programs typical of those in chapter 3 of [4]. These are programs which count, sum or generate comments based on user input. None of these programs use objects except for `System.out`, strings, and (once or twice) an instance of `NumberFormat`. Later, the students must shift from a "do it all myself right here" paradigm to a "find/build some objects to cooperate in getting the job done" paradigm.

The missed opportunities of this approach are in the richness of tasks that can be done with existing objects. Why should students be summing numbers when they could be doing more interesting things which are pedagogically just as valid?

Another common approach in existing textbooks is to make extensive use of `System.out` and the `String` class as example objects. These programs use objects but the object creation is hidden by the run-time system (in the case of `System.out`) or special language support (strings). *Creating* objects – including multiple instances from the same class – is an important part of students understanding what objects are and how they work.

By using a sufficiently complex and interesting predefined class such as `Robot` students can begin using the OO fundamentals from the beginning and avoid a paradigm shift later on.

5.2 Robots are Visual

The fact that robot worlds have a visual representation provides a number of benefits.

- Many problems can be specified using a picture of the initial situation and another of the final situation, plus a few lines of text. We find that students have fewer questions about homework problems specified this way.
- The animation provides visual feedback on the correctness of an algorithm. If the student's program doesn't result in the same image as the problem statement, there must be a bug (of course, just because it looks the same doesn't mean it was done right…).
- Students can often see where their program goes wrong simply by watching the animation. Because the human brain is highly optimized to process visual input this is faster and easier than, say, scanning a list of numbers from a console program.
- Because robots provide output visually, traditional input and output can be delayed. This is particularly attractive in Java where I/O is cumbersome, at best.

5.3 Robots are Fun

Robots are fun! This is, perhaps, the biggest advantage of the approach. Students enjoy directing robots to do various tasks. Acting out a program in class is more fun than tracing a listing. Visual output is more fun than textual output.

6 Related Work

Since we developed this course textbooks have, in general, moved objects earlier. Most, however, use a scattering of Java classes such as `String` or `Random` (which we find inferior to the approach recommended here) or expect students to write their own classes almost as soon as they learn about objects. See [1] for a more thorough treatment.

A few textbooks come closer. Slack [9] uses turtle graphics rather than a robot but uses a spiral approach. Chapter 2 is a difficult chapter, packed with concepts illustrated with turtle graphics. Chapters 4, 5, 7 and parts of 10 then come back to cover the concepts in more depth.

Morelli [5] uses a "CyberPet" as an interesting example to illustrate object-oriented fundamentals as well as iteration and selection. The difference is that it is not a pre-defined class, as are the robots. Instead students develop it as they go along.

Wu [10] gives a well-crafted rationale for using objects early. The objects his students use, however, are a simplified interactive I/O library used in the context of other problems.

Otterbein College uses Karel the Robot as an introduction to programming. They have a sophisticated environment developed by Duane Buck which supports a Java-like syntax (as well as Pascal and Lisp) and has an integrated compiler and execution environment. Differences from our approach is that our students use a Java development

environment, importing the necessary classes. Their students use one environment for Karel and a different one for other programs. In the Otterbein environment only a single robot can be used in any given program. The environment is available at http://math.otterbein.edu/JKarelRobot/. Their larger approach is discussed in [3].

Joseph Bergin has made his own translation of *Karel++* to Java available on the web, along with classes implementing the robot. It may be found at http://csis.pace.edu/~bergin/KarelJava/Karel++JavaEdition.html.

Our own work is at http://www.undergrad.math.uwaterloo.ca/~cs130/.

Rich Pattis, the originator of Karel the Robot, has extended the idea into artificial life simulations. The result is a much more complex set of classes for students to master, but one that also affords many more possibilities [7].

7 Future Directions

The largest difficulty we have encountered is that *Karel++*, and thus our translation of that book, is only an introduction to object-oriented programming. It doesn't cover numerous topics expected in a CS1 course and so we jump to a standard text after several weeks. Students often complain of this discontinuity.

To overcome this I am writing a full CS1 textbook which starts with Karel but also includes all the expected CS1 topics. This text will also include a number of innovations beyond those discussed here.

First, we currently discuss attributes of an object (instance variables) quite late. After stressing in week 1 that an object has both behavior and attributes, it seems wrong to focus on behavior for so long without mentioning attributes. In the text I will cover object usage, extending objects (behaviors), selection, and then attributes, bringing two core object-oriented concepts into a much closer relationship.

Second, there will be a richer collection of classes. In addition to the current `Robot` and `World` (renamed `City`) will be a `CityBlock` – the immediate environment of a robot. Students will be able to extend the CityBlock class and robots will be able to interact with `CityBlock` objects. Picking up and putting down beepers then becomes just one example of possible interactions. After studying how classes interact to implement beepers, students might go on to implement a robot that rakes leaves into piles or spreads piles of salt onto the streets or city blocks that display a different color after being visited by a robot. Many of these projects make good use of instance variables. This also allows simulations inspired by artificial life [7, 8].

Third, I am introducing a parallel track which reinforces concepts using non-robot objects. Currently too many students don't recognize that they are learning general concepts via Karel the robot. Including other examples should alleviate this.

Since students love graphics and GUI applications, I will follow [4] and others in providing a "GUI Supplement." In chapter 1 we will instantiate and invoke methods on a robot object followed by instantiating and invoking methods on a `JFrame` object. In chapter 2 students will extend robots to do new things and also extend `JFrame` to do something new – paint a scene. Other chapters will follow a similar structure.

8 Conclusions

We have been extremely pleased with using interesting predefined classes (Karel the Robot and its world) to teach object-oriented fundamentals (object instantiation, method calls, inheritance) and traditional procedural fundamentals (iteration, selection) early in the course. The result has been a course that is fun for both students and instructors, and where students understand the fundamental concepts early, allowing them to approach advanced topics with confidence.

References

[1] Becker, Byron Weber. *Pedagogies for Teaching CS1 in Java*. http://www.math.uwaterloo.ca/~bwbecker/papers/sigcse2001/javaPedagogies/index.html.

[2] Bergin, Joseph, Mark Stehlik, Jim Roberts, and Richard Pattis. *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming*. John Wiley & Sons, 1997.

[3] Buck, Duane and David J. Stucki. Design Early Considered Harmful: Graduated Exposure to Complexity and Structure Based on Levels of Cognitive Development. *SIGCSE Bulletin* 1 (2000), p. 75-79.

[4] Lewis, John and William Loftus. *Java Software Solutions.* Addison-Wesley, 2000.

[5] Morelli, Ralph. *Java, Java, Java: Object-Oriented Problem Solving*. Prentice-Hall, 2000.

[6] Pattis, Richard E.. *Karel the Robot: A Gentle Introduction to the Art of Programming*, John Wiley & Sons, 1981.

[7] Pattis, Richard E. Teaching OOP in C++ Using an Artificial Life Framework. *SIGCSE Bulletin* 1 (1997), p. 39-43.

[8] Resnick, Mitchel. *Turtles, termites, and traffic jams: explorations in massively parallel microworlds.* The MIT Press, 1994.

[9] Slack, James M. *Programming and Problem Solving with Java*. Brooks/Cole, 2000.

[10] Wu, C. Thomas. *An Introduction to Object-Oriented Programming with Java*. WCB/McGraw-Hill, 1999.