Assignment 1: Introductions and Karel the Robot

Based on a handout by Eric Roberts and Mehran Sahami

In this first assignment, we'll get to know more about you, you'll see just how powerful Karel the Robot can be as you use Karel to solve a variety of programming problems, and you'll reflect on how you went about composing your solutions.

Assignment due: Tuesday, July 1 at 4:00pm PDT

Part One: Getting to know you (getting to know all about you!)

We'd love to get to know you in CS106A! As your first task, there is a text file in the assignment folder called `hello.txt` that contains a handful of questions we'd like you to answer. In that text file go ahead and answer the questions, save the file and include it in your submission folder when you submit. Your section leader (who may not be assigned to you until after you submit) will then get to read it when they grade your assignment. Also, we'd love it if you send an email to Tom, Yves and Karl with the same answers to the questions (you can just copy and paste the text into the message body of the email).

Part Two (the real deal): Karel the Robot

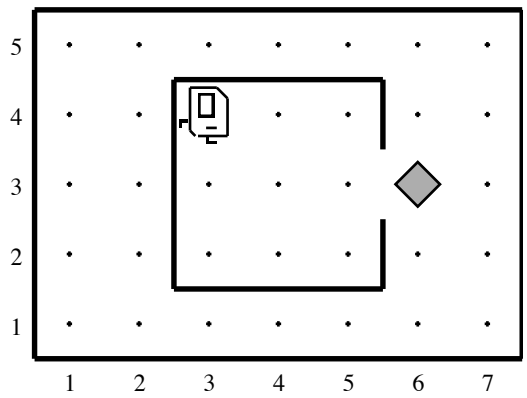
The meat of this assignment consists of four Karel programs. First, make sure that you've downloaded and installed Eclipse per the instructions on the course website. There is a starter project including all of these problems on the CS106A web site in the area for Assignment 1. To work on these programs, download that starter folder as described in Handout #02 (Using Karel in Eclipse). From there, you can edit the program files so that the assignment actually does what it's supposed to do, which will involve a cycle of coding, testing, and debugging until everything works. The final step is to submit your assignment using the **Submit Project** entry under the **Stanford Menu**. You can submit your programs as you finish them and that you can submit more than one version of your project. If you discover an error after you've submitted one of these problems, just fix your program and submit a new copy.

Also, please remember that your Karel programs must limit themselves to the language features described in *Karel the Robot Learns Java* in the `Karel` and `SuperKarel` classes. You may not use other features of Java, even though the Eclipse-based version of Karel accepts them.

We recommend reading through Handout #01 about the Stanford Honor Code before starting this assignment to make sure that you are aware of our Honor Code policies.

Problem 1: `CollectNewspaperKarel`

Your first task is to solve a simple story-problem in Karel's world. Suppose that Karel has settled into its house, which is the square area in the center of the following diagram:



Karel starts off in the northwest corner of its house as shown in the diagram. The problem you need to get Karel to solve is to collect the newspaper – represented (as all objects in Karel's world are) by a beeper – from outside the doorway and then to return to its initial position.

This exercise is extremely simple and exists just to get you started. You can assume that every part of the world looks just as it does in the diagram. The house is exactly this size, the door is always in the position shown, and the beeper is just outside the door. Thus, all you have to do is write the sequence of commands necessary to have Karel

1. Move to the newspaper,
2. Pick it up, and
3. Return to its starting point.

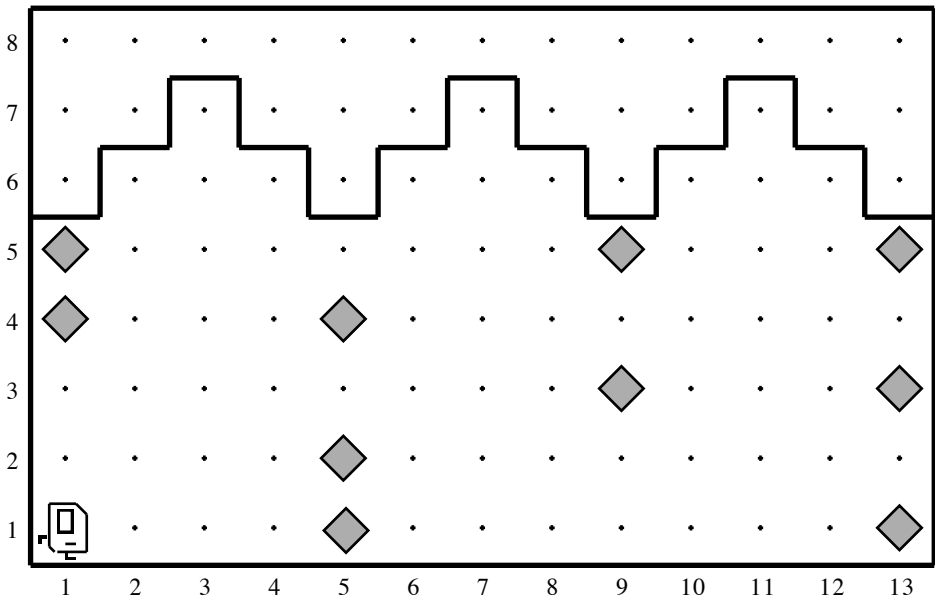
Even though the program is only a few lines, it is still worth getting at least a little practice in decomposition. In your solution, include a private method for each of the steps shown in the outline.

A Word of Advice

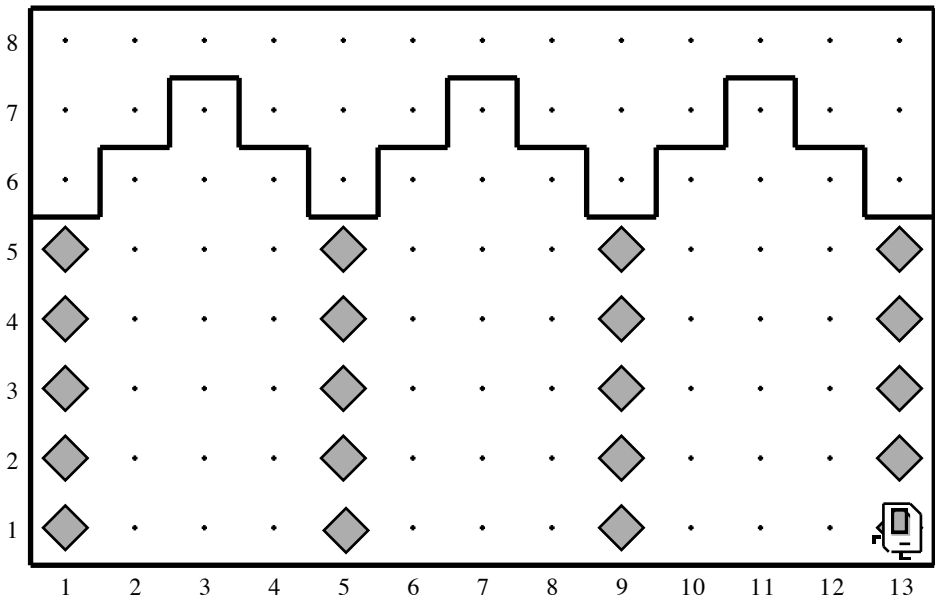
You could go on to the harder problems on this assignment, but you might why want to try submitting your project as soon as you are done with this first problem. Every year, a handful of students run into some kind of problem with the electronic submission option provided in the Stanford version of Eclipse. If you wait until 3:55P.M. the day the assignment is due to submit any of your work, you may discover that there is some aspect of the submission process that you didn't quite understand only after it's too late to get any help. So right now, as soon as you've got this first program working (and as soon as you get the green light from the course staff that submissions are open), go ahead and hit the submit button to make sure that you can ship things off. Once you've done so, you'll know that you've got the submission process under control. Remember, we only look at the last submission you make before the due date, so it doesn't hurt to submit new versions of your solution as you finish them.

Problem 2: `StoneMasonKarel`

Karel has been hired to repair the damage done to the Quad in the 1989 earthquake. In particular, Karel is to repair a set of arches where some of the stones (represented by beepers, of course) are missing from the columns supporting the arches, as follows:



When Karel is done, the missing stones in the columns should be replaced by beepers, so that the final picture resulting from the world shown above would look like this:



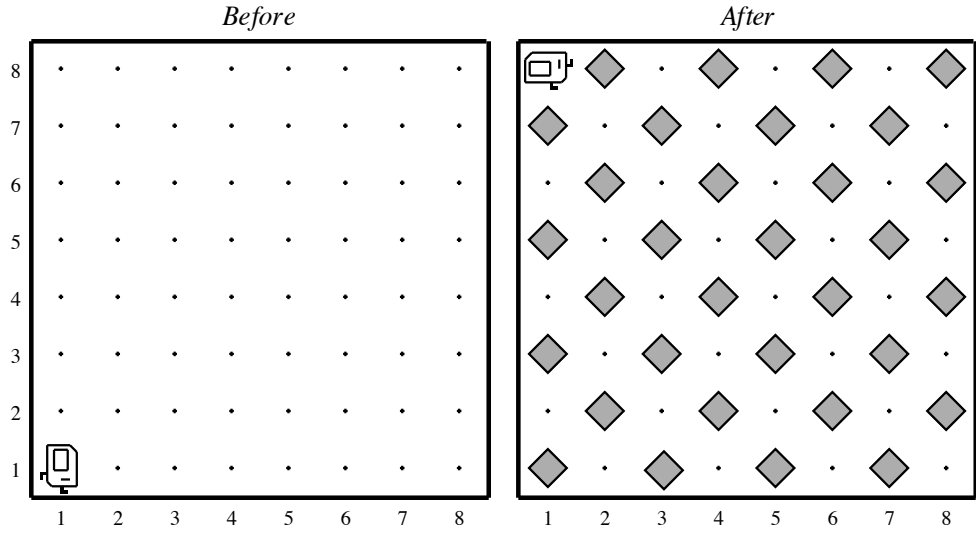
Your program should work on the world shown above, but it should be general enough to handle any world that meets certain basic conditions as outlined at the end of this problem. There are several example worlds in the starter folder, and your program should work correctly with all of them.

Karel's final location and the final direction he is facing at end of the run do not matter. Karel may count on the following facts about the world, listed below:

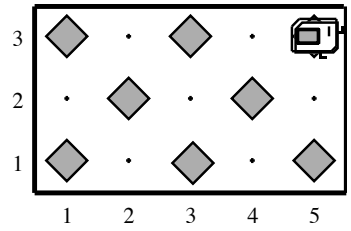
- Karel starts at 1st Avenue and 1st Street, facing east, with an infinite number of beepers in Karel's beeper bag.
- The columns are exactly four units apart, on 1st, 5th, 9th Avenue, and so forth.
- The end of the columns is marked by a wall immediately after the final column. This wall section appears after 13th Avenue in the example, but your program should work for any number of columns.
- The top of the column is marked by a wall, but Karel cannot assume that columns are always five units high, or even that all columns are the same height.
- Some of the corners in the column may already contain beepers representing stones that are still in place. Your program should not put a second beeper on these corners.

Problem 3: `CheckerboardKarel`

In this exercise, your job is to get Karel to create a checkerboard pattern of beepers inside an empty rectangular world, as illustrated in the following before-and-after diagram. (Karel's final location and the final direction it is facing at end of the run do not matter.)



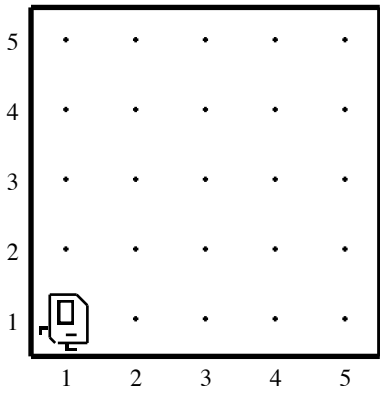
As you think about how you will solve the problem, you should make sure that your solution works with checkerboards that are different in size from the standard 8x8 checkerboard shown in the example. Odd-sized checkerboards are tricky, and you should make sure that your program generates the following pattern in a 5x3 world:



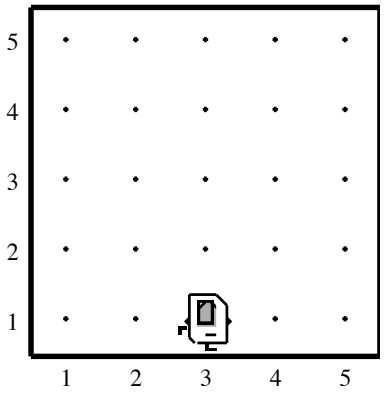
Another special case you need to consider is that of a world which is only one column wide or one row high. The starter folder contains several sample worlds that test these special cases, and you should make sure that your program works for each of them.

Problem 4: `MidpointFindingKarel`

As an exercise in solving algorithmic problems, program Karel to place a single beeper at the center of 1st Street. For example, if Karel starts in the world



Karel should end standing on a beeper in the following position:



Note that the final configuration of the world should have only a single beeper at the midpoint of 1st Street. Along the way, Karel is allowed to place additional beepers wherever it wants to, but must pick them all up again before it finishes.

In solving this problem, you may count on the following facts about the world:

- Karel starts at 1st Ave. and 1st St., facing east, with an infinite number of beepers in its bag.
- The initial state of the world includes no interior walls or beepers.
- The world need not be square, but you may assume that it is at least as tall as it is wide.
- Your program, moreover, can assume the following simplifications:
- If the width of the world is odd, Karel must put the beeper in the center square. If the width is even, Karel may drop the beeper on either of the two center squares.
- It does not matter which direction Karel is facing at the end of the run.

The interesting part of this assignment is to come up with a strategy that works. There are *many* different algorithms you can use to solve this problem (the section leaders and I have easily seen at least ten different approaches), so be creative and have fun coming up with one!

Advice, Tips, and Tricks

All of the Karel problems you will solve (except for **`CollectNewspaperKarel`**) should be able to work in a variety of different worlds. When you first run your Karel programs, you will be presented with a sample world in which you can get started writing and testing your solution. However, we will test your solutions to each of the Karel programs (except for **`CollectNewspaperKarel`**) in a variety of test worlds. Unfortunately, each quarter, many students submit Karel programs that work brilliantly in the default worlds but which fail catastrophically in some of the other test worlds. Before you submit your Karel programs, *be sure to test them out in as many different worlds as you can*. We've provided several test worlds in which you can experiment, but you should also develop your own worlds for testing.

When writing your Karel programs, to the maximum extent possible, try to use the top-down design techniques we developed in class. Break the task down into smaller pieces until each subtask is something that you know how to do using the basic Karel commands and control statements. These Karel problems are somewhat tricky, but appropriate use of top-down design can greatly simplify them.

As mentioned in class, it is just as important to write clean and readable code as it is to write correct and functional code. A portion of your grade on this assignment (and the assignments that follow) will be based on how well-styled your code is. Before you submit your assignment, take a minute to review your code to check for stylistic issues like these:

Have you added comments to your methods? To make your program easier to read, you can add comments before and inside your methods to make your intention clearer. Good comments give the reader a clue about what a method does and, in some cases, how it works. Did you add comments to your methods to indicate what they do?

Not-so-Good Code	Good Code

`public void fillRowWithBeepers() {`
`  while (frontIsClear()) {`
`  putBeeper();`
`  move();`
`  }`
`  putBeeper();`
`}`
	`/* Makes Karel move to the end of`
	` * the row, dropping a beeper`
	` * before each step he takes.`
	` *`
	` * Precondition: None`
	` * Postcondition: Karel is facing`
	` * the same direction as before,`
	` * and every step between Karel's`
	` * old position and new position`
	` * has had a beeper added to it.`
	` */`
	`public void fillRowWithBeepers() {`
	`  while (frontIsClear()) {`
	`  putBeeper();`
	`  move();`
	`  }`
	`  putBeeper();`
	`}`

Did you decompose the problem? There are many ways to break these Karel problems down into smaller, more manageable pieces. Breaking the problem apart elegantly will result in a small number of easy-to-read methods, each of which performs just one small task. Breaking the problem apart in other ways may result in methods that are trickier to understand and test. Look over your code and check to see whether you've decomposed the problem into smaller pieces. Does your code consist of a few enormous methods (not so good), or many smaller methods (good)?

Is your code indented properly? In Java, each line of code can be indented by any amount. Does your indentation help show how the different pieces of code are related to one another? For example:

Not-so-Good Code	Good Code

`public void run() {`
`move();`
`while (frontIsClear()) {`
`move();`
`turnRight();`
`if (beepersPresent()) {`
`pickBeeper();`
`}`
`}`
`}`
	`public void run() {`
	`move();`
	`while (frontIsClear()) {`
	`move();`
	`turnRight();`
	`if (beepersPresent()) {`
	`pickBeeper();`
	`}`
	`}`
	`}`

This is not an exhaustive list of stylistic conventions, but it should help you get started. As always, if you have any questions on what constitutes good style, feel free to stop on by the Tresidder LaIR with questions, come visit us during office hours, or email your section leader with questions!

Part Three: Karel Writeup

As a last step in this assignment, we'd like you to answer three short questions about the programs you've written. There is a file called **`writeup.txt`** included with the starter code for this programming assignment. Before you submit your final version of the assignment, please answer those questions. Your section leader can then discuss your answers during your interactive grading session for this assignment to help you improve your programming technique and style.

Your grade for the writeup is based purely on whether or not you actually answer the questions rather than on the quality of your answers. That said, we'd appreciate it if you answered them honestly so that your section leader can offer better feedback.

For reference, the questions we'd like you to answer in the **`writeup.txt`** file are listed here.

1. The Karel programs you wrote in the last three parts of this assignment need to work in a variety of worlds. How did you test your code to see if it would work correctly in multiple different worlds?
2. There are many different approaches you can take when writing your solutions to the **`CheckerboardKarel`** and **`MidpointFindingKarel`** problems. When you were working on these problems, did you consider any solution strategies other than the one you chose to implement? If so, what other approaches did you consider?
3. Choose one of the Karel problems other than **`CollectNewspaperKarel`**. Describe how you decomposed the program into smaller methods. How did you decide how to split the code across multiple methods? Did you consider any decompositions other than the one you ended up choosing?