Eric Roberts
CS106A

Handout #35
May 2, 2012

Practice Midterm Examination #1

Review session: **Sunday, May 6, 7:00–9:00 P.M., Hewlett 200**
Midterm exams: **Tuesday, May 8, 9:00–11:00 A.M., CEMEX Auditorium**
Tuesday, May 8, 3:15–5:15 P.M., Braun Auditorium

This handout is intended to give you practice solving problems that are comparable in format and difficulty to those which will appear on the midterm examination next Tuesday. A solution set to this practice examination will be distributed on Friday, along with a second practice exam.

Time and place of the exam

The midterm exam is scheduled at two different times to accommodate those of you who have scheduling constraints. You may take the exam at either time and need not give advance notice of which exam you plan to take. If you cannot take the exam at either of the two scheduled times or if you need special accommodations, please send an e-mail message to `eroberts@cs` stating the following:

- The reason you cannot take the exam at either of the scheduled times and/or the details of the special accommodations you require.
- A two-hour period on Tuesday or Wednesday at which you could take the exam. This time must be during the regular working day, and must therefore start between 8:30 and 3:00 (so that it ends by 5:00).

In order to schedule an alternate exam, I must receive an e-mail message from you by 5:00 P.M. on Friday, May 4. Late requests will not be honored. Instructions for taking the alternate midterm will be sent to you by electronic mail on Monday.

Review session

There will be a review session on Sunday evening from 7:00 to 9:00 P.M. at which we will go over the solutions to the practice exams and answer questions.

Coverage

The exam covers the material presented in class through Monday, April 30, which means that you are responsible for the Karel material plus Chapters 1–6 and 8–9 of *The Art and Science of Java*, plus the use of mouse listeners from Chapter 10.

Note: To conserve trees, I have cut back on answer space for the practice midterm. The actual exam will have much more room for your answers and for any scratch work.

Please remember that the midterm is <u>open-book.</u>
9:00–11:00 in CEMEX Auditorium
3:15–5:15 in Braun Auditorium

General instructions

Answer each of the five questions included in the exam. Write all of your answers directly on the examination paper, including any work that you wish to be considered for partial credit.

Each question is marked with the number of points assigned to that problem. The total number of points is 60. We intend for the number of points to be roughly comparable to the number of minutes you should spend on that problem. This leaves you with an additional hour to check your work or recover from false starts.

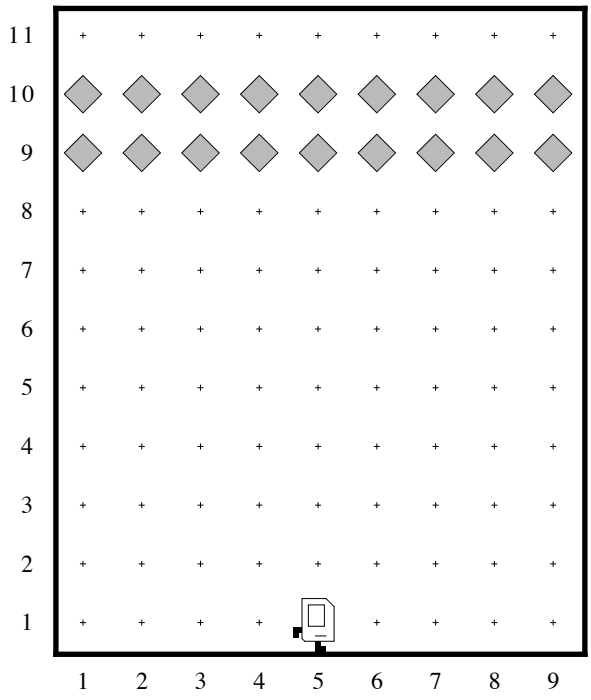
In all questions, you may include methods or definitions that have been developed in the course, either by writing the `import` line for the appropriate package or by giving the name of the method and the handout or chapter number in which that definition appears.

Unless otherwise indicated as part of the instructions for a specific problem, comments will not be required on the exam. Uncommented code that gets the job done will be sufficient for full credit on the problem. On the other hand, comments may help you to get partial credit if they help us determine what you were trying to do.

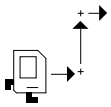
The examination is open-book, and you may make use of any texts, handouts, or course notes. You may not, however, use a computer of any kind.

Problem 1: Karel the Robot (10 points)

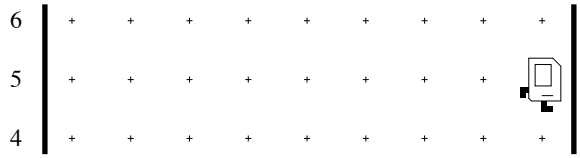
Not wanting to miss out on all the fun, Karel has decided that it too can learn to play Breakout! Imagine that Karel starts out in a world that looks like this:



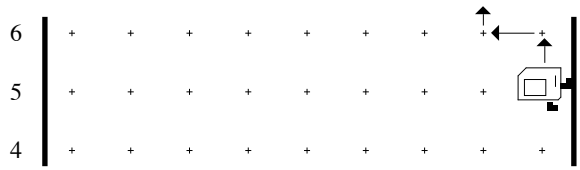
Your job is to teach Karel to play a simple game of Breakout. The first step is to get Karel to move in a more-or-less diagonal line, which of course really requires Karel to move in little stair steps like this:



Fairly soon, Karel will come up against the east wall of the world and find itself in the following position on 5th Street:



Karel now has to bounce, just as in the Java Breakout game you wrote. Bouncing sounds as if it might be tricky, but since Karel is always moving at 45°, you can implement a standard bounce operation simply by turning left. Doing so puts Karel in the following position, from which it can again implement its stair-step diagonal motion:



As Karel proceeds, it will eventually hit one of the beepers near the top of the world. When it does, Karel can make it look like Breakout simply by picking the beeper up and then bouncing just as if it had hit a wall. Karel then keeps going, bouncing off beepers and walls (including the south wall, since there is no paddle).

There are two important aspects of this problem that you should keep in mind:

1. If Karel ends up in a corner, it will have to bounce twice—once off each wall—to avoid being blocked. The easiest way to manage this is to have Karel check if it is still blocked after making its first bounce, and, if so, bounce again.
2. Karel cannot simply take both steps in its diagonal motion without checking to make sure that (a) it can move in each of the directions without hitting a wall and (b) that there is no beeper on the square in the middle, which it would surely hit first. Thus, you have to check for bounces halfway through each step as well as at the end.

Write the program to implement this simulation of Breakout. In your solution, you may use any Karel method defined in the course handouts just by giving its name. For example, you may use **`turnRight`** or **`moveToWall`** without writing down the complete definition. You may also assume that Karel always begins facing east on 1st Street and that the world is at least 2×2 in size. You may also make the following assumptions:

1. *Don't worry about getting Karel to stop.* Given that Karel can't count, there isn't any easy way for Karel to determine whether the bricks are gone. The easiest thing to do

is to let the program run forever. You can achieve this goal without violating Karel's rules by starting with at least one beeper in the bag and using a loop of the form

```
while (beepersInBag()) {
    Have Karel take one diagonal step, bouncing as appropriate.
}
```

2. *Don't worry about whether Karel can in fact hit all the beepers.* Given that there is no paddle and everything is behaving deterministically, Karel could easily (and indeed eventually does) get stuck in a loop where it just keeps going back and forth without hitting the last few beepers.

Problem 2: Simple Java expressions, statements, and methods (10 points)

(2a) Compute the value of each of the following Java expressions. If an error occurs during any of these evaluations, write "Error" on that line and explain briefly why the error occurs.

`5.0 / 4 - 4 / 5`	_____
`7 < 9 - 5 && 3 % 0 == 3`	_____
`"B" + 8 + 4`	_____

(2b) Assume that the method **`mystery`** has been defined as given below:

```
private String mystery(String s) {
   String result = "";
   int len = s.length();
   int j = 0;
   int k = 9;
   while (j < k) {
      if (j < 4) {
         result += s.charAt(k % len);
      }
      if (j / 2 != 1) {
         result += s.charAt(j % len);
      }
      j++;
      k--;
   }
   return result;
}
```

What is the value of

```
mystery("abcdefg")
```

(2c) What output is printed by the following program:

```
/*
 * File: Problem2c.java
 * --------------------
 * This program doesn't do anything useful and exists only to
 * test your understanding of parameters and string methods.
 */

import acm.program.*;

public class Problem2c extends ConsoleProgram {

   public void run() {
      String s1 = "To err";
      String s2 = "is human!";
      s1 = forgive(s1, s2);
      println(s1 + " " + s2);
   }

   private String forgive(String me, String you) {
      String heart = me.substring(0, you.length() - me.length());
      you = "" + you.charAt(me.length());
      int amount = heart.length();
      me = me.substring(amount + 2) + me.charAt(amount);
      heart += understanding(you, 2) + you + me;
      return heart;
   }

   private char understanding(String you, int num) {
      return (char)(you.charAt(0) + num);
   }

}
```

Problem 3: Simple Java programs (15 points)

In Assignment #2, you wrote a program to find the largest and smallest integers in a list entered by the user. For this problem, write a similar program that instead finds the largest and the second-largest integer. As in the homework problem, you should use 0 as a sentinel to indicate the end of the input list. Thus, a sample run of the program might look like this:

```
SecondLargest
This program finds the two largest integers in a
list.  Use 0 to signal the end of the input.
 ? 73
 ? 77
 ? 68
 ? 36
 ? 84
 ? 91
 ? 44
 ? 0
The largest value is 91
The second largest is 84
```

To reduce the number of special cases, you may make the following assumptions in writing your code:

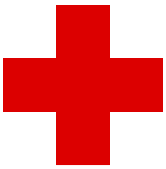
- The user must enter at least two values before the sentinel.
- All input values are positive integers.
- If the largest value appears more than once, that value should be listed as both the largest and second-largest value, as shown in the following sample run:

```
SecondLargest
This program finds the two largest integers in a
list.  Use 0 to signal the end of the input.
 ? 1
 ? 8
 ? 5
 ? 8
 ? 0
The largest value is 8
The second largest is 8
```

Problem 4: Using the graphics and random number libraries (15 points)

Write a **`GraphicsProgram`** that does the following:

1. Creates the following cross as a **`GCompound`** containing two filled rectangles:



   The color of the cross should be red, as in the emblem of the International Red Cross (it actually is red in the diagram, but that's hard to see on the printed copies, which are of course in black and white). The horizontal rectangle should be 60x20 pixels in size and the vertical one should be 20x60. They cross at their centers.
2. Adds the cross to the canvas so that it appears at the center of the window.
3. Moves the cross at a speed of 3 pixels every 20 milliseconds in a random direction, which is specified as a random real number between 0 and 360 degrees.
4. Every time you click the mouse inside the cross, its direction changes to some new random direction; its velocity remains the same. Clicks outside the cross have no effect.

If you were actually to write such a program, you would presumably supply some means of making it stop, such as when the cross moves off the screen. For this problem, just have the program run continuously in a loop without worrying about objects moving off screen or how to stop the simulation.

Problem 5: Strings and characters (10 points)

> *The waste of time in spelling imaginary sounds and their history (or etymology as it is called) is monstrous in English . . .*
>
> —George Bernard Shaw, 1941

In the early part of the 20th century, there was considerable interest in both England and the United States in simplifying the rules used for spelling English words, which has always been a difficult proposition. One suggestion advanced as part of this movement was the removal of all doubled letters from words. If this were done, no one would have to remember that the name of the Stanford student union is spelled "Tresidder," even though the incorrect spelling "Tressider" occurs at least as often. If double letters were banned, everyone could agree on "Tresider."

Write a method **`removeDoubledLetters`** that takes a string as its argument and returns a new string with all doubled letters in the string replaced by a single letter. For example, if you call

```
removeDoubledLetters("tresidder")
```

your method should return the string **`"tresider"`**. Similarly, if you call

```
removeDoubledLetters("bookkeeper")
```

your method should return **`"bokeper"`**.

In writing your solution, you should keep in mind the following:

- You do not need to write a complete program. All you need is the definition of the method **`removeDoubledLetters`** that returns the desired result.
- You may assume that all letters in the string are lower case so that you don't have to worry about changes in capitalization.
- You may assume that no letter appears more than twice in a row. (It is likely that your program will work even if this restriction were not included; we've included it explicitly only so that you don't even have to think about this case.)