# Karel Prepares for the Olympics! (25 points)



Inspired by the events of the 2016 Rio Olympics (where Stanford athletes earned more medals than all but nine countries) as well as the 2018 Men's World Cup, Karel has decided to train for the 2020 Olympics in Tokyo. But in order to become the world's best striker, Karel first needs to learn ball control!

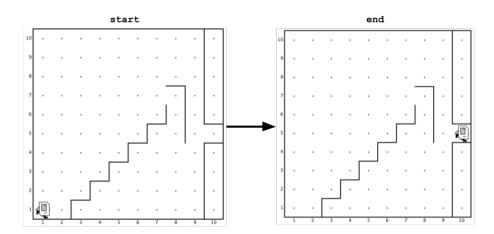
Write a program called StrikerKarel to help Karel learn the basics of the role:

```
public class StrikerKarel extends SuperKarel
```

Karel starts off at the corner of (1,1), standing on top of the ball (a beeper). Karel has no beepers in their bag and there are no other beepers in the world. There is one "goal" in the final column of the world, represented by an opening in a wall between the last and second-to-last columns. Karel needs to move the ball from (1,1) to this "goal".

Two challenges make Karel's task more difficult:

- In order to simulate dribbling, Karel cannot move more than once without the ball touching the ground. That is, the ball must be put
  down each time Karel changes location.
- In order to simulate defenders, there are walls blocking Karel's potential paths. Karel must move around these walls.



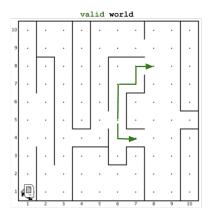
### Assumptions and Specifications

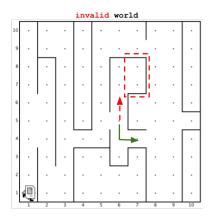
- The world will have at least two columns but may be any height.
- There will be exactly one "goal" in the final column. The goal corner is the only corner in the world that is blocked to the north, east, and south simultaneously.
- At every reachable corner other than the goal, Karel will be able to find an opening in the wall that allows them to move east without first moving west.
- Karel's ending direction does not matter, so long as Karel is in the goal and on top of a beeper.
- You are limited to the Java instructions shown in the Karel coursereader. This means the only variables allowed are loop control variables used within the control section of a for loop. You are not allowed to use syntax like local variables, instance variables, parameters, return values, Strings, return or break, etc.

For example, the world on the right is  ${f not}$  valid because if Karel moves north following the red dashed arrow,

they will be trapped by walls to the north, south, and east in the column indicated by the red d  $% \left( 1\right) =\left( 1\right) +\left( 1\right) +\left$ 

Karel would have to move west in order to move east again. Your program  $\mathbf{does}$   $\mathbf{not}$  need to handle this kind of world.





# You Prepare to Be a Section Leader! (20 points)



### Part A: Getting grilled (10 points)

It's time for section, and your students are grilling you on how to evaluate complicated expressions!

For each expression below, determine its final value. Be sure to write a literal value of the appropriate type (e.g., 7.0 rather than 7 for a

```
double, Strings in double quotes, chars in single quotes, true/false for a boolean).

i. (double) (16 / 5) * 10
```

```
i. (double)(16 / 5) * 10

ii. 'D' - 'A' == '3'

iii. 2 + 5 + "M" + 2 * 5 + 2

iv. 200 + 19 % 10 - (42 / 10.0 / 2) * 100

v. !((false || 3 != 4) && !(7 / 2.0 < 3.5))</pre>
```

### Part B: LaIR Lunacy (10 points)

Now you are helping someone in the LaIR, but they have not named their variables or methods descriptively :-(

Trace through the program's execution to figure out what is happening. For each <code>println</code> call along the way, indicate what the program would output at that point in the execution.

**IMPORTANT:** You must enter your answers in the order the program prints them. To this end, each <code>println</code> has a comment indicating which answer it corresponds to. Correct answers in the wrong order will not earn full credit.

```
public void run() {
   int alexandria = 14;
   int ben = 18;
   alexandria = tinky(ben, alexandria);
   println(ben); // ii
   GOval cory = new Goval(19, 87, 19, 91);
   int deray = 66;
   winky(cory, deray);
   println(cory.getHeight()); // iii
   println(deray); // iv
   String elizabeth = "NSP";
   elizabeth.toLowerCase();
   println(elizabeth); // v
}

private int tinky(int alexandria, int ben) {
   alexandria /= 2;
   ben /= 2;
   println(alexandria); // i
   return ben;
```

```
private void winky(GOval cory, int deray) {
  deray = 76;
  cory.setSize(19, 92);
  deray++;
}
```

# Colin Prepares for the Saturday Crossword! (40 points)



(The series of events described in this problem is 100% true.)

A few weeks ago, Colin got a subscription to the New York Times daily crossword puzzle. He has been having lots of fun solving the easier puzzles but also wants to work up to the harder ones. After one particularly frustrating attempt at a Saturday puzzle, Colin found a series of blog posts documenting one person's journey to solving a Saturday puzzle independently. In the posts, the blogger describes a Crossword Trainer program he wrote to help him improve.

Colin was intrigued, but he was too busy writing exam questions to write his own Crossword Trainer program. Instead, he wants you to write it for him!

# Part A: Test the Teacher (15 points)

Let's begin by writing the core strategy for the Crossword Trainer program. The program will display a clue to Colin, and he will have to guess the answer. To simulate solving a clue in a partially-completed puzzle, you should randomly reveal some, none, or all of the answer's letters to Colin before asking for his guess. Write a method called testoneClue:

```
private boolean testOneClue(String clue, String answer, double revealChance)
```

This method takes three parameters:

- clue: a clue to show Colin
- answer: the clue's answer
- revealChance: a number between 0 and 1 (inclusive) that serves as the probability that each letter in answer will be revealed as part of a hint before Colin makes a guess

This method should return true if the Colin's guess is correct and false if not.

### Assumptions and Specifications

- The method should begin by printing the crossword clue.
- After displaying the clue, you should print a hint in which each letter of answer is revealed with probability revealChance. If a letter should not be revealed, you should represent it as a dash "-".

For example, a hint for the answer "PYRE" could be "---E".

- $\bullet$  After displaying the hint, you should prompt Colin to enter a guess with "Guess: " .
- · Your guess verification should be case-insensitive; that is, capitalization should not affect Colin's correctness.
- You may assume that all of the letters in answer will be capitalized.

```
MyCrosswordTrainer [completed]
              Probability of revealing a letter? 2
              Enter a valid probability between 0 and 1: 0.4
              Smelly cheese
               -R--
               Guess: brie
               Correct!
              Viking's farewell
testOneClue
   (Part A)
              Guess: FiRe
               The answer was: PYRE
               Smartest farm animal
              Guess: PIG
               Correct!
               Percent correct: 66,66666666666667%
```

# Part B: Feed in the File (25 points)

Now, let's build the rest of our Crossword Trainer. For this part, assume you have a perfect implementation of testOneClue from Part A. Here's how it works: You will be provided a text file named "crossword.txt", which has the following format:

```
[number of clues]
[clue]#[ANSWER]
[clue]#[ANSWER]
[clue]#[ANSWER]
...
```

For example, "crossword.txt" might look like this:

```
3
Smelly cheese#BRIE
Viking's farewell#PYRE
Smartest farm animal#PIG
```

You should begin by asking Colin what probability he wants for revealing each letter. Then, test Colin on each clue in "crossword.txt" After Colin has gone through all clues, print the percentage of clues he guessed correctly. You should be able to match the full sample run above.

### Assumptions and Specifications

- There will always be at least one clue in the file.
- You should make sure that Colin enters a probability between 0 and 1, inclusive. If not, print "Enter a valid probability between 0 and 1: " and make him re-enter a probability.
- · You may assume that Colin will enter a number for the probability.
- You may assume that the '#' character will not appear within any answers.
- The '#' character may, however, appear within clues.
- You should print "Correct!" on correct guesses and "The answer was: [answer]" on incorrect guesses.
- There should be an empty line between clues.

# Candidates Prepare for the Midterm Elections! (35 points)



Since winning their respective primaries, candidates for the contested Senate seat in Wyoming have been hard at work campaigning. Moreover, their campaign teams have been hard at work canvassing to gauge public opinion.

You are independent pollster employed by one of the campaign teams to gather data on how their candidate is doing. With midterm elections approaching, you are about to put together your final report... only to find that your intern was actually working for the other candidate and just deleted all of your data!

In a panic to provide something to the campaign manager, you decide to violate the real-life researcher honor code and fabricate all of your data. Sad times :-(

Write a program called  ${\tt PollsterPanic}$ , described below, to generate your fake polling data:

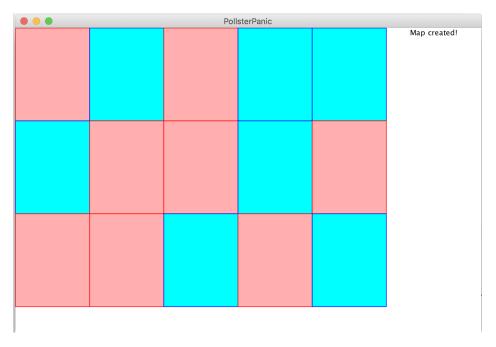
public class PollsterPanic extends GraphicsProgram

# Part A: Arbitrary Assignments (20 points)

First, generate a grid consisting of **15 equally-sized rectangular** zones within the conveniently rectangular state of Wyoming. When you are done creating the grid, you should display the message, "Map created!". This message must be at the top of the screen, centered in the space between the right edge of the map and the right edge of the window.

You should set each zone to a color randomly chosen from the options of red and blue. If it is helpful, you may assume for this problem that these are the only two colors that exist. It is not important whether the zones have their colors filled in on the grid, but it is important that each zone be assigned a color.

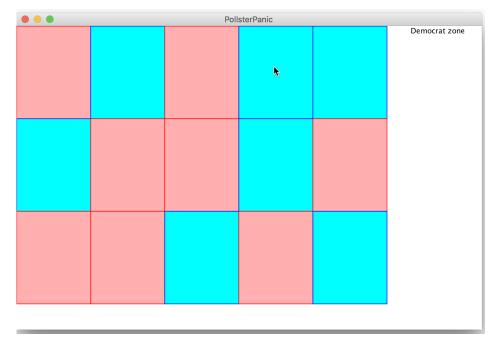
As an example, a map that you create might look like this (note that we used a lighter fill color so that you can more clearly see the color assigned to each zone, but you can use any fill color you want, or no fill color):

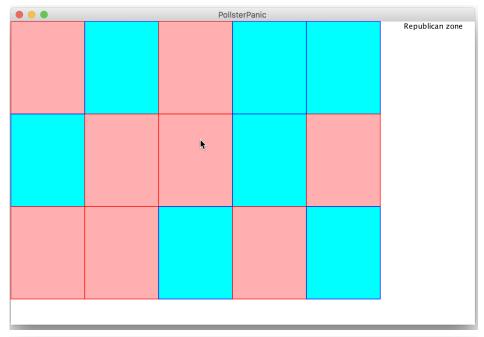


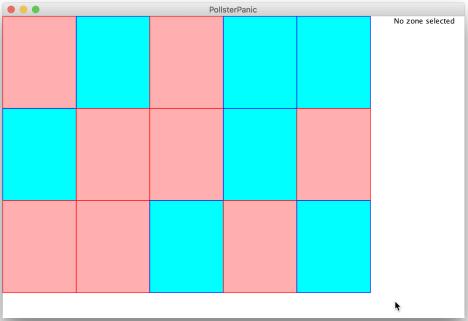
# Part B: Disseminate Deception (15 points)

Now, let's give the campaign manager a way to interpret the results you just made up. When they click on a zone on the map, your program should replace the "Map created!" message with one of three messages:

- "Democrat zone" if a blue zone is selected
- "Republican zone" if a red zone is selected
- "No zone selected" if they clicked somewhere off the grid







# \*\*\* CS 106A MIDTERM SYNTAX REFERENCE \*\*\*

This document lists some of the common methods and syntax that you will use on the exam.

### Karel the Robot (Karel reader Ch. 1-6)

<pre>public class Name extends SuperKarel { }</pre>	
<pre>turnLeft(); turnRight(); turnAround();</pre>	rotates Karel 90° counter-clockwise, clockwise, or 180°
move();	moves Karel forward in current direction by one
	square
<pre>pickBeeper();</pre>	picks up a beeper if present on Karel's corner; else
	error
<pre>putBeeper();</pre>	places a beeper, if present in beeper bag; else error
<pre>frontIsClear(), frontIsBlocked()</pre>	Is there a wall in front of Karel?
leftIsClear(), leftIsBlocked()	Is there a wall to Karel's left (counter-clockwise)?
rightIsClear(), rightIsBlocked()	Is there a wall to Karel's right (clockwise)?
beepersPresent(), noBeepersPresent()	Are there any beepers on Karel's current corner?
beepersInBag(), noBeepersInBag()	Are there any beepers in Karel's beeper bag?
<pre>facingNorth(), notFacingNorth(),</pre>	Is Karel facing north, south, east, or west?
<pre>facingEast(), notFacingEast(),</pre>	
<pre>facingSouth(), notFacingSouth(),</pre>	
<pre>facingWest(), notFacingWest()</pre>	

#### RandomGenerator (A&S 6.1)

<pre>RandomGenerator rg = RandomGenerator.getInstance();</pre>	
<pre>rg.nextBoolean()</pre>	returns a random true/false result;
<pre>rg.nextBoolean(probability)</pre>	pass an optional probability from 0.0 - 1.0, or default to 0.5
<pre>rg.nextColor()</pre>	a randomly chosen Color object
<pre>rg.nextDouble(min, max)</pre>	returns a random real number between <b>min</b> and <b>max</b> , inclusive
<pre>rg.nextInt(min, max)</pre>	returns a random integer between <b>min</b> and <b>max</b> , inclusive

### String (A&S Ch. 8)

<pre>String s = "hello";</pre>	
<pre>s.charAt(i)</pre>	the character in this String at a given index
<pre>s.contains(str)</pre>	true if this String contains the other's characters inside it
<pre>s.endsWith(str)</pre>	true if this String ends with the other's characters
<pre>s.equals(str)</pre>	true if this String is the same as <b>str</b>
<pre>s.equalsIgnoreCase(str)</pre>	true if this String is the same as <b>str</b> , ignoring capitalization
<pre>s.indexOf(str)</pre>	first index in this String where given String begins (-1 if not found)
<pre>s.lastIndexOf(str)</pre>	last index in this String where given String begins (-1 if not found)
<pre>s.length()</pre>	number of characters in this String
<pre>s.replace(s1, s2)</pre>	a new string with all occurrences of s1 changed to s2
<pre>s.startsWith(str)</pre>	true if this String begins with the other's characters
<pre>s.substring(i, j)</pre>	characters in this String from index <i>i</i> (inclusive) to <i>j</i> (exclusive)
<pre>s.substring(i)</pre>	characters in this String from index <i>i</i> (inclusive) to the end of the String
<pre>s.toLowerCase()</pre>	a new String with all lowercase or uppercase letters
<pre>s.toUpperCase()</pre>	

#### Character/char (A&S Ch. 8)

char c = Character.toUpperCase(s.charAt(i));	
Character.isDigit(ch), .isLetter(ch),	methods that accept a char and return boolean values of
<pre>.isLowerCase(ch), .isUpperCase(ch),</pre>	true or false to indicate whether the character is of the
.isWhitespace( <i>ch</i> )	given type
Character.toLowerCase(ch),	accepts a character and returns lower/uppercase version of
.toUpperCase(ch)	it

### Integer/int (A&S Ch. 8)

<pre>int num = Integer.parseInt("106");</pre>	
<pre>Integer.parseInt(String)</pre>	accepts a numerical String and returns the value as an int

#### **Scanner**

closes the scanner

# sc.close() ConsoleProgram

public class <i>Name</i> extends Cons	oleProgram { }
readInt(" <i>prompt"</i> ),	Prompts/reprompts for a valid int or double, and returns it
readDouble(" <i>prompt"</i> )	
readLine(" <i>prompt"</i> );	Prompts/reprompts for a valid String, and returns it
readBoolean(" <i>prompt"</i> ,	Prompts/reprompts for either <i>yesString</i> or <i>noString</i> (case-insensitive).
"yesString", "noString");	Returns true if they enter yesString, false if they enter noString.
<pre>promptUserForFile("prompt",</pre>	Prompts for a filename, re-prompting until input is a file that exists in the
"directory");	given directory. Returns the full file path ("directory/filename").
println(" <i>text"</i> );	Prints the given text to the console, followed by a newline ('\n').
<pre>print("text");</pre>	Prints the given text to the console.

#### GraphicsProgram

<pre>public class Name extends GraphicsProgram { }</pre>	
add(shape), add(shape, x, y);	displays the given graphical shape/object in the window (at x, y)
getElementAt(x, y)	returns graphical object at the given x/y position, if any (else null)
<pre>getHeight(), getWidth()</pre>	the height and width of the graphical window, in pixels
pause( <i>ms</i> );	halts for the given # of milliseconds
remove(shape);	removes the graphical shape/object from window so it will not be seen
<pre>setBackground(color);</pre>	sets canvas background color

#### **Graphical Objects (A&S Ch. 9)**

GRect rect = new GRect(10, 20, 50, 70);

new GImage(" <i>filename</i> ", x, y)	image from the given file, drawn at (x, y)
new GLabel("text", x, y)	text with bottom-left at (x, y)
new GLine(x1, y1, x2, y2)	line between points (x1, y1), (x2, y2)
new $GOval(x, y, w, h)$	largest oval that fits in a box of size w * h with top-left at (x, y)
new GRect $(x, y, w, h)$	rectangle of size w * h with top-left at (x, y)
<pre>obj.getColor(), obj.getFillColor()</pre>	returns the color used to color the shape outline or interior
<pre>obj.getX(), obj.getY(),</pre>	returns the left x, top y coordinates, width, and height of the shape
<pre>obj.getWidth(), obj.getHeight()</pre>	
<pre>obj.move(dx, dy);</pre>	adjusts location by the given amount
<pre>obj.setFilled(boolean);</pre>	whether to fill the shape with color
<pre>obj.setFillColor(Color);</pre>	what color to fill the shape with
<pre>obj.setColor(Color);</pre>	what color to outline the shape with
<pre>obj.setLocation(x, y);</pre>	change the object's x/y position
<pre>obj.setSize(w, h);</pre>	change the object's width and height
<pre>Label.setLabel(String);</pre>	changes the text that a GLabel displays
<pre>Label.getAscent(), Label.getDescent()</pre>	returns a GLabel's ascent or descent from the baseline

#### **Colors**

```
rect.setColor(Color.BLUE);
```

```
Color.BLACK, BLUE, CYAN, GRAY, GREEN, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW Color name = new Color(r, g, b); // red, green, blue from 0-255
```

#### Mouse Events (A&S Ch. 10)