CS106A
Spring 2018

Chris Piech
June 20th, 2018

Extra Practice Problems

Some of these problems are courtesy of Julia Daniel!

Short Problems

1. Read integers in the console from a user until the user enters a blank like. For each print "EVEN" if the integer is even and "ODD" if the integer is odd.

2. Write a double for loop to print out a rectangle of "0"s. The rectangle should have LINES number of strings and COLS number of "0"s per line (assume they are defined as constants). If LINES = 3 and COLS = 4 the following strings should be printed to the console.

```
0000
0000
0000
```

3. Populate an array of size 10 with random booleans.

4. Populate a grid with 3 rows and 2 colums where each value of the grid has a value equal to 2.

5. Populate a grid with 3 rows and 2 colums where each value of the grid has a value equal to its row index plus its col index.

6. Write a method

```
private void printMatrix()
```

that prints the contents of a grid to the screen (in any reasonable format).

7. Open a file named "doubles.txt" with 10 lines, one double per line and store each double in an array.

8. Write a **`GraphicsProgram`** such that when a mouse is clicked, you add a single 10 x 10-pixel rectangle with upper left corner in the place the mouse was clicked.

9. Make a hashmap that associates strings to integers and put in the following entries which associates universities and their endowment (in billions of dollars):

```
"Stanford" -> 22.4
"Berkeley" -> 4.0
"Harvard" -> 35.0
```

Then write a loop that can iterate over all of the key/value pairs and print them out. Any order / format is fine.

10. Put 10 random Integer pairs (both the key and the value are in the range 0, 100) in a hashmap. Then print out all the values whose keys are even.

11. Write a method:

```
String addQuotation(String str)
```

Which takes in an input string and add quotation marks (") to the start and end of the string. Return the result.

11. Use a loop to create a string that contains all of the capital letters in the alphabet, eg: "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

12. Write a method:

```
String makeCamelCase(String input)
```

That, returns the input in camelcase. The input is assumed to be all lowercase. Whenever the input has a space, remove the space and make the next letter uppercase. For example:
If the input was **"this is a test"**, you would return **"thisIsATest"**.

13. Write a ConsoleProgram that has two buttons one which says "yes" and one which says "no". Each time a user clicks on a button you should printout whether or not the "yes" button has been pressed more, the "no" button has been pressed more, or whether they have been pressed an equal number of times.

13. Write a variable type **IdCounter** which has a single public method `getNextId`. `getNextId` should return an integer, and should never return the same integer twice. For example, a user of the **IdCounter** class might do something like this:

```
IdCounter idCounter = new IdCounter ();
int idForChris = idCounter.getNextId();  // can return any id for chris
int idForNick = idCounter.getNextId();   // should return a different id
```

14. Suppose that the integer array **list** has been declared and initialized as follows:

```
private int[] list = { 10, 20, 30, 40, 50 };
```

This statement sets up an array of five elements with the initial values shown below:

list

10	20	30	40	50

Given this array, what is the effect of calling the method

```
mystery(list);
```

if **mystery** is defined as:

```
public void mystery(int[] array) {
   int tmp = array[array.length - 1];
   for (int i = 1; i < array.length; i++) {
      array[i] = array[i - 1];
   }
   array[0] = tmp;
}
```

Work through the method carefully and indicate your answer.

Long Problems

Problem 1: Strings (15 Points)

An *isogram* is a word that contains no repeated letters. For example, the word "computer" is an isogram because each letter in the word appears exactly once, but the word "banana" is not because 'a' and 'n' appear three times each. "Isogram" is itself an isogram, but "isograms" is not because there are two copies of 's'.

There are many long isograms in English; for example, "uncopyrightable" and "computerizably." Your job is to write a method that, given a list of all the words in the English language, finds out what the longest isogram actually is. Write a method

```
private String longestIsogram(ArrayList<String> allWords)
```

that accepts as input a `ArrayList<String>` containing all words in English (stored in lower-case) and returns the longest isogram in the list. If multiple words are tied as the longest isogram, feel free to return any one of them.

```
private String longestIsogram(ArrayList<String> allWords) {
```

Problem 2: Random Numbers (25 Points)

Suppose you want to hold a never-ending birthday party, where every day of the year someone at the party has a birthday. How many people do you need to get together to have such a party?

Your task in this program is to write a program that simulates building a group of people one person at a time. Each person is presumed to have a birthday that is randomly chosen from all possible birthdays. Once it becomes the case that each day of the year, someone in your group has a birthday, your program should print out how many people are in the group.

In writing your solution, you should assume the following:

- There are 366 possible birthdays (this includes February 29).
- All birthdays are equally likely, including February 29.

You might find it useful to represent birthdays as integers between 0 and 365, inclusive.

```
public class NeverendingBirthdayParty extends ConsoleProgram {
```

Problem 3: Arrays (25 points)

A *magic square* is an $n \times n$ grid of numbers with the following properties:

1. Each of the numbers 1, 2, 3, …, n^2 appears exactly once, and
2. The sum of each row and column is the same.

For example, here is a 3 × 3 magic square, which uses the numbers between 1 and 9:

4	9	2
3	5	7
8	1	6

and here is a 5 × 5 magic square, which uses the numbers between 1 and 25:

11	18	25	2	9
10	12	19	21	3
4	6	13	20	22
23	5	7	14	16
17	24	1	8	15

Write a method

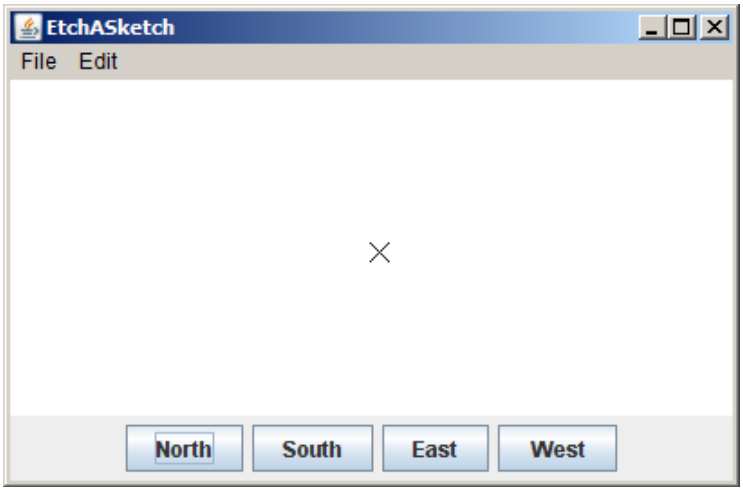
```
private boolean isMagicSquare(int[][] square, int n);
```

that accepts as input a two-dimensional array of integers (which you can assume is of size $n \times n$) and returns whether or not it is a magic square.

Problem 4: Graphics and Interactivity (35 points)

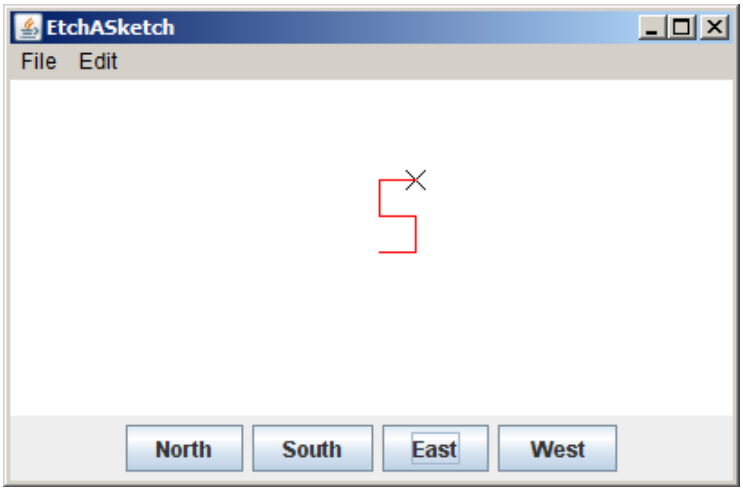
Write a **`GraphicsProgram`** that does the following:

1. Add buttons to the **South** region labeled **`"North"`**, **`"South"`**, **`"East"`**, and **`"West"`**.
2. Create an `X`-shaped cross 10 pixels wide and 10 pixels high.
3. Adds the cross so that its center is at the center of the graphics canvas. Once you have completed these steps, the display should look like this:



4. Implement the actions for the button so that clicking on any of these buttons moves the cross 20 pixels in the specified direction. At the same time, your code should add a **red** **`GLine`** that connects the old and new locations of the pen.

Keep in mind that each button click adds a new **`GLine`** that starts where the previous one left off. The result is therefore a line that charts the path of the cross as it moves in response to the buttons. For example, if you clicked **`East`**, **`North`**, **`West`**, **`North`**, and **`East`** in that order, the screen would show a Stanford "S" like this (note the "S" would be red, even though it does not appear so in the black and white handout):



Problem 4: Arrays (20 points)

In Sudoku, you start with a 9x9 grid of numbers in which some of the cells have been filled in with digits between 1 and 9, as shown in the upper right diagram.

		7	4		3		2	9
	3	8		2		7		4
4	9		5		7	1	3	
8	4		9	7	6	3	1	2
7	2		3			9		6
	6	9		1	4		8	
	1	6		4	2	8		3
2			6		1			5
9		4	8		5	2	6	1

Your job in the puzzle is to fill in each of the empty spaces with a digit between 1 and 9 so that each digit appears exactly once in each row, each column, and each of the smaller 3x3 squares. Each Sudoku puzzle is carefully constructed so that there is only one solution.

Suppose that you wanted to write a program to check whether a proposed solution was in fact correct. Because that task is too hard for an array problem on an exam, your job here is simply to check whether the upper-left 3x3 square contains each of the digits 1 through 9. In the completed example (shown in the bottom right diagram), the 3x3 square in the upper left contains exactly one instance of each digit and is therefore legal.

1	5	7	4	8	3	6	2	9
6	3	8	1	2	9	7	5	4
4	9	2	5	6	7	1	3	8
8	4	5	9	7	6	3	1	2
7	2	1	3	5	8	9	4	6
3	6	9	2	1	4	5	8	7
5	1	6	7	4	2	8	9	3
2	8	3	6	9	1	4	7	5
9	7	4	8	3	5	2	6	1

If, however, you had made a mistake filling in the puzzle and come up with the following

1	5	7
6	3	8
4	9	6

instead, the solution would be invalid because this square contains two instances of the value 6 (and no instances of the value 2).

Your task in this problem is to write a method

```
private boolean checkUpperLeftCorner(int[][] matrix)
```

which looks at the 3x3 square in the upper left corner of the matrix and returns **`true`** if it contains one instance of each of the digits from 1 to 9. If it contains an integer outside of that range or contains duplicated values, **`checkUpperLeftCorner`** should return **`false`**.

In writing your solution, you may assume that the variable passed in as the **`matrix`** parameter has already been initialized as a 9x9 array of **`int`**s. You are also completely free to ignore the values outside of the 3x3 square in the upper left. Those values will presumably be checked by other code in the program that you are not responsible for.

Problem 6: Java programming (30 points)

> *Q: What do you call Enron corporate officers who contributed money to Senators on both the left* ***and*** *the right?*
> *A: Ambidextrous scallywags.*
>
> —Steve Bliss, posting to the Googlewhacking home page

The Google search engine (which was developed here at Stanford by Larry Page and Sergey Brin) has rapidly become the search engine of choice for most users. A few years ago, it also gave rise to a pastime called *Googlewhacking*. The goal of the game is to find a pair of English words so that both appear on exactly one Web page in Google's vast storehouse containing billions of pages. For example, before they were listed on the Googlewhacking home page, there was only one web page that contained both the word *ambidextrous* and the word *scallywags*.

Suppose that you have been given a method

```
public String[] googleSearch(String word)
```

that takes a single word and returns an array of strings containing the URLs of all the pages on which that word appears. For example, if you call

```
googleSearch("scallywags")
```

you would get back a string array that looks something like this:

`http://www.scallywags.ca/`
`http://www.effect.net.au/scallywags/`
`http://www.scallywags1.freeserve.co.uk/`
`http://www.scallywagsbaby.com/`
`http://www.sfsf.com.au/ScallywagsCoaches/`
`http://www.theatlantic.com/unbound/wordgame/wg906.htm`
`http://www.maisemoregardens.co.uk/emsworth.htm`

Each of the strings in this array is the URL for a page that contains the string *scallywags*. If you were to call

```
googleSearch("ambidextrous")
```

you would get a different array with the URLs for all the pages containing *ambidextrous*.

Your job in this problem is to write a method

```
public boolean isGooglewhack(String w1, String w2)
```

that returns `true` if there is exactly one web page containing *both* `w1` and `w2`. It should return `false` in all other cases, which could either mean that the two words never occur together or that they occur together on more than one page. Remember that you have the `googleSearch` method available and therefore do not need to write the code that actually scans the internet (thankfully!).