Assignment 6
CSc 210 Fall 2017
Exercises Due October 9th, 8:00 pm MST
Programs Due October 12th, 8:00 pm MST

Introduction

In class we have been discussing the implementation of classes. On this assignment, you will write several programs that will have you write classes and create objects.

NOTE: THIS ASSIGNMENT HAS TWO DUE DATES.
There are two parts (and two due dates!) to this assignment, exercises done via CodeStepByStep and programs to be turned in via github classroom. The exercises will be due Monday night while the actual programs will be due Thursday night.

You will turn in your code via github classroom. Here is the link to receive your repository: https://classroom.github.com/a/otkPR53Z

For each Java program assigned, you will turn in a .java file for that program. We should be able to run "javac *file*.java" and then "java *file*" on lectura and your program should compile and run successfully.

Your Java programs should adhere to our style guidelines we will give to you. Here is the link: https://www2.cs.arizona.edu/classes/cs210/fall17/StyleGuidelines.pdf

Specification Part I: CodeStepByStep (15 points)

Complete the following exercises on CodeStepByStep by Monday, October 9th, at 8:00 pm.

- Java->classes and objects->Halloween
- Java->classes and objects->BankAccount-toString
- Java->classes and objects->BankAccount-transactionFee
- Java->classes and objects->BankAccount-transfer
- Java->classes and objects->Circle

Specification Part II: Boundary Fill (40 points)

If you've ever used a paint program (such as Microsoft Windows' Paint, or the multi-platform GIMP), you've probably used the 'fill' tool that lets you easily fill a bounded region with a color. One way to implement such a tool is with an algorithm known as Boundary–Fill.

For this assignment, you'll implement a version of Boundary-Fill that uses a queue to remember work that still needs to be done (a queue is a essentially a modified link list, read more here).

Boundary–Fill requires an image, a location within the image, and a new color to be used for filling. To keep things simple, we'll use a 2D array of characters as our image, meaning that locations will be (row,column) pairs and colors will be represented by different characters. Our goal is to replace all locations in the connected region defined by the initial location's character with the new character. Matching characters are part of the same region if one can be reached from the other by following a path that consists of only that same character and that can be traversed without following a diagonal direction (that is, without moving NE, SE, SW, or NW).

For example, consider Figure 1, below. The '@' at (3,1) is not part of the region defined by the other '@' characters. The '+' at (3,2) is part of the region of '+' characters because it's 'north' of the '+' at (4,2). A change of the '+' at (4,2) to another character would create three separate connected regions of '+'.

```
  0123456789     0123456789     0123456789     0123456789
 +----------+   +----------+   +----------+   +----------+
0|@@@@@@@@@=|  0|@@@@@@@@@=|  0|@@@@@@@@@=|  0|@@@@@@@@@=|
1|**@@@**@@=|  1|**@@@**@@=|  1|**@@@**@@=|  1|**@@@**@@=|
2|++@@@**@@=|  2|1+@@@**@@=|  2|..@@@**@@=|  2|..@@@**@@=|
3|+@+@@@@@@=|  3|.@+@@@@@@=|  3|.@+@@@@@@=|  3|.@1@@@@@@=|
4|++++++====|  4|2+++++====|  4|1+++++====|  4|......====|
5|===+++====|  5|===+++====|  5|===+++====|  5|===2++====|
 +----------+   +----------+   +----------+   +----------+
   Figure 1       Figure 2       Figure 3       Figure 4

  ----------     ----------     ----------     ----------
  (3,0)          (2,0)(4,0)     (4,0)          (3,2)(5,3)   <--- Queues
  ----------     ----------     ----------     ----------
```

The version of Boundary–Fill that we'll use for this assignment uses a queue to keep track of the left–most endpoints of horizontal spans of characters that still need to be changed.

Here's how it works. In Figure 1, let's start with location (3,0), which is a '+'. We add it to the queue, shown below the figure. The rest of the algorithm continues so long as there are locations held in the queue. After removing (3,0) from the queue, we change it and all of the '+'s directly connected to it ***only on*** row 3 to the new character ('.'). Here, only (3,0) is changed. Next, we look for connected spans of '+' both directly above and directly below the span of ‘+’ that we just changed. In this case, there’s a span above and a span below. We enqueue the left–most location of those spans (in this case, (2,0) and (4,0), marked with ’1’ and ’2’ for illustration purposes only), as shown in Figure 2. In processing row 2, we find no new spans of ‘+’ (Figure 3), and so no locations are added to the queue, leaving just (4.0) on the queue. Dequeuing and processing (4,0) reveals two new spans (Figure 4). You should be able to see how the algorithm will complete from there.

Your job is to write a complete, well-documented Java program that implements the version of the Boundary– Fill algorithm described above. We know that Java doesn’t provide a dedicated queue class, so we’ll write our own. You are to implement a queue class named MyQueue that represents a queue using any approach you would like (LinkedList, array, etc). However, you are not allowed to use any built in java classes within your Queue class, you must implement the Queue yourself. If you are struggling with the concept of a queue, we will cover queues in section on October 9th. The exact method names, method arguments, etc., your class has are up to you.

You will implement two additional classes: Prog6 and Image. Prog6 should contain your main method, handle reading input into your program, initializing data, and start the Boundary-Fill algorithm. The Image class should contain your representation of your image and the Boundary-Fill algorithm discussed above should be contained here. Your Image class will rely on your MyQueue class for implementing the Boundary-Fill algorithm.

Write your main method in the Prog6 class to accept as command line arguments the name of the file holding the image, the row and column coordinates of the starting point, and the character to be used to fill the area from the command line. For example:

```
java Prog6 pretty.dat 23 7 "*"
```

(The asterisk is in quotes because punctuation symbols on the command line can cause strange behaviors otherwise.) If the user doesn’t give all four parts, print out an error message to stderr and exit. You should also report an error and exit if invalid positions are given. Note the position given in the image file and on the command line are 1-based.

A sample ‘image’ is available in your repositories: prog6sample.dat. The first line has two integers, separated by at least one space. The first number is the number of rows in the image, the second the number of columns. The rest of the lines consist of characters, one image row per line. You may assume files will have valid format. You must check files exist or if there are errors opening the file.

You should create your own images for testing purposes. Feel free to share your testing images with classmates on Piazza.

For each image, location, and fill character specified, your program is to:

- display the original image, location, character at that location, and the new character to be used for filling
- display the final image and the count of the total number of pixels that were modified.

Example:

```
% cat prog7sample.dat
12 20
XX#------XXXXXX----X
XXXXX--XXXX------XXX
XXXX---------XXXXXXX
XXXX---------X######
XX-----XXXXX-XXXX#XX
XX-----XXX-XX-X#####
XXXXXXXXXX------XXXX
====XXXXXX------X==X
XXX=====XXXXXXXX====
===XXXX======XXXXXX=
=XXXX===XXXX=====XX=
======XXXXXXXXXX====
% java Prog6 prog7sample.dat 2 6 "*"
Starting fill at location (2,6).
Original char: -
New char: *
XX#------XXXXXX----X
XXXXX--XXXX------XXX
XXXX---------XXXXXXX
XXXX---------X######
XX-----XXXXX-XXXX#XX
XX-----XXX-XX-X#####
XXXXXXXXXX------XXXX
====XXXXXX------X==X
XXX=====XXXXXXXX====
===XXXX======XXXXXX=
=XXXX===XXXX=====XX=
======XXXXXXXXXX====

Number of pixels modified: 47
```

```
XX#******XXXXXX****X
XXXXX**XXXX******XXX
XXXX*********XXXXXXX
XXXX*********X######
XX*****XXXXX*XXXX#XX
XX*****XXX-XX-X#####
XXXXXXXXXX------XXXX
====XXXXXX------X==X
XXX=====XXXXXXXX====
===XXXX======XXXXXX=
=XXXX===XXXX=====XX=
======XXXXXXXXXX====
```

When you're done with the assignment and are ready to submit it, turn in all of your files via your github repository (Prog6.java, MyQueue.java, Image.java, and any additional classes you may have created). Note: Running **javac *.java; java Prog6** in your repository's root directory should successfully compile and execute your program.

Miscellaneous

In order to receive a grade, your repo must only contain your source java files in the root directory of your repo.

This assignment will be submitted through github classroom. Make sure all of your code you would like to submit is in your repository when the due date arrives.

Note: Your output must match what is defined here in the spec. We will give you a small selection of test cases so you can make sure you have the right format. Do not print extraneous output, you may lose a lot of points (This includes prompts when reading in input!). **These test cases can be found in your repo.**

Remember, do not cheat! Refer to the syllabus and first lecture for more information.