COSC-211: Data Structures
HW2: Stacks, Queues, and Pancakes

Due Thursday, February 21, 11:59pm

Reminder regarding intellectual responsibility: This is an individual assignment, and the work you submit should be your own. Do not look at anyone else's code, and do not show anyone your code (except for me and the course TAs). If you talked with any of your classmates about this assignment, please list their names in a comment at the top of your submission.

1 Introduction

In this assignment, you will implement a stack and a queue from scratch, then use your stack and queue to solve a pancake flipping problem. This is a two-week assignment, but I highly recommend treating it like two one-week assignments: implement your stack and queue (Section 2.1 below) in the first week, and solve the pancake flipping problem (Section 2.2 below) in the second week (but note that all of your files must be submitted together at the end of the assignment).

As always, you are welcome to write code on your own computer, but I will test your code on `remus/romulus` and you are responsible for making sure your code works properly on those servers. **Code that does not compile will not receive credit.**

2 Your Tasks

2.1 Stacks and Queues

Your first job is to implement a `Stack` class and a `Queue` class, both of which will store `int` values. Your classes should meet the following specifications. When class and method names are given, your code must follow *exactly* the name, input and return types, capitalization, etc. given in the specification. This is important to ensure that other programmers who might use your code know exactly how to interact with it.

2.1.1 Stack

1. Must be in a file called `Stack.java`
2. Must contain the following methods:
   - `void push(int a)`
   - `int pop()`
   - `int peek()`
   - `int size()`
   - `boolean isEmpty()`
3. Must be implemented using a `Vector`

2.1.2 Queue

1. Must be in a file called `Queue.java`
2. Must contain the following methods:
   - `void enqueue(int a)`
   - `int dequeue()`
   - `int peek()`
   - `int size()`
   - `boolean isEmpty()`
3. Must be implemented using an array
4. Must not be an "ever expanding" array—that is, you must implement at least one of the modifications discussed in class (shift items upon dequeue, shift upon resize, wraparound queue,...) so that you don't end up with a lot of wasted space at the start of your array.
   In a comment at the top of your `Queue.java` file: What modifications did you use, and why did you make those particular implementation choices?

Make sure you *test your code* at this point before moving on to the next part of the assignment. You probably will want to write some extra code to test your stack and your queue (no need to submit your test code).

2.2 Pancake Flipping

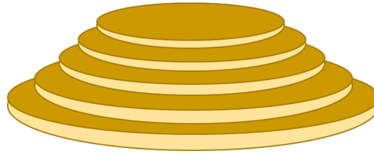
The second part of this assignment is about using stacks and queues to solve a pancake flipping problem.

2.2.1 The Problem

Imagine you wake up on a lazy Saturday morning and decide to make some pancakes on your electric griddle[1]. Unfortunately since you're so used to eating in Val, your pancake technique is a bit rusty and all of your pancakes come out different sizes. For example, you might make a batch of pancakes that looks like this:

You don't want to bother with making a new batch of pancakes to try to get them all the same size, but you would like your pile of pancakes to be more aesthetically pleasing. So you decide to reorder your pancakes so they look more like this:

[1] Which definitely isn't allowed in the dorms, so don't tell ResLife!



with the largest pancake on the bottom and the smallest on top, and all the other pancakes sorted by size in between.

Unfortunately, your plate is rather small so you don't have the space to disassemble the pile of pancakes and put it back together in the right order[2]. So the only way you can reorder the pancakes is by *flipping* some number of pancakes on top of the pile. That is, you insert your spatula below the kth pancake from the top of the pile, then flip the top k pancakes to reverse their order.

For example, you might flip the top four pancakes in your pile as follows:

Original pancakes   Insert spatula below top 4 pancakes   Flip top 4 pancakes

You'll sort your pile of pancakes by repeatedly flipping different numbers of pancakes on top of the pile until your pancakes are in order, top to bottom, from smallest to largest.

2.2.2 A Pancake Flipping Algorithm

Let's start out by assuming that our pancakes have unique sizes (i.e., no two pancakes are exactly the same size), and that they start off labeled in increasing order of size (so in a pile of five pancakes, the smallest pancake is labeled 1 and the largest pancake is labeled 5). In the above example, the original ordering of the pancakes is then $[2\ 5\ 3\ 1\ 4]$ (where the top of the pile is at the right), and the goal is to get the pancakes into the order $[5\ 4\ 3\ 2\ 1]$. In this case, the fastest way to accomplish this is with the following sequence of flips:

1. Flip top 3 pancakes: $[2\ 5\ 4\ 1\ 3]$
2. Flip top 2 pancakes: $[2\ 5\ 4\ 3\ 1]$
3. Flip top 4 pancakes: $[2\ 1\ 3\ 4\ 5]$
4. Flip top 5 pancakes: $[5\ 4\ 3\ 1\ 2]$
5. Flip top 2 pancakes: $[5\ 4\ 3\ 2\ 1]$

But this does not lend itself to any sort of systematic strategy for finding a way of getting *any* pile of *any number* of pancakes sorted.

[2] And you don't want to spread out your pancakes on the floor: when was the last time you *really* cleaned?

One observation that we can make is that the only way to flip a pancake to the *bottom* of the pile is to first flip it to the *top* of the pile, and then flip over the whole pile. We can use this observation to specify an algorithm for how to flip any pile of pancakes, growing a set of sorted pancakes on the bottom of the pile:

1. Flip the largest unsorted pancake to the top of the pile.
2. Flip all of the pancakes that are not yet sorted.
3. Repeat steps 1 and 2 until all pancakes are sorted.

Running this algorithm on our earlier example results in the following sequence of flips:

1. Flip top 4 pancakes: [2 4 1 3 5]
2. Flip top 5 pancakes: [5 3 1 4 2]
3. Flip top 2 pancakes: [5 3 1 2 4]
4. Flip top 4 pancakes: [5 4 2 1 3]
5. Flip top 0 pancakes: [5 4 2 1 3]
6. Flip top 3 pancakes: [5 4 3 1 2]
7. Flip top 0 pancakes: [5 4 3 1 2]
8. Flip top 2 pancakes: [5 4 3 2 1]

This takes us 8 flips (ok, 6 if you don't count flipping 0 pancakes as a "real" flip), which is more than we needed above. But now we have a pancake-flipping strategy that will work for any starting configuration of pancakes.

2.2.3 Tasks

Begin by downloading a file called `Flipper.java`, which contains some scaffolding code:

```
$ wget -nv -i https://tinyurl.com/cosc211hw2flip
```

`Flipper` is set up to take an integer as a command line argument. You will run the program using a command like:

```
$ java Flipper 6
```

where the integer argument is the number of pancakes you want in your pile. I have provided a method called `initialize()` that will initialize the stack of pancakes to contain the specified number of pancakes in a random order (the pile of pancakes is represented as a stack of integers). Right now all `Flipper` does is create a new `Stack` object and call the `initialize` method.

Your job is to fill in the `Flipper` class so that it successfully performs a sequence of flips to convert an unsorted pile of pancakes into a sorted pile of pancakes. Specifically, you must complete the following tasks:

1. Fill in the `flip()` method, which takes as input a stack of pancakes and an int `pos`. This method should flip the top `pos` pancakes on the stack; at the end of the method the pancakes should be contained *in the same stack as before*, just in a different order. For example, if the stack `s` originally contains [5 3 1 2 4] and you call `flip(s,4)`, then when the method completes `s` should contain [5 4 2 1 3].

   Your `flip` method should use some combination of stacks and/or queues as auxiliary data structures to help you perform the flip operation. Think carefully about what data structure(s) might be most appropriate for different parts of this task. There are plenty of ways to accomplish this that don't use stacks and queues, but one of our goals in this assignment is to practice using stacks and queues, so please don't use an alternative approach!

2. Fill in the `flipAllPancakes()` method, which takes as input a stack of pancakes in no particular order. In this method you should implement the algorithm described in Section 2.2.2. Print out the stack of pancakes after *every* flip operation (you likely will want to write `print()` method in your `Stack` class).

You may find that it would help to have additional or different variations on the `flip` method (for example, perhaps you want to specify which *pancake* should be flipped to the top of the pile, rather than which *position*). Feel free to write any helper methods you want, adding comments to your code to make it clear what these extra methods are intended to do.

2.2.4 One Last Task

Modify the `Stack` class so that it is generic. Make sure you update your `Flipper` code so that it uses your modified `Stack` class properly.

3 Submit your work

Make sure you *test your code* thoroughly before you submit it. **Code that does not compile will not receive credit.**

Submit all of your java files using either the submission web site or (from `remus/romulus`) the `cssubmit` command:

```
cssubmit *.java
```

and use the numeric menu to select the correct course and assignment. Please submit *all of your files in a single submission*.

This assignment is due on Thursday, February 21, 11:59pm.