Layla Oesper

This assignment is due by **10pm on Wednesday October 18** and is worth 24 points.

1 Goals

The goal of this assignment is to learn about and get practice with Stacks and Queues. You will also get a chance to see how the complexity of these data structures translates into processing time.

2 Your Assignment

This is a partner assignment (unless you were not assigned a partner). You will be working with a new partner than the previous partner assignments. You will complete the following tasks: (1) You will implement a Stack using a Linked List. (2) You will implement a Queue using a circular array. (3) You will use a program that I will provide to look at how the running time of your implementations changes as the number of items in the Stack/Queue grows. (4) You will write two short methods that utilize your Stack and Queue implementations to complete some small tasks.

Implementing a Stack with a Linked List

You will complete the code in the provided file `LLStack.java` that implements a Stack using a linked list. Notice that this class implements that Stack ADT provided in the interface `Stack.java`. You implementation will be a generic implementation that works with any datatype that is an Object. This is the first time we are doing an implementation like this. Section 2.4 in your book shows how to implement a generic ArrayList and has some good suggestions if you are struggling - in particular, look at the program style box on page 71 if you are seeing weird looking compiler messages. You will need to complete the methods in the following table. Make sure that you implement these methods exactly as described (do not change the parameters, method names or return types). Also, make sure that all of your methods keep the instance variables up to date.

MethodName(Parameters)	**Return type**	**description**
`LLStack()`	`LLStack`	Creates an empty Stack object.
`isEmpty()`	`boolean`	Returns true if the stack is empty; false otherwise.
`size()`	`int`	Returns the number of items currently stored in the Stack.
`push(E item)`	`void`	Add item to the top of the Stack. Note, this should work with any type E.
`pop()`	`E`	Removes and returns the item on the top of the stack. If the stack is empty a `NoSuchElementException` is thrown.
`peek()`	`E`	Returns the item on the top of the stack, but does not modify the stack. If the stack is empty a `NoSuchElementException` is thrown.
`toString()`	`String`	Returns a string representation of the stack. See below for a specific example of the format the string should take.

Your `toString()` method should return a string in **exactly** the following format (three examples shown with a stack of integers).

```
top [ ] bottom       (empty stack)
top [ 1 ] bottom     (stack with one element)
top [ 5 2 8 ] bottom (stack with 3 elements)
```

In particular, use the `toString()` method for whatever objects are stored in the Stack to get the item representations.

Implementing a Queue with a Circular Array

You will complete the code in the provided file `ArrayQueue.java` that implements a Queue using a circular array. Notice that this class implements that Queue ADT provided in the interface `Queue.java`. You implementation will be a generic implementation that works with any datatype that is an Object. You will need to complete the methods in the following table. Make sure that you implement these methods exactly as described (do not change the parameters, method names or return types). Also, make sure that all of your methods keep the instance variables up to date.

MethodName(Parameters)	**Return type**	**description**
`ArrayQueue()`	`ArrayQueue`	Creates an empty Stack object. I've already provided this method. Do not change it.
`isEmpty()`	`boolean`	Returns true if the queue is empty; false otherwise.
`size()`	`int`	Returns the number of items currently stored in the Queue.
`enqueue(E item)`	`void`	Add item to the rear of the Queue. Note, this should work with any type E.
`dequeue()`	`E`	Removes and returns the item on the front of the queue. If the queue is empty a `NoSuchElementException` is thrown.
`peek()`	`E`	Returns the item at the front of the Queue, but does not modify the queue. If the queue is empty a `NoSuchElementException` is thrown.
`toString()`	`String`	Returns a string representation of the queue. See below for a specific example of the format the string should take.
`resizeArray()`	`void`	This should be a private method that doubles the size of the underlying array. You will need to call this method in your enqueue method.

Your `toString()` method should return a string in **exactly** the following format (two examples shown with a queue of integers).

```
front: 4 rear: 5
front [ 1 ] rear      (queue with one element)
```

```
front: 6 rear: 0
front [ 5 2 8 ] rear  (queue with 3 elements wrapped)
```

In particular, use the `toString()` method for whatever objects are stored in the Queue to get the item representations.

Analyzing Runtime of Your Implementations

I have provided you with the file `StackQueueTimeTest.java` that tests your Stack and Queue implementations and measures their runtime (in terms of clock time while running). You should test your implementations frequently as you code them in their individual main() functions, but when you finish both you can use StackQueueTimeTest to ensure your implementations comply with the Stack and Queue interfaces and are efficient. You can compile the test program in the normal way: `javac StackQueueTimeTest.java`. Java will automatically look for the rest of the files it needs and also compile them (as long as they are all in the same directory), including the Stack interface and both of your implementations, so be sure all of those files are in the same folder with `StackQueueTimeTest.java`.

The command `java StackQueueTimeTest` will run the test by executing a large number of pushes and pops on both of your implementations and output the total runtime of each for different input sizes. Note, this code should not run instantaneously, but it should definitely finish in under a minute. If it is taking longer for your implementations, you are likely doing something that is not efficient in your code. There is a TON of noise in these actual-time numbers, and Java does all sorts of optimizations that make it hard to compare implementations, but you should see a general trend of runtime growing linearly with N when you use big enough N's to yield runtimes of a second or more. Linear growth means that increasing N by some factor should increase runtime by the same factor, e.g. doubling N should double the runtime, or tripling N should triple the runtime.

For example, on my computer, a test with an N of 50,000,000 (fifty-million) takes .6 seconds for LLStack and .5 seconds for ArrayQueue. A test with an N of 100,000,000 (one-hundred-million), or twice as big as the previous N, takes 0.95s for LLStack and 1.05s for ArrayQueue, or twice as long. A test where N is again doubled to be 200,000,000 (two-hundred-million), takes 2s for LLStack and ArrayQueue, or again twice as long as the previous.

This is exactly what you would expect given linear growth in runtime. Why do we expect doubling the number of operations (N) to double the runtime? **Turn in a text file with your code that answers the previous question.**

*** The above tests are meaningless with small N's! You MUST use an N big enough to generate runtimes of at least .1 seconds to get meaningful results, and results will be more consistent for even bigger N's. ***

Using Your Implementations

Lastly, you will turn in a file `StackQueueSolver.java` that implements a few methods that use your stack an queue implementations. In particular, this file will contain the following 3 **static** methods. Note: This class is just a container for these methods, so it does not need any instance variables or constructors.

MethodName(Parameters)	Returns	description
`reverseStack(Stack s)`	`Stack s`	Returns a new Stack that contains the items in `s` in the reverse order. For an added challenge, make sure that `s` is unchanged at the end of the method.
`testStackReverse(int m)`	`void`	This method should first create a stack containing the numbers $1, 2, \ldots, m$, with m on top. Use the `toString()` method to print the contents of that stack. Then call the `reverseStack` method on it and use the `toString()` method to print the contents of that stack.
`lastCustomer(int m, int n)`	`int`	This methods will simulate some quirky behavior of a line for food in the LDC. Originally there are m students in line, number them $1, 2, \ldots, m$. The person at the front of the line is then told to go to the back of the line, doing this n times. The person at the front of the line, then gets their food. This process repeats until the line is empty (assuming no one else enters the line). The method `lastCustomer` should return the number of the customer that is served LAST. You may assume that $m > 0$ and $1 \leq n \leq m$. Use a queue to compute your solution.

3 Submission and Grading

You'll submit all your files to Moodle as a zipped file. One one partner from each pair needs to submit the assignment. Specifically, put these files into a directory named `[your_last_name_your_partner's_last_name]HW8`, zip this directory, upload it to Moodle. For example, if my partner was Schiller my directory would be named `OesperSchillerHW8` and the resulting file would be `OesperSchillerHW8.zip`.

Grading

This assignment is worth 24 points. Below is a partial list of the things that we'll look for.

- Do you implement all of the requested methods as they are described? We'll test this out by running your various methods on different test cases.
- How efficient is your code? More efficient code will get a higher grade.
- Do your classes exhibit good organization and commenting? Don't forget to follow the Style Guidelines on Moodle.

Start early, ask lots of questions, and have fun! Layla, the lab assistants, and the prefect are all here to help you succeed - don't hesitate to ask for help if you're struggling![1]

[1] This assignment modified from `http://www.cs.cmu.edu/ mrmiller/15-121/Homework/hw8/hw8.html`, and an assignment by Sherri Goings. Thanks for sharing!