Project #2

Daniel Williamson

10/26/16

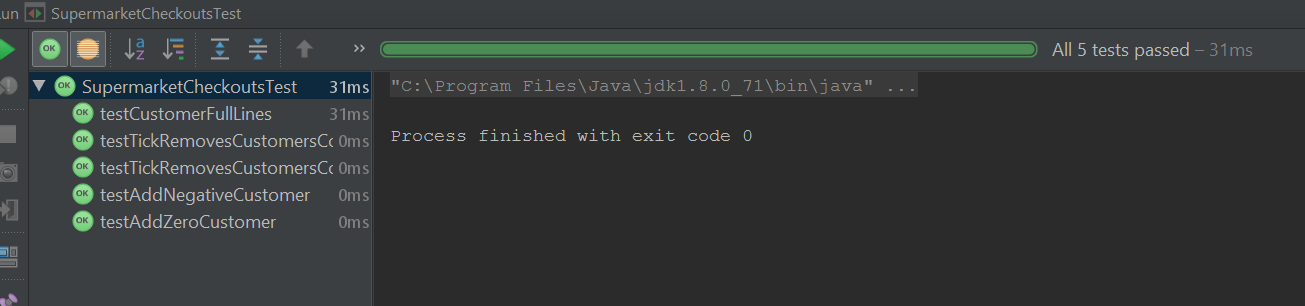
SuperMarket: O(N)

Approach:

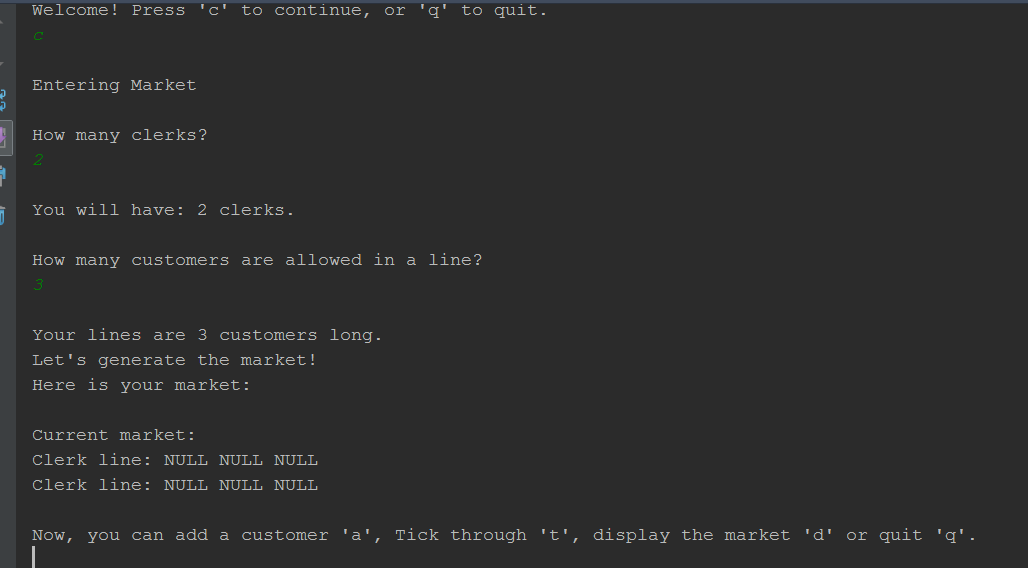
My approach, just like many of projects, starts off with doing some research. I did some reading up in the book and internet to help me out. I also reached out to piers, namely Stephen White, whom helped a ton throughout this entire project. After brushing up on queues and having a discussion with piers I began by creating a customer. The basic individual that makes up a supermarket. After defining what a customer was, I moved onto the clerk. The person that interacts with the customer, by adding and removing the items the customer holds. After getting that to work, I moved onto creating what the supermarket does. I used an arraylist because I needed it to hold objects, the clerks, whom hold the customers. I did away with the simple queue skeleton code, simply because, in my discussions with Stephen it was clear it was a little difficult to understand. Once I had a super market created, I did some testing to see whether it would work or not, buy creating clerks and adding customers into them. The most difficult part, I felt, about this process was getting the customers to fall into a clerks line that had the least amount of customers, regardless of if they had a lot of items. But, I made it through with a lot of trial and error. Finally, I ran the tests. At first I failed a few, but eventually got all the bugs out of them and was able to pass all tests. I then proceeded onto the supermarket app.

The supermarket app was thankfully the easiest of them all. It was just spitting out some print statements and asking for some user input. It was easy because I added toStrings into each customer, clerk and the supermarket. Doing this, made life so much easier. Especially when it came down to printing out the customers that are in the lines.

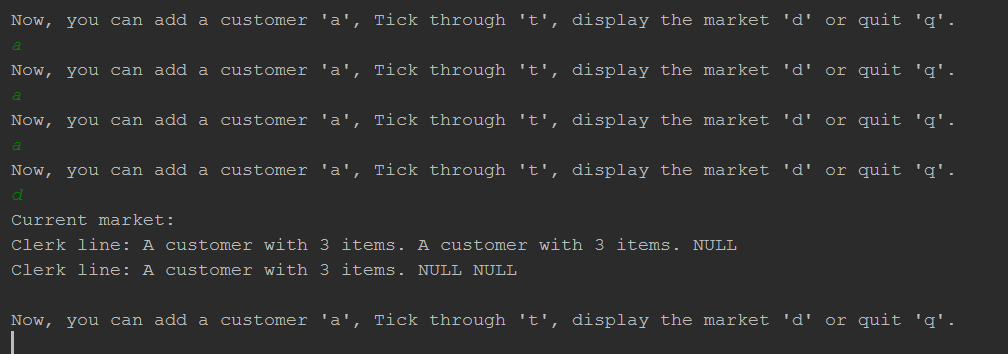
Here is a photo of my supermarket tests all passing:



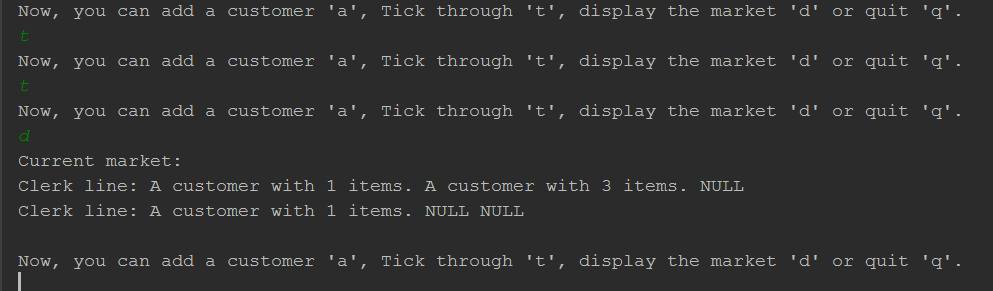
Here is a photo of the beginning of the supermarket app. I create 2 clerks/lines and they can each hold up to 3 customers. It also prints out the market, null shows that there are no customers in the market.



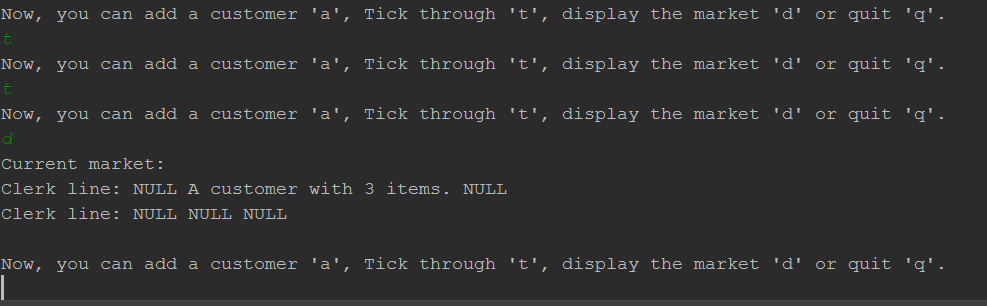
Now, let’s add some customers. I add 3 customers and then display the market. And the number of items each customer holds is randomly generated. They just all happen to be 3. They also move into the smallest line, when available.



Now, we do some ticks. Each tick removes an item from a customer. We do 2 ticks in the photo and we can now see that the first two customers have just 1 item.



In this photo, we do two more ticks, so that we remove the customer after hitting 0 items.

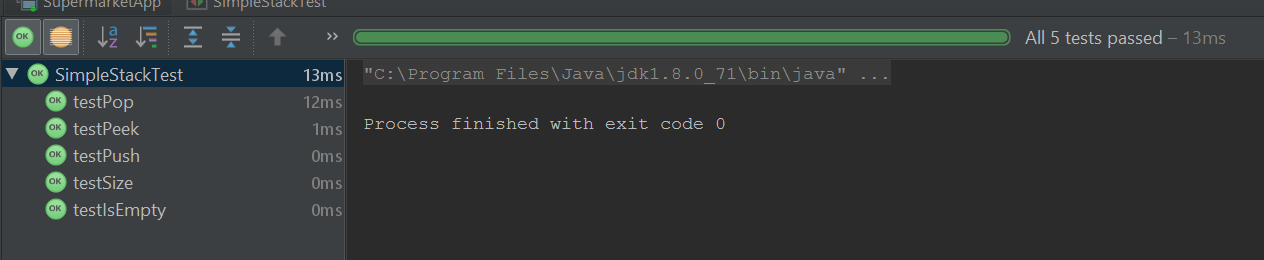


Stack: O(1)

Approach:

Like, previously stated, I began by reading up and researching how a stack functions. It’s a rather easy concept. I also, had discussions with Stephen White about completing this part of the project. Once I had a better grasp on how it works. I began coding. With much discussion with Stephen I went ahead and created a Stack of nodes. This creates a node and links them together in a stack formation. It allows us to remove a node and add nodes. And we use this class in conjunction with simple stack to pass all the tests. Since we pretty much created the ability to remove and add nodes, we add some more magic to get them to pop and push. The most difficult part about this section was wiring the nodes when pushing and popping.

Here is a photo of passing all the tests in the simple stack test.



Elimination: O(N)

Approach:

So, just like all the other parts of this project, I began with doing some research online and speaking with my peers. Stephen White was once again, my go to guy for discussion. Once I had a much better understanding of how a circularly linked list works, I began by making a circular linked list class. This class, essentially holds all the information needed for being a circularly linked node. Keeping track of the first and last nodes. While maintaining the pointers to the current and previous nodes. This was absolutely the most difficult part of the project. I had the most difficulties with this, and worked heavily with Stephen. Once I got that bugger working, the rest was simpler. An issue I ran into, was getting the count to wrap around the list. Since it is circular, if I had three nodes in a list, and asked for the 4th node. I wouldn’t receive the 1st node. Which would be the 4th. I would receive whatever node was last. Which is not correct. So, as you can see by the picture, I did not pass all 5 tests. The issue I am seeing, it is skipping the first node while counting through. I’ve tried multiple things, but nothing seems to be working.

