Assignment 4: Queues

Part 2: Shoppers

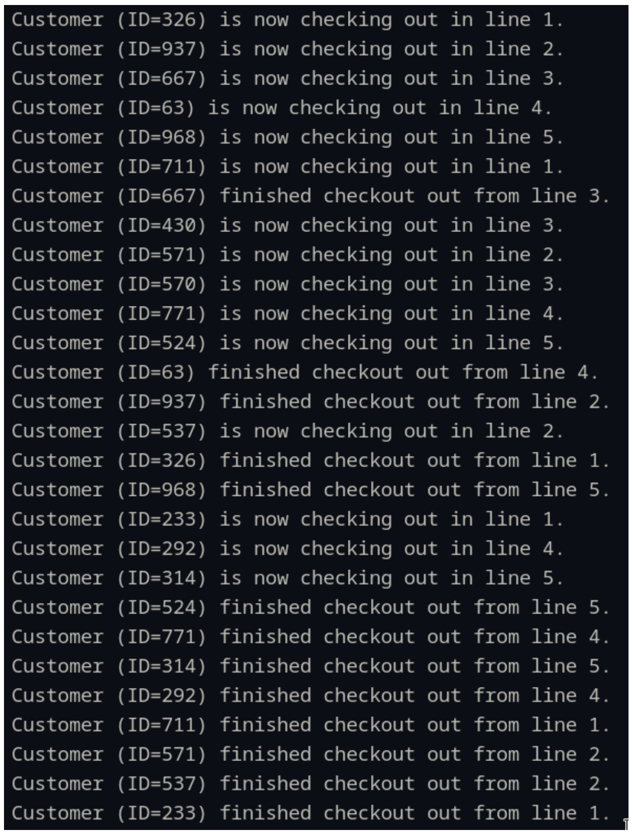
CS3305/W01 Data Structures

Casey Hampson

September 9, 2024

Program Output

The output is a little hard to parse (at least for me), so I'll describe the basic idea. What we see first is the initial five customers added to each line, then one more goes to 1, as it's the first available. Then someone checks out from line 3, making it the shortest line now, so the next customer goes to that line. Now 2-5 are the shortest, so it goes in order, and so on.



```
Customer (ID=326) is now checking out in line 1.
Customer (ID=937) is now checking out in line 2.
Customer (ID=667) is now checking out in line 3.
Customer (ID=63) is now checking out in line 4.
Customer (ID=968) is now checking out in line 5.
Customer (ID=711) is now checking out in line 1.
Customer (ID=667) finished checkout out from line 3.
Customer (ID=430) is now checking out in line 3.
Customer (ID=571) is now checking out in line 2.
Customer (ID=570) is now checking out in line 3.
Customer (ID=771) is now checking out in line 4.
Customer (ID=524) is now checking out in line 5.
Customer (ID=63) finished checkout out from line 4.
Customer (ID=937) finished checkout out from line 2.
Customer (ID=537) is now checking out in line 2.
Customer (ID=326) finished checkout out from line 1.
Customer (ID=968) finished checkout out from line 5.
Customer (ID=233) is now checking out in line 1.
Customer (ID=292) is now checking out in line 4.
Customer (ID=314) is now checking out in line 5.
Customer (ID=524) finished checkout out from line 5.
Customer (ID=771) finished checkout out from line 4.
Customer (ID=314) finished checkout out from line 5.
Customer (ID=292) finished checkout out from line 4.
Customer (ID=711) finished checkout out from line 1.
Customer (ID=571) finished checkout out from line 2.
Customer (ID=537) finished checkout out from line 2.
Customer (ID=233) finished checkout out from line 1.
```

Source Code

```java
// Name: Casey Hampson
// Class: CS 3305/W01
// Term: Fall 2024
// Instructor: Sharon Perry
// Assignment: 04-Part-2-Shoppers


import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Random;


// stores a random id to differentiate between customers
class Customer {
    final private static Random random = new Random();
    public int id;

    public Customer() {
        this.id = random.nextInt(0, 1000);
    }
}


class CheckoutLine extends Thread {
    final private Queue<Customer> line;
    final private static Random random = new Random();

    private static int counter = 1;
    final private int id;

    // initialize the queue and the counter
    public CheckoutLine() {
        this.line = new LinkedList<>();
        this.id = counter++;
    }

    public int Size()  { return this.line.size(); }
    public int GetId() { return this.id; }

    // adds a customer to the queue to check out
    public void AddCustomer(Customer customer) {
        System.out.printf(
            "Customer (ID=%d) is now checking out in line %d.\n",
            customer.id, this.id
        );
        this.line.add(customer);
    }



    // the thread will just randomly sleep from between 2 to 10 seconds,
    // then remove the first customer in the queue (the one currently checking out)
    // until there is no one left in the line
    @Override
    public void run() {
        try {
            while(!this.line.isEmpty()) {
                Thread.sleep(random.nextInt(2000,10000));
                Customer customer = this.line.remove();
                System.out.printf(
                    "Customer (ID=%d) finished checkout out from line %d.\n",
                    customer.id, this.id
                );
            }
        } catch (InterruptedException e) {
            throw new RuntimeException("ERROR (InterruptedException): in CheckoutLine.Run(): ", e);
        }
    }
}




class Store {
    final private int num_checkout_lines = 5;
    final private ArrayList<CheckoutLine> checkout_lines;
    final private Random random = new Random();

    // initialize the checkout lines,
    // and as in the problem description, enqueue 1 customer in each to start
    public Store() {
        this.checkout_lines = new ArrayList<>();
        for (int i=0; i<this.num_checkout_lines; i++) this.checkout_lines.add(new CheckoutLine());
        this.checkout_lines.forEach(line -> line.AddCustomer(new Customer()));
    }

    // grabs the current shortest line (or the first one that is available,
    // if there are more than one shortest line)
    private int GetShortestLine() {
        int shortest_line_idx = 0;
        for (int i=1; i<this.num_checkout_lines; i++) {
            if (this.checkout_lines.get(i).Size() < this.checkout_lines.get(shortest_line_idx).Size())
                shortest_line_idx = i;
        }

        return shortest_line_idx;
    }

    // starts all the threads for the lines, then continually adds customers
    // to the shortest line (up to <num_customers>)
    // the time bounds are roughly a fifth of that of the individual lines
    // to allow roughly even flow of customers into the lines
    public void Start(int num_customers) {
        this.checkout_lines.forEach(line -> line.start());
        try {
            while (num_customers > 0) {
                Thread.sleep(random.nextInt(500,2000));
                this.checkout_lines.get(this.GetShortestLine()).AddCustomer(new Customer());
                num_customers--;
            }
        } catch (InterruptedException e) {
            throw new RuntimeException("ERROR (InterruptedException): in Store.Start(): ", e);
        }
    }



}


public class P2 {
    public static void main(String[] args) {
        Store store = new Store();
        store.Start(10);
    }
}
```