Assignment 5 – Part 2: Shoppers

Ryan Brinson

9/22/2023

Output:

```
rbrinson2@DESKTOP-U8KJ4OP:~/Documents/CS3305/Assignments/A5$ java A5P2
Customer 1 enters line 1
Customer 2 enters line 2
Customer 3 enters line 3
Customer 4 enters line 4
Customer 5 enters line 5
Customer 6 enters line 4
Line 1: 1
Line 2: 2
Line 3: 3
Line 4: 4 6
Line 5: 5
Customer 3 leaves line 3
Line 1: 1
Line 2: 2
Line 3:
Line 4: 4 6
Line 5: 5
Customer 2 leaves line 2
Line 1: 1
Line 2:
Line 3:
Line 4: 4 6
Line 5: 5
Customer 7 enters line 2
Line 1: 1
Line 2: 7
Line 3:
Line 4: 4 6
Line 5: 5
Customer 4 leaves line 4
Line 1: 1
Line 2: 7
Line 3:
Line 4: 6
Line 5: 5
Customer 5 leaves line 5
Line 1: 1
Line 2: 7
Line 3:
Line 4: 6
Line 5:
Customer 8 enters line 3
Line 1: 1
Line 2: 7
Line 3: 8
Line 4: 6
Line 5:
```

Code:

```
// Name: Ryan Brinson
// Class: CS 3305 W04
// Term: Spring 2023
//  Instructor:  Carla McManus
//  Assignment:  05-Part-2-Shoppers

import java.util.LinkedList;
import java.util.Queue;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class A5P2 {
    private static CheckoutLines store = new CheckoutLines();

    public static void main(String[] args) {

        // Creation of threads that will add and remove customers
        ExecutorService executor = Executors.newFixedThreadPool(2);
        // Thread 1 adds customers
        executor.execute(new EnterCustomer());
        // Thread 2 removes customers
        executor.execute(new LeaveCustomer());

        // Shutdown the Executor
        executor.shutdown();
    }


    // ----- Thread 1: Enter Customer -----
    // Class who's only task is to add customers at random times
    private static class EnterCustomer implements Runnable{
        public void run(){
            try {
                for (int i = 0; i < 10; i++){
                    // Envoke the add customer method from the Checout class
                    store.addCustomer();
                    store.printCheckoutLine();
                    // Put the thread to sleep at some randome time
                    // between 0 and 5 seconds
                    Thread.sleep((int)(Math.random() * 5000));
                }
```

```
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }

        }
    }

    // ----- Thread 2: Leave Customer -----
    // Class that's only task is to remove customers at random times
    private static class LeaveCustomer implements Runnable{
        public void run(){
            try {
                // There are a total of 15 customers in this program
                // because of that this loop runs 15 times
                for (int i = 0; i < 15; i++){
                    // Envoke the remove customer from out Checkout class
                    store.removeCustomer();
                    store.printCheckoutLine();
                    // Put the thread to sleep at some randome time
                    // between 0 and 5 seconds
                    Thread.sleep((int)(Math.random() * 5000));
                }

            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }

        }
    }

}


// ---- Checkout Lines Class -----
class CheckoutLines {
    private Queue<Integer>[] checkOutLines;
    private static Lock lock = new ReentrantLock();
    private Integer person;

    // Cechkout Constructor
    CheckoutLines(){
        // Initialize the array
        checkOutLines = new LinkedList[5];
        person = 0;
```

```
        for (int line = 0; line < 5; line++){
            // Initialize each checkout line as a linked list
            checkOutLines[line] = new LinkedList<Integer>();
        }

        for (int line = 0; line < 5; line++){
            // Queue a new person into each line
            checkOutLines[line].offer(++person);
            System.out.printf("Customer %d enters line %d\n", person, line + 1);
        }
    }


    // Print Line
    private void printLine(Queue<Integer> l, Integer line){
        System.out.printf("Line %d: ", line + 1);

        // Using the for each to print each person in the given line
        for (Integer person : l) {
            System.out.printf("%d ", person);
        }
        System.out.println();
    }

    // Print Checkout Line
    public void printCheckoutLine (){

        // Iterorates through each of the 5 lines to print their contents
        for (int line = 0; line < 5; line++){
            printLine(checkOutLines[line], line);
        }
    }

    // Pick Line
    public Integer pickLine(){
        // set the min and the max size to the first queue
        int min = checkOutLines[0].size();
        int max = checkOutLines[0].size();
        int indexMin = 0;

        // Iterate through the rest of the lines
        for (int line = 1; line < 5; line++){
            // If the ith line is shorter, set that to the min
            if (checkOutLines[line].size() < min) {
```

```
                min = checkOutLines[line].size();
                indexMin = line;
            }
            // if the ith line is larger, set that to the max
            if (checkOutLines[line].size() > max)
                max = checkOutLines[line].size();
        }

        // if min and max are equal, all the lines are the same length
        if (min == max){
            // The customer chooses a random line
            return (int)(Math.random() * 5);
        }
        // Else the customer chooses the shortest line
        else return indexMin;
    }


    // ---- Threaded Tasks -----

    // Remvove Customer
    public void removeCustomer(){
        // Lock the method from being called too soon
        lock.lock();

        try {
            // Choose a random line
            int line = (int)(Math.random() * 5);

            // If that random line is empty, choose another one
            while (checkOutLines[line].isEmpty()){
                line = (int)(Math.random() * 5);
            }
            // Update the queue and print the action
            System.out.printf("Customer %d leaves line %d\n",
checkOutLines[line].poll(), line + 1);

        } finally {lock.unlock();} // Unlock the method so it can me used again

    }

    // Add Customer
    public void addCustomer(){

        // Lock to prevent the thread from calling it again too soon
```

```
        lock.lock();

        try {
            // Call the method for picking lines
            int line = pickLine();

            // Use the Offer method to add the person to the line
            checkOutLines[line].offer(++person);

            // Print the action taken
            System.out.printf("Customer %d enters line %d\n", person, line + 1);

        } finally {lock.unlock();} // Unlock the thread so it can be used again

    }
}
```