Chris W Jones

CMSC350 – 7380

27 June 2015

# Project 1 Documentation

## Analysis

When I read through the requirements for this project, I started by writing down all of the numbers and how they related. For example, the scenario takes place over 10 minutes. There are 5 gates. Each gate has a limit of people it can process in a minute. There is a line that feeds the gates. 20 people are in line at the outset. 10 people are added to the line each after each minute.

Once I had a list of the numbers and their associations, I decided how I could turn each of these into a programming structure. For some, like gates and the security line, I decided that they needed to be turned into objects. For others, like the time, I decided that a simple variable in a loop would suffice. And still for others like people, I decided that a simple string.

From there, I sketched out pseudocode showing an outline of the program’s flow and expected output. In the intervening hours of coding, I was surprised how much time and effort my initial plan had saved. Having a plan was easily the best thing I did in writing this project.

## Design

After deciding which objects (classes) that I would need to write, I decided what each class would need to do. I started with the most important objects to this project: the gates. The way I decided to implement the Gate was as an ArrayDeque<String>. I chose to use an ArrayDeque to hold the passengers processed because it is quick to add elements taking only O(1+min(index, numberOfElements)) and when I need to print out the people processed it takes only O(1) to get a given element. I also chose the ArrayDeque because that was the point of our lesson this week.

The next class that I created was SecurityLine since that was the other big chunk of this problem. I again decided to go with an ArrayDeque<String>. The performance benefits are the same. I chose to use a String with the ArrayDeque because the program’s whole goal is to output either on a GUI or a command line and as stated by Doug McIlroy, who created Unix pipes, “Write programs to handle text streams, because that is a universal interface” (Salus, 1994).

Finally, I needed a way to generate people to fill all of these queues. Since each time that I would be adding people to the SecurityLine it would be in an increment of 10, I really just needed an array of strings filled with 10 people’s names. I decided to represent a person as a string instead of another object because I realized that I didn’t need to perform any sort of actions to the passengers. I just needed them that have a name.

## Implementation details

I’m going to discuss my implementation decisions in the same order as my design decisions since that was the way I wrote them all.

Each Gate needed a way to have people sent to it, have a limit of some sort, report if it could accept another passenger and a way to clear the passengers out of the gate. When a passenger is sent to the Gate, it checks that it hasn’t processed all of the passengers it can for that minute. If it can still accept people, the person is pushed onto the end of the ArrayDeque, otherwise, null is returned. In the main program loop, if a Gate is full (returns null), the next Gate is tried in the Gate array until all of the Gates have processed all the people they can for the minute.

The SecurityLine constructor is defined twice. It can either accept no arguments or an array of Gates. Without any arguments, the Gates used are the ones specified in the assignment so one Gate that can process 1 person/min, two Gates that can process 2 people/min, and two Gates that can process 3 people/min.

The main method used in the SecurityLine class is the tick() method. It accepts the current minute in the simulation. This in turn will print out the beginning summary, add 10 people to the line, clear the gates from the previous minute, process the passengers through the gates, and print out the end of minute summary.

Since all of the gates are in an array, it is easy to loop through the array with a for loop and call a method on each gate. I would have loved to use a function from a functional programming language (like Clojure): map. With a map function, I wouldn’t have needed a for loop since map is used to apply a function over a group of objects.

The final thing I wrote was a person generator. I need each person to have some sort of name. I decided to have a static function that returns a String[] array of 10 people’s names. I wrote a list of 10 names. The method then picks a random upper-case letter to use as a last name identifier so we get a name like “Hilary D” or “John W”. The last letter is the same for each group of 10 people to easily differentiate groups of people added to the SecurityLine. Before returning the 10 names, the array is shuffled so that it looks like a random group of people.

This program accepts no user input so I have included no input validation. The only error checking is to make sure that when adding people to a Gate, there are actually people in line to add to the Gate. I found this bug in an edge case. Starting out with 20 people in line and adding 10 per minute, it takes 20 minutes to catch up so that the Gates process all of the people added to line.

## User Interface

I purposefully included no user interface. The project doesn’t need one. This project is a simulation of a security line. I could think of nothing that a user would need to input to it. I could think of no one who would run this program without having even a trivial understanding of Java. It was for those reasons that I decided to simply output all of the simulation details to System.out.

## Testing

Testing of the application was decidedly short. Since there is no user input, I just checked the output of all of the functions. I considered using JUnit to test all of my classes but decided against it since it would add increased complexity to my code by way of the testing classes and including the library in my build path.

In order to test the program, I simply ran it and watched the output. The first few times I ran it, I discovered several errors in my program. Most of the issues related to my understanding of the ArrayDeque class in Java. I incorrectly thought that when I called push() and pop() on it, they operated in a FIFO order. Since that class doesn’t, I had to change several spots in my code where I was using those operations to make it behave in that manner.

After that, I compared what I was outputting to the requirements of this project. I realized that I wasn’t outputting everything I was asked for. I had to write some new methods and rewrite others in order to have the correct output at the beginning of every minute and at the end of every minute. I also made sure as I was writing the various classes that they all had a way to output a string representing the various objects. I found a way to output the ArrayDeques as a string by first converting them into arrays and using the Arrays.toString() method on them.

## Conclusions

This is the hardest section to write in this documentation. I’m not sure what I can conclude from this project. While writing this project, I learned that ArrayDeque is the most performant queue in Java ("What is the fastest Java collection with the basic functionality of a Queue?"). It was often compared to a Linked List which I know we’ll be discussing in week 3 of class so I didn’t explore much more as to why ArrayDeque is better.

Many situations in life can be represented as a stack in Java. Any time that a collection of objects needs to be maintained in a specific order where an object is then picked off either at the head or at the tail, a stack is probably a good choice.

References

D, C. (2011, May 25). What is the fastest Java collection with the basic functionality of a Queue? Retrieved June 28, 2015, from http://stackoverflow.com/questions/6129805/what-is-the-fastest-java-collection-with-the-basic-functionality-of-a-queue

Salus, P. (1994). *A quarter century of UNIX*. Reading, Mass.: Addison-Wesley Pub.