CSCI-201: Principles of Software Development - Fall 2020

PA3: Fragile Express x SalEats

Concepts covered:

- Locks Programming
- Semaphore Programming
- Multi-Threaded Programming Design

Prelude

Another day, another batch of deliveries. With extra protection gear and a Bola wire launcher on hand, timefall and bandits are distant concerns of the past for Porters under the employment of Fragile Express. Today is an odd day because Fragile Express have entered into a contract with a struggling food delivery company, SalEats.

According to the Old Ones who survived the mass quarantine caused by the first voidout, back in the golden age of social interaction, SalEats dominated the streets of Figueroa and was the lifeline of those who were stuck in the pits of despair southwest of the Epstein Family Plaza. Literally tens of people used to live there overnight, now it's a ghost town.

SalEats used to run a robust logistics system originally proudly designed by the best programmers handpicked by its director but the data center was lost to a timefall before the recovery team could arrive onsite. The team only managed to recover a small portion of data - namely a program called "MOSS". Nobody knows exactly how it does what it does but legends say it relentlessly punishes anyone who dares to violate a piece of holy text called the "Viterbi Academic Integrity Honor Code".

You are the *Tech Lead* of Fragile Express and your order is to design a cutting-edge logistics system that supports the scheduling of Porters and food deliveries.

The Assignment

In this assignment, you will first read in a JSON file containing various information regarding the restaurants and their menus. The JSON format is generally the same as the one we used in the previous assignments, so feel free to reuse the parser you've built. However, you will need to alter the classes as we have modified the JSON object. Here's a sample JSON:

```
{
  "data": [
    {
      "name": "Boba Guys",
      "address": "1670 Beverly Blvd",
      "latitude": 34.063950,
      "longitude": -118.265360,
      "drivers": 2,
      "menu": [
          "strawberry matcha milk tea",
          "black sesame latte",
          "thai tea"
        ]
    },
    {
      "name": "Twinkle Brown Sugar",
      "address": "131 S Central Ave",
      "latitude": 34.048160,
      "longitude": -118.239240,
      "drivers": 3,
      "menu": [
          "brown sugar milk tea",
          "mango tango green tea",
          "rose lychee green tea",
          "brown sugar fluffy coffee",
          "sea salt iced coffee",
          "strawberry green milk tea"
        ]
    },
    {
      "name": "One Zo",
      "address": "500 N Atlantic Blvd",
      "latitude": 34.068280,
      "longitude": -118.133680,
      "drivers": 1,
      "menu": [
          "one zo boba milk tea",
          "caramel milk tea",
          "honey green tea",
          "one zo fruit tea"
        ]
    }
  ]
}
```

Aside from the JSON file, you will need to read in a second file. Below is a layout of the second text file, which contains information for the orders:

```
0, Boba Guys, strawberry matcha milk tea
0, Twinkle Brown Sugar, sea salt iced coffee
1, One Zo, caramel milk tea
5, Boba Guys, black sesame latte
18, One Zo, one zo boba milk tea
18, Twinkle Brown Sugar, strawberry green milk tea
25, Twinkle Brown Sugar, rose lychee green tea
```

- The first parameter indicates when the order is ready.
- The second parameter indicates the location that the order is coming from.
- The third parameter indicates which food is being delivered.

It takes each Porter (driver) exactly one second to travel one mile, which is the equivalent of a *USAF X-51A Waverider* jet traveling at Mach 6. Leveraging the Chiral Network and their trusty delivery truck, everything is possible.

You can assume that each porter will only travel back and forth between one restaurant, and the order is not completed until the porter returns to the restaurant.

You will NOT be using the Haversine formula for distance calculation. You can use the simplified formula provided below as pseudocode:

```
private Double latitude;
private Double longitude;
public double getDistance(double lat, double lon) {
        double delta = lon - longitude;
        double dist = Math.sin(Math.toRadians(lat)) *
Math.sin(Math.toRadians(latitude))
                + Math.cos(Math.toRadians(lat)) *
Math.cos(Math.toRadians(latitude)) * Math.cos(Math.toRadians(delta));
        dist = Math.acos(dist);
        dist = 3963.0 * dist;
        return dist;
}
```

Just like the previous assignment, prompt the user to enter a filename and check for the file's existence and validity when the program first runs. You will also prompt the user for the latitude and longitude of their current location. **All input must be validated.**

You will need to utilize locks and conditions to settle the availability of the porters. For example, one porter cannot take two orders at the same time. The first trip must be completed before starting the second trip. The program should output when an order is picked up and when it is completed.

Example output:

```
What is the name of the file containing the restaurant information? pa3.txt

What is the name of the file containing the schedule information? schedule.txt

What is the latitude? 34.021160

What is the longitude? -118.287132

Starting execution of program...
[00:00:00.00] Starting delivery of strawberry matcha milk tea from Boba Guys!
[00:00:00.00] Starting delivery of sea salt iced coffee from Twinkle Brown Sugar!
[00:00:01.00] Starting delivery of caramel milk tea from One Zo!
[00:00:05.00] Starting delivery of black sesame latte from Boba Guys!
[00:00:06.40] Finished delivery of strawberry matcha milk tea from Boba Guys!
[00:00:06.60] Finished delivery of sea salt iced coffee from Twinkle Brown Sugar!
[00:00:11.40] Finished delivery of black sesame latte from Boba Guys!
[00:00:18.00] Starting delivery of strawberry green milk tea from Twinkle Brown
Sugar!
[00:00:19.80] Finished delivery of caramel milk tea from One Zo!
[00:00:19.80] Starting delivery of one zo boba milk tea from One Zo!
[00:00:24.60] Finished delivery of strawberry green milk tea from Twinkle Brown
Sugar!
[00:00:25.00] Starting delivery of rose lychee green tea from Twinkle Brown Sugar!
[00:00:31.60] Finished delivery of rose lychee green tea from Twinkle Brown Sugar!
[00:00:38.60] Finished delivery of one zo boba milk tea from One Zo!
All orders complete!
```

Keep in mind, due to the nature of threading, timing between different runs may differ and that is okay. There is a generous time tolerance during grading.

For your convenience, we have included some test cases with different levels of difficulty.

Grading Criteria

You will be graded based on the correctness of scheduling as well as the order and duration of deliveries.

- 20% - File I/O
  - 5% - JSON I/O
  - 5% - Text file I/O
  - 10% - Input validation
- 80% - Program Execution
  - 40% - Output correctness

- 40% - Semaphores and/or locks implementation