## Question:

Assume that you're book publisher and you sell those books on Amazon. Being a publisher means you have an unlimited inventory and can sell any number of books as per the demand.

However, your book has got a competition in market, and a different book on same topic and concepts is also available on Amazon by a different publisher.

You being no ordinary seller, you've come up with a strategy to control the prices of your book dynamically, based on:

1. How your competitor prices his book
2. The demand in market.

To do this you can use Amazon's APIs. The APIs allows you to send a POST request to set price of a book, send a GET request to get updates on sales. It also allows you to download the anonymized competitors' price (data.csv).

Here is your strategy you plan to implement:

## The strategy:

1. Using the past data, you're calculating averages of last 5 prices of your competitor's book (say avg_comp_price)
2. You send a HTTP request (POST /register-request) to Amazon's API server, to set price of the book on the portal
3. Amazon's API server being bit overloaded, accepts the request, sends 200 OK response along with an ID of the request (as acknowledgement).
4. You poll after x seconds (choose x) to get the update if there has been a sale. To do that, you send a HTTP request (GET /request-data/:id) to get the information.
5. If there was sale, you set the price of your book as: avg_comp_price * 1.1
6. If there was no sale, you set the price of your book as: avg_comp_price * 0.9

## Your task:

Write a program so that you can automate this process.

## Classes to create:

| Functionality | ClassName | Functionality |
|---|---|---|
| Data Manager | DataManager | 1. This receives a request to get data for last 5 points.<br>2. If it has the data for all these points in memory, it just sends them.<br>3. If not, it reads the relevant data from CSV, and stores it in memory, and then sends the data. |
| Amazon API Server | APIServer | 1. An HTTP Server, listening on a port.<br>2. Has following end-points:<br>a. POST /register-request<br>Returns a unique ID, to be passed on to next request<br><br>b. GET /request-data/:id<br>If <id> is even, returns { sold: true }<br>If <id> is odd, returns { sold: false } |
| Scheduler | Scheduler | 1. Triggers a tick event every 5 sec |
| Strategy | Algorithm | 1. This talks to all 3 components mentioned above.<br>2. On every tick event of the Scheduler, it requests the data from the Data Manager.<br>3. Then it computes price average (avg_comp_price)<br>4. It then makes the first HTTP request (POST /register-request) to the API Server to get an ID<br>5. After x seconds, it makes second HTTP request (GET /request-data/:id)<br>6. Based on the data received in last HTTP request, it calculates the final price of your book, and prints on stdout. |

**Important:** The Algorithm should communicate with DataManager over some sort of message queue.

## Things to Note:

1. Please code in Python (or your preferred language?)
2. The program should not hang/freeze and should be built, assuming it will run for 5-6 hours
3. Memory consumption and CPU use to be optimized
4. Code should be modular, and object oriented, and should follow a microservice architecture.
5. Use Events driven architecture, or threads, or async programming, whichever you feel will help meet the requirements of handling scale. (Scale can come as following same strategy for many books or a few strategies independently setting the price of the same book - no need to implement)
6. Organize the code, so that even if one component fails/terminates, the others remain functional. For example, if Algorithm fails, the Scheduler should continue to tick, and so on.
7. Disk IO or file read should not block the execution of the program. Also, instead of loading the entire file data into memory at once, if code can read one line at a time (and optimize memory consumption), it will be highly preferred.
8. Push the code to GitHub, with a ReadMe, and mention relevant details (like Python version to use)

9. You’re free to use any packages/libraries of your choice. But they should be listed in ReadMe, with installation instructions.