**Santander Corporate & Investment Banking (FX)**

**Java Developer 2022 - Market Price Handler Exercise**

**Background**:

We would like to connect to a FX price feed of spot prices from the market. With each price, we want to modify it by applying a margin adjustment to each price (like commission).

With the new price, we then want to make this available to clients/components so they can get the **latest price** for an instrument whenever they ask for it (history is not required).

You can imagine that there could be a UI to show this price to a real user, but in this exercise, you do not need to write a UI, typically a colleague would be working on this part.

The definition of a Price consists of unique id, instrument name, bid, ask and timestamp. You can assume that the Bid means the sell price (which is lower) and the Ask is the buy price (which is higher).

The market price feed will be given to you in CSV format line by line for EUR/USD, GBP/USD and EUR/JPY, e.g. here are some individual messages (each message could have 1 or more lines in it):

106, EUR/USD, 1.1000,1.2000,01-06-2020 12:01:01:001

…

107, EUR/JPY, 119.60,119.90,01-06-2020 12:01:02:002

…

108, GBP/USD, 1.2500,1.2560,01-06-2020 12:01:02:002

…

109, GBP/USD, 1.2499,1.2561,01-06-2020 12:01:02:100

…

110, EUR/JPY, 119.61,119.91,01-06-2020 12:01:02:110

**Tasks**:

Use Java to write classes that implement a subscriber to process the feed and store the prices for the UI. Frameworks are optional. The focus is clear and concise code.

Some examples of tasks could be:

* A subscriber to listen to the market prices. You can assume the feed is coming from a messaging system where all you have to do is implement an interface, e.g. void onMessage(String message).
* Each message is a CSV string (see above format).
* With an incoming price, process each with a margin (add commission) function, assume it is simply -0.1% on bid, +0.1% on ask (subtract from bid, add to ask).
* Publish the adjusted price to REST endpoint (just show where the endpoint will be, do not need to implement a webserver).
* Write a suitable test that gets the latest price, i.e. a client that gets, then prints out to console.

It is important to break the problem down into smaller testable pieces, and write the minimum amount of code, that should be close to production quality. You can also write tests where you think most appropriate.

Please spend up to 2 hours on the task. Provide java code files in a text document, zip or github. If you make any assumptions, please add comments in the code or into the document.

<End>