Code Challenge: Capital Gains

Challenge Context

You are tasked with implementing a command line application (CLI) that calculates how much tax you should pay based on the profit or losses of a stock market investment.

Please make sure you read all the instructions below, and feel free to ask for clarifications if needed.

Sample usage of the Capital Gains

How the program should work

Input

Your program is going to receive arrays, one per line, containing stock market operations encoded in `JSON`, through the standard input (stdin) containing the following fields:

Field Name	Meaning
`operation`	If the operation was a `buy` or `sell`
`unit-cost`	The stock's unit cost using a currency with two decimal places
`quantity`	The quantity of stocks negotiated

Here is an example of input (formatted for clarity, each array would be on its own line in actual input):

```
[{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
{"operation":"sell", "unit-cost":20.00, "quantity": 5000}]
[{"operation":"buy", "unit-cost":20.00, "quantity": 10000},
{"operation":"sell", "unit-cost":10.00, "quantity": 5000}]
```

The operations are in the order they happened, this means the second operation in the array happened after the first one and so on.

Each line is an independent simulation, your program should not carry the state obtained from one line to the others.

The last line of the input will be a blank line.

Output

For each line of the input, the program should return an array describing how much taxes should be paid for each operation received encoded in `JSON` through standard output (stdout), containing the following field:

Field Name	Meaning

Field Name	Meaning
`tax`	The amount of taxes to be paid based on operation

Here is an example of the expected output:

```
[{"tax": 0.0}, {"tax": 10000.0}]
[{"tax": 0.0}, {"tax": 0.0}]
```

The result list should have the exact size of the input list of operations. e.g., given three operations (buy, buy, sell), there should be three results representing the taxes for each operation.

Capital Gains Rules

In summary, the program will receive two kinds of operations (`buy` and `sell`) to calculate the capital gains tax. The tax is fixed at 20% of the **overall profit** obtained by trading stocks, and operations with a total amount equal to or below $ 20,000.00 do not pay taxes. The rules to calculate the tax are detailed below:

- The percentage you pay in taxes is 20% of the operation's profit but since the stocks could have been purchased at different prices, the **weighted average** price should be considered as the buying price (see below)
- The **weighted-average** price is the average buying price taking into account the amount of stocks purchased so far. So when you buy stocks you should recalculate the weighted average price using this formula `new-weighted-average-price = ((current-stock-quantity * weighted-average-price) + (new-stock-quantity * new-price)) / (current-stock-quantity + new-stock-quantity)`. e.g., if you have bought 10 stocks for $ 20.00, sold 5 and bought other 5 for $ 10.00, the weighted average is `((5 x 20.00) + (5 x 10.00)) / (5 + 5) = 15.00`.
- Losses (i.e., you are selling for a price lower than the weighted average price you have bought) do not pay any taxes, and you need to deduct the loss from subsequent profits before calculating the tax. In other words, tax applies to the **overall profit**, so you should use a past loss on multiple future profits (until you deduct the entire amount);
- You do not pay any taxes nor deduct the profit from accumulated losses if the total amount of the operation (unit cost of selling x quantity) is less than or equal to $ 20,000.00 (but losses should be deducted from the subsequent profits) - remember to use the total amount and not the profit to decide whether you should pay taxes;
- You do not pay any taxes for buying stocks.

You can assume that no operation will sell more stocks than you currently have.

Capital Gains Examples

Note: Operations are shown in different lines to fit page width, each input line will contain a whole array

Case #1

Operation	Unit Cost	Quantity	Tax	Explanation

Operation	Unit Cost	Quantity	Tax	Explanation
`buy`	10.00	100	0	Buying stocks do not pay taxes
`sell`	15.00	50	0	Total amount less than $ 20,000
`sell`	15.00	50	0	Total amount less than $ 20,000

Input:

```
[{"operation":"buy", "unit-cost":10.00, "quantity": 100},
{"operation":"sell", "unit-cost":15.00, "quantity": 50},
{"operation":"sell", "unit-cost":15.00, "quantity": 50}]
```

Output:

```
[{"tax": 0.0},{"tax": 0.0},{"tax": 0.0}]
```

Case #2

Operation	Unit Cost	Quantity	Tax	Explanation
`buy`	10.00	10000	0	Buying stocks do not pay taxes
`sell`	20.00	5000	10000	Profit of $ 50,000: 20% of taxes is $ 10,000 and there is no previous losses to use
`sell`	5.00	5000	0	Loss of $ 25,000: no tax

Input:

```
[{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
{"operation":"sell", "unit-cost":20.00, "quantity": 5000},
{"operation":"sell", "unit-cost":5.00, "quantity": 5000}]
```

Output:

```
[{"tax": 0.0},{"tax": 10000.0},{"tax": 0.0}]
```

Case #1 + Case #2

When the application receives two lines, they should be handled as independent simulations. The program should not carry the state from processing the first input to the other executions.

Input:

```
[{"operation":"buy", "unit-cost":10.00, "quantity": 100},
{"operation":"sell", "unit-cost":15.00, "quantity": 50},
{"operation":"sell", "unit-cost":15.00, "quantity": 50}]
[{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
{"operation":"sell", "unit-cost":20.00, "quantity": 5000},
{"operation":"sell", "unit-cost":5.00, "quantity": 5000}]
```

Output:

```
[{"tax": 0.0},{"tax": 0.0},{"tax": 0.0}]
[{"tax": 0.0},{"tax": 10000.0},{"tax": 0.0}]
```

Case #3

Operation	Unit Cost	Quantity	Tax	Explanation
`buy`	10.00	10000	0	Buying stocks do not pay taxes
`sell`	5.00	5000	0	Loss of $ 25,000: no taxes
`sell`	20.00	3000	1000	Profit of $ 30,000: deduct Loss of $ 25,000 and pay 20% of $ 5,000 in taxes ($ 1,000)

Input:

```
[{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
{"operation":"sell", "unit-cost":5.00, "quantity": 5000},
{"operation":"sell", "unit-cost":20.00, "quantity": 3000}]
```

Output:

```
[{"tax": 0.0},{"tax": 0.0},{"tax": 1000.0}]
```

Case #4

Operation	Unit Cost	Quantity	Tax	Explanation
`buy`	10.00	10000	0	Buying stocks do not pay taxes
`buy`	25.00	5000	0	Buying stocks do not pay taxes
`sell`	15.00	10000	0	Considering average price of $ 15 ((10×10,000 + 25×5,000) ÷ 15,000) there is no profit or loss

Input:

```
[{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
{"operation":"buy", "unit-cost":25.00, "quantity": 5000},
{"operation":"sell", "unit-cost":15.00, "quantity": 10000}]
```

Output:

```
[{"tax": 0.0},{"tax": 0.0},{"tax": 0.0}]
```

Case #5

Operation	Unit Cost	Quantity	Tax	Explanation
`buy`	10.00	10000	0	Buying stocks do not pay taxes
`buy`	25.00	5000	0	Buying stocks do not pay taxes
`sell`	15.00	10000	0	Considering average price of $ 15 there is no profit or loss
`sell`	25.00	5000	10000	Considering average price of $ 15 profit of $ 50,000: pay 20% of $ 50,000 in taxes ($ 10,000)

Input:

```
[{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
{"operation":"buy", "unit-cost":25.00, "quantity": 5000},
{"operation":"sell", "unit-cost":15.00, "quantity": 10000},
{"operation":"sell", "unit-cost":25.00, "quantity": 5000}]
```

Output:

```
[{"tax": 0.0},{"tax": 0.0},{"tax": 0.0},{"tax": 10000.0}]
```

Case #6

Operation	Unit Cost	Quantity	Tax	Explanation
`buy`	10.00	10000	0	Buying stocks do not pay taxes
`sell`	2.00	5000	0	Loss of $ 40,000: total amount is less than $ 20,000, but we should deduct that loss regardless of that
`sell`	20.00	2000	0	Profit of $ 20,000: if you deduct all the loss, profit is zero. Still have $ 20,000 of loss to deduct
`sell`	20.00	2000	0	Profit of $ 20,000: if you deduct all the loss, profit is zero. Now there is no loss to deduct

Operation	Unit Cost	Quantity	Tax	Explanation
`sell`	25.00	1000	3000	Profit of $ 15,000 and zero losses to deduct: pay 20% of $ 15,000 in taxes ($ 3,000)

Input:

```
[{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
{"operation":"sell", "unit-cost":2.00, "quantity": 5000},
{"operation":"sell", "unit-cost":20.00, "quantity": 2000},
{"operation":"sell", "unit-cost":20.00, "quantity": 2000},
{"operation":"sell", "unit-cost":25.00, "quantity": 1000}]
```

Output:

```
[{"tax": 0.0},{"tax": 0.0},{"tax": 0.0},{"tax": 0.0},{"tax": 3000.0}]
```

Case #7

Operation	Unit Cost	Quantity	Tax	Explanation
`buy`	10.00	10000	0	Buying stocks do not pay taxes
`sell`	2.00	5000	0	Loss of $ 40,000: total amount is less than $ 20,000, but we should deduct that loss regardless of that
`sell`	20.00	2000	0	Profit of $ 20,000: if you deduct all the loss, profit is zero. Still have $ 20,000 of loss to deduct
`sell`	20.00	2000	0	Profit of $ 20,000: if you deduct all the loss, profit is zero. Now there is no loss to deduct
`sell`	25.00	1000	3000	Profit of $ 15,000 and zero losses to deduct: pay 20% of $ 15,000 in taxes ($ 3,000)
`buy`	20.00	10000	0	All stocks were sold. Buying new ones changes the average price to the paid price of the new stocks ($ 20)
`sell`	15.00	5000	0	Loss of $ 25,000
`sell`	30.00	4350	3700	Profit of $ 43,500: if you deduct the loss of $ 25,000, there is $ 18,500 of profit left. Pay 20% of $ 18,500 in taxes ($ 3,700)
`sell`	30.00	650	0	Profit of $ 6,500: no loss to deduct, but the total amount is less than $ 20,000

Input:

```
[{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
{"operation":"sell", "unit-cost":2.00, "quantity": 5000},
{"operation":"sell", "unit-cost":20.00, "quantity": 2000},
{"operation":"sell", "unit-cost":20.00, "quantity": 2000},
{"operation":"sell", "unit-cost":25.00, "quantity": 1000},
{"operation":"buy", "unit-cost":20.00, "quantity": 10000},
{"operation":"sell", "unit-cost":15.00, "quantity": 5000},
{"operation":"sell", "unit-cost":30.00, "quantity": 4350},
{"operation":"sell", "unit-cost":30.00, "quantity": 650}]
```

Output:

```
[{"tax": 0.0},{"tax": 0.0},{"tax": 0.0},{"tax": 0.0},{"tax": 3000.0},
 {"tax": 0.0},{"tax": 0.0},{"tax": 3700.0},{"tax": 0.0}]
```

Case #8

Operation	Unit Cost	Quantity	Tax	Explanation
`buy`	10.00	10000	0	Buying stocks do not pay taxes
`sell`	50.00	10000	80000	Profit of $ 400,000: pay 20% of $ 400,000 in taxes ($ 80,000)
`buy`	20.00	10000	0	Buying stocks do not pay taxes
`sell`	50.00	10000	60000	Profit of $ 300,000: pay 20% of $ 300,000 in taxes ($ 60,000)

Input:

```
[{"operation":"buy", "unit-cost":10.00, "quantity": 10000},
{"operation":"sell", "unit-cost":50.00, "quantity": 10000},
{"operation":"buy", "unit-cost":20.00, "quantity": 10000},
{"operation":"sell", "unit-cost":50.00, "quantity": 10000}]
```

Output:

```
[{"tax": 0.0},{"tax": 80000.0},{"tax": 0.0},{"tax": 60000.0}]
```

Case #9

Operation	Unit Cost	Quantity	Tax	Explanation
`buy`	5000.00	10	0	Buying stocks do not pay taxes

Operation	Unit Cost	Quantity	Tax	Explanation
`sell`	4000.00	5	0	Loss of $5,000: total amount is equal $ 20,000, but we should deduct that loss regardless of that
`buy`	15000.00	5	0	Buying stocks do not pay taxes
`buy`	4000.00	2	0	Buying stocks do not pay taxes
`buy`	23000.00	2	0	Buying stocks do not pay taxes
`sell`	20000.00	1	0	Total operation amount <= $ 20,000: it does not pay taxes nor touches the losses
`sell`	12000.00	10	1000	Profit of $ 10,000: if you deduct the loss of $ 5,000, there is $ 5,000 of profit left. Pay 20% of $ 5,000 in taxes ($ 1,000)
`sell`	15000.00	3	2400	Profit of $ 12,000 and zero losses to deduct: Pay 20% of $ 12,000 in taxes ($ 2,400)

Input:

```
[{"operation":"buy", "unit-cost":  5000.00, "quantity":  10},
{"operation":"sell", "unit-cost":  4000.00, "quantity":   5},
{"operation":"buy",  "unit-cost": 15000.00, "quantity":   5},
{"operation":"buy",  "unit-cost":  4000.00, "quantity":   2},
{"operation":"buy",  "unit-cost": 23000.00, "quantity":   2},
{"operation":"sell", "unit-cost": 20000.00, "quantity":   1},
{"operation":"sell", "unit-cost": 12000.00, "quantity":  10},
{"operation":"sell", "unit-cost": 15000.00, "quantity":   3}]
```

Output:

```
[{"tax": 0.0},{"tax": 0.0},{"tax": 0.0},{"tax": 0.0},{"tax": 0.0},
{"tax": 0.0},{"tax": 1000.0},{"tax": 2400.0}]
```

Application State

The program **should not** rely on any external database, and the application's internal state should be handled by an explicit in-memory structure. The application state needs to be reset at the start of the application.

Rounding Decimals

The program should round to the nearest hundredth values (two decimal places) when dealing with decimal numbers. For example:

If you buy 10 shares for $20.00 and 5 shares for $10.00, the weighted average price is (10 x 20.00 + 5 x 10.00) / 15 = 16.67.

Error handling

Please assume that input parsing errors will not happen. We will not evaluate your submission against input that contains errors, is formatted incorrectly, or which breaks the contract.

Numbers output format

In the most common libraries of some languages that handle JSON, the trailing zeros are removed. If this occurs in your chosen language, prefer returning a number (Int, Long, Float, Double, BigDecimal, etc) with fewer digits rather than casting them to other types (strings and so on).

Our expectations

We at Nubank value the following criteria:

- **Simplicity**: the solution is expected to be a small project and easy to understand;
- **Elegance**: the solution is expected to be easy to maintain, have a clear separation of concerns, and well-structured code organization;
- **Operational**: the solution is expected to solve the problem, cover possible corner cases, and be extensible for future design decisions;
- **Quality**: as you structure your code, we expect you to write tests to ensure the program is working properly. Well-written tests help to build a robust and maintainable solution;
- **Good practices**: besides the unit tests, we recommend and expect that you write tests that cover your solution from end to end, i.e., from the input to the program's output. One way to do that is by adding integration tests;
- **Test validation**: We expect the solution to be validated with tests. The absence of tests may impact the quality assessment of your solution, once we consider the tests as an essential part of the development process.

We will look for:

- The use of referential transparency when applicable;
- Quality unit and integration tests;
- Documentation where it is needed;
- Instructions on how to run the code.

Lastly, but not least expected:

- You may use open source libraries you find suitable to support you in solving the challenge, e.g., json parsers; Please refrain as much as possible from adding frameworks and unnecessary boilerplate code.
- The challenge expects a **standalone** command-line application; please refrain from adding unnecessary infrastructure and/or dependencies. You are expected to be able to identify which tools are required to solve the problem without adding extra layers of complexity.

General notes

- This challenge may be extended by you and a Nubank engineer on a different step of the process;
- The Capital Gains application should receive the operations data through `stdin` and return the processing result through `stdout`, rather than through, for example, a REST API.

Packing The Solution for submission

- You should submit your solution source code as a compressed file (zip) containing the code and documentation. Please make sure not to include unnecessary files such as compiled binaries, libraries, etc.;
- Please do not upload your solution to public repositories such as GitHub, BitBucket, etc.;
- If you are going to use dockerized builds, do not upload your image in public hubs such as DockerHub, Sloppy.io, etc.

Remove Personal Information

⚠ **IMPORTANT:** Please remove all personal information from the files of the challenge before submitting the solution. Pay special attention to the following:

- Source files like code, tests, namespaces, packaging, comments, and file names;
- Automatic comments your development environment may add to solution files;
- Code documentation such as annotations, metadata, and README.MD files;
- Version control configuration and author information.

If you plan to use git as the version control system, execute the following in the repository root to export the solution anonymized:

```
git archive --format=zip --output=./capital-gains.zip HEAD
```

Add a README

Your solution should contain a README file containing:

- Discussing the technical and architectural decisions;
- Reasoning about the frameworks used (if any framework/library was used);
- Instructions on how to compile and run the project;
- Instructions on how to run the tests of the solution;
- Additional notes you consider important to be evaluated.

Running Environment

It must be possible to build and run the application under Unix or Mac operating systems. Dockerized builds are welcome.

FAQ

> Q: How do I read the input through stdin? Does it have to be in a file input.txt? Do I have to ask the user to input the file name through the terminal?

A: Reading through stdin is usually the simplest way to read input in any command line application, e.g., Console.ReadLine() in C# or input() in Python. Your solution should expect the user to input each line to the terminal and press 'enter'. This also allows us to pass the input through Input Redirection. For example:

```
./capital-gains < input.txt
```

Additionally, we don't expect your solution to print out any explanation to the user, such as `"Please, insert the operations:"`. You can assume that the user knows what input your program expects in which order. The only output expected to be printed is the JSON tax responses.

> Q: Can there be a buy event after sell events? In that case, should the average purchase price be recalculated using the new purchase along with the previous purchases?

A: Yes, the average purchase price should always consider all previous purchase events up to the current sell event. Please refer to Case #7 for a practical example.