Handout

Part I: Implementation (180 min)

Task

Imagine you are a software engineer at Quantedge. You have been tasked to develop a **portfolio management system** (PMS) for the company. This system will be responsible for tracking our portfolios' market positions across time.

The PMS will typically receive a request asking for the market positions of a portfolio on a specific date, i.e.:

```
{
        "date": "2021-01-04",
        "portfolio_code": "BLSH"
}
```

It will then have to construct the portfolio positions using the data from various sources, and return something like this:

```
[
        {
                "date": "2021-01-04",
                "security": {
                        "asset_class": "mcc",
                        "security_code": "BRKCRD2022-03-01",
                        "commodity_1": "BRK",
                        "commodity_2": "CRD",
                        "swap_rate": 20.0,
                        "expiry": "2022-03-01"
                },
                "broker_code": "Bernstein",
                "quantity": 7.25
        },
          {
            "date": "2021-01-04",
            security": {
                      "security_code": "BRKCRD2022-03-01",
                      "asset_class": "mcc",
                      "commodity_1": "BRK",
                      "commodity_2": "CRD",
                      "swap_rate": 20.0,
                      "expiry": "2022-03-01"
            },
            "broker_code": "Alabaster",
            "quantity": 1.05
          },
          {
            "date": "2021-01-04",
            "security": {
                      "security_code": "CA_2022-03-01",
                      "underlying": "CASHEW",
                      "expiry": "2022-03-01",
                      "asset_class": "laf"
            },
            "broker_code": "Bernstein",
            "quantity": -504.0
          },
        .
        .
        .
]
```

In reality, this might be an API endpoint on the PMS, but we do not expect you to code out a backend server for this exercise. Instead, it is sufficient for you to encapsulate all the business logic within an entrypoint function as follows:

```
def entrypoint(request_file_path: str):
    request = read_json(request_file_path)
    response = perform_main_business_logic(request)
    save_to_file(response, "response.json")
```

The entrypoint function will

- Accept a file path to a request (provided in JSON format in the test cases folder)
- Perform business logic to construct the portfolio positions (the output above) and output them to another JSON file

More details about the request and response formats can be found below (see API Contract section).

The example entrypoint is written in Python, but there are no constraints on programming language. Use whichever language you are most comfortable with.

Definitions

Before we begin, keep the following definitions in mind as you read through this specification. These definitions and examples should be sufficient for you to understand the business problem at hand.

Term	Definition
Security	Any financial instrument we trade is called a security.
Asset Class	An asset class is a related group of securities.
Broker	An intermediary between us and the securities exchange to help us place our trades on the exchange.
Transaction	A single event that adds or removes a certain quantity of a security from a portfolio.
Position	A position refers to the quantity of a security held as of a particular date with a specific broker.
Portfolio	A group of positions arising from the same investment strategy forms a portfolio. We have two kinds of portfolios: 1. Simple portfolios: A simple portfolio is constructed solely from the underlying transactions tagged to it. 2. Compound portfolios: A compound portfolio is composed of 2 constituent portfolios. Note that the constituent portfolios can be simple or compound portfolios.

Input Data

The main task of the PMS is to construct portfolio positions given some set of input data. You are provided with 3 types of input data, as described below.

Securities (securities.json)

This contains the exhaustive list of securities to manage via the PMS. This file is structured in the form of a key-value mapping from a security's identifier to its metadata. For this PMS, we are looking to handle 2 *hypothetical* asset classes - **Multi-Commodity Contracts (MCCs)** and **Leveraged Agriculture Forwards (LAFs)**.

Multi-Commodity Contract

- **security_code (str):** unique identifier for this security
- **asset_class (str)**: "mcc"
- **commodity_1 (str):** first commodity defined on this contract
- **commodity_2 (str):** second commodity defined on this contract
- **swap_rate (float):** exchange rate or how many units of commodity_1 trades for 1 unit of commodity_2
- **expiry (date, YYYY-MM-DD):** expiry date for this contract

Leveraged Agriculture Forward

- **security_code (str):** unique identifier for this security
- **asset_class (str)**: "laf"
- **underlying (str):** underlying physical agricultural product that backs this security
- **expiry (date, YYYY-MM-DD):** expiry date for this forward

Portfolios (portfolios.json)

This file contains the list of portfolios (both simple and compound) Quantedge has. Each portfolio should have the following fields:

- **portfolio_code (str)**: unique identifier for this portfolio
- **description (str)**: description of the portfolio
- **children (list, optional)**: list of constituent portfolios; always size 2
- **operation (str, optional)**: defines the relationship between the 2 children portfolios. eg. “add” means we sum the two portfolios together, and “subtract” means we subtract the second portfolio from the first. This will be further explained in the Portfolio Aggregation section.

Transactions (transactions.csv)

This file contains all our transactions from 1st Jan 2021 to 31st Dec 2021. Each row represents a trade we entered or exited with a broker.

- **date (str):** trade execution date
- **asset_class (str)**: asset class of the security
- **security_code (str)**: unique identifier of a security. For this assignment, this can be either the “security_code” from Multi-Commodity Contract or from Leveraged Agriculture Forward. It is guaranteed that the two asset classes do not have any overlapping identifiers.
- **portfolio_code (str)**: unique identifier of a portfolio
- **broker_code (str)**: unique identifier of a broker
- **quantity (float)**: the amount traded; can be positive or negative. The interpretation for a quantity’s sign does not matter for the scope of this assignment. You can think of positive quantity as buying a security and a negative quantity as selling it.

Test cases (test_cases/*.json)

This folder contains some test cases and their corresponding expected output. You can feed these test cases into your entrypoint function and compare your response with the provided one to verify correctness. However, do note that the test suite provided is not comprehensive; you might be have to come up with more tests to cover some other cases.

Specifications

We will now talk about the economics that the PMS must apply to correctly construct portfolios.

Position Calculations

Recall that transactions add or remove a quantity of security from a portfolio. By inspecting transactions, we can construct the positions of any portfolio at any arbitrary date.

Examples:

Consider an equity portfolio called BuyOnly that was started on 2022-08-01 (i.e. no other transactions prior), with a naive buy-only strategy. BuyOnly is very bullish on securities that start with A, i.e. AAPL and AMZN.

Case 1:

Observed Transactions:

- 2022-08-01: Buy 100 shares of AAPL via Broker A
- 2022-08-02: Buy another 100 shares of AAPL via Broker A

On 2022-08-02, our BuyOnly portfolio will now contain the following positions

- 200 shares of AAPL with Broker A

Case 2:

Observed Transactions:

- 2022-08-01: Buy 100 shares of AAPL via Broker A
- 2022-08-02: Buy 100 shares of AAPL via Broker B
- 2022-08-02: Buy 50 shares of AMZN via Broker B

On 2022-08-02, our BuyOnly portfolio will now contain the following positions

- 100 shares of AAPL with Broker A
- 100 shares of AAPL with Broker B
- 50 shares of AMZN with Broker B

Portfolio Aggregation

At the most general level, a portfolio is a set of positions that changes across time. There is no constraint on what a portfolio can contain - it can have mixed asset classes, negative positions, or even no positions. For simplicity, we only look at a portfolio at a specific point in time in terms of absolute quantity per security.

Note the following properties of a portfolio:

1. A set of portfolios can decompose into a set of positions.
2. Therefore, a set of portfolios is itself a portfolio. We call sets of portfolios **compound portfolios**.

For this assignment, we assume a set of portfolios has a max size of 2 portfolios. The simplest compound portfolio is the *sum* of 2 portfolios, but we may also want to construct a compound portfolio as the *difference* between 2 portfolios.

Example:

```
# given the following 3 actual portfolios

BuyHigh = [
        {
                "date": "2022-01-01",
                "security_code": "CO_2022-03-01",
                "broker": "broker_a",
                "quantity": 100
        },
]
SellLow = [
        {
                "date": "2022-01-01",
                "security": "CO_2022-03-01",
                "broker": "broker_a",
                "quantity": -180
        },
]

RiskFreePortfolio = []
```

Case 1: BuyHighSellLow = BuyHigh + SellLow

```
BuyHighSellLow = [
        {
                "date": "2022-01-01",
                "security_code": "CO_2022-03-01",
                "broker": "broker_a",
                "quantity": -80
        },
]
```

Case 2: BuyLowSellHigh = RiskFreePortfolio - BuyHighSellLow

```
BuyLowSellHigh = [
        {
                "date": "2022-01-01",
                "security_code": "CO_2022-03-01",
                "broker": "broker_a",
                "quantity": 80
        },
]
```

Securities

Recall we trade **Multi-Commodity Contracts (MCCs)** and **Leveraged Agriculture Forwards (LAFs)**.

In the equity portfolio example above (AAPL and AMZN), calculating the absolute quantity for each security is a simple task. In reality, there are idiosyncrasies for the 2 asset classes above that need to be handled differently.

Multi-Commodity Contract

MCCs are contracts that are described by 2 commodities, an expiry date, and a swap rate that represents the exchange between its 2 constituent commodities.

Assume we only trade **Crude Oil (CRD) MCCs**, so all the MCCs you process are guaranteed to have crude oil as one of its commodities.

Some example MCCs:

```
security_1 = {
        "security_code": "CRDLBR2022-03-01",
        "commodity_1": "CRD",
        "commodity_2": "LBR",
        "swap_rate": 0.1,
        "expiry": "2022-03-01"
}

security_2 = {
        "security_code": "LBRCRD2022-03-01",
        "commodity_1": "LBR",
        "commodity_2": "CRD",
        "swap_rate": 10,
        "expiry": "2022-03-01"
}
```

Please note the following

- Crude Oil MCCs usually have **CRD** as commodity_1, but this is not guaranteed.
- It is market convention to describe the quantity of an MCC in terms of units of commodity_1.
- Any Crude Oil MCC position in a portfolio must be described in quantity of **CRD.**
- Therefore, if a Crude Oil MCC has **CRD** as commodity_2, adjust the quantity using the contract's swap rate before you sum them.

Examples:

Refer to the securities shown above.

Case 1: Make 2 transactions for security_1, each time buying 100.

- Absolute Qty = **200 CRD**

Case 2: Make 2 transactions for security_2, each time buying 100. Since commodity_1 is not **CRD**, we have to use the swap_rate of 10 to get the absolute quantity.

- Absolute Qty = **200 LBR = 20 CRD**

Leveraged Agriculture Forward

LAFs are securities that are described by an underlying agricultural product and an expiry date.

A position for LAFs should be reported in **leveraged quantity**, as opposed to the original transaction quantity. The leveraged quantity is equal to the product of a **leverage factor** and the original transaction quantity.

The leverage factor is simply the number of months (integer) between the transaction date and expiry date. For simplicity, we can just consider the year and month (ignoring the exact date) when calculating the leverage factor.

An example LAF:

```
security_3 = {
        "name": "CO_2022-03-01",
        "underlying": "COFFEE",
        "expiry": "2022-03-01"
}
```

Case 1: Make a transaction on 2021-12-23 to buy 100 units of security_3.

- Leverage factor = number of months between 2021-12 and 2022-03 = **3 months**
- Leveraged quantity = 3 * 100 = **300**

Case 2: Make a transaction on 2021-03-31 to buy 100 units of security_3.

- Leverage factor = number of months between 2021-03 and 2022-03 = **12 months**
- Leveraged quantity = 12 * 100 = **1200**

API Contract

Entrypoint

To reiterate, it is sufficient for you to encapsulate all the business logic within an entrypoint function as follows:

```
def entrypoint(request_file_path: str):
    request = read_json(request_file_path)
    response = perform_main_business_logic(request)
    save_to_file(response, "response.json")
```

As mentioned above, while the example above is in Python, you are free to use the programming language of your choice.

Request file format

Recall that the request should contain 2 parameters:

- **date**: position as-of-date
- **portfolio_code**: unique identifier of a portfolio that the user wants to view the positions for

```
{
        "date": "2021-01-04",
        "portfolio_code": "BLSH"
}
```

Response file format

The response should contain an entry for each non-zero position tagged to the requested portfolio. Each entry should contain the following:

- **date (str)**: position as-of-date
- **security (json)**: security metadata
- **broker_code (str)**: broker code
- **quantity (float)**: number of units of the security at the as-of-date. Should be non-zero; should be omitted from the response if the dollar value nets to zero.

```
[
        {
                "date": "2021-01-04",
                "security": {
                        "asset_class": "mcc",
                        "security_code": "BRKCRD2022-03-01",
                        "commodity_1": "BRK",
                        "commodity_2": "CRD",
                        "swap_rate": 20.0,
                        "expiry": "2022-03-01"
                },
                "broker_code": "Bernstein",
                "quantity": 7.25
        },
          {
            "date": "2021-01-04",
            security": {
                      "security_code": "BRKCRD2022-03-01",
                      "asset_class": "mcc",
                      "commodity_1": "BRK",
                      "commodity_2": "CRD",
                      "swap_rate": 20.0,
                      "expiry": "2022-03-01"
            },
            "broker_code": "Alabaster",
            "quantity": 1.05
          },
          {
            "date": "2021-01-04",
            "security": {
                      "security_code": "CA_2022-03-01",
                      "underlying": "CASHEW",
                      "expiry": "2022-03-01",
                      "asset_class": "laf"
            },
            "broker_code": "Bernstein",
            "quantity": -504.0
          },
        .
        .
        .
]
```

Grading

Correctness	This position retrieval service will be used regularly by various teams to make important trading decisions, therefore, it is important that your core business logic is error-free, and the format of your output adheres to the API specifications.
Robustness	As software engineers, sometimes we fall into the trap of making assumptions to make our jobs easier. These assumptions can be about the data or user inputs, and are often fundamentally incorrect. We look out for how well you perform the following: • Question the assumptions • Handle edge cases • Validate input • Test your own code on all of the above
Software Design	For a new piece of software, it is crucial to establish a strong foundation for future extensions. We look out for how your code can be extended to accommodate future use cases and additional user requirements. Well-justified abstractions and code modularity are desired.
Speed	In the production setting, retrieving and calculating positions can be a computationally expensive operation. We are also interested in how you optimize your service to reduce latency.
Code Clarity	We all love code that is easy-to-understand, well-documented and coherent, and we strive to write such code for our code reviewers, future engineers who could take over our code, and of course, our future selves.

Logistics

Task Clarifications

If you require any additional clarification during the 180-minute implementation period, feel free to reach out to us via email (dev@quantedge.com).

Submission

Submission will be via email. Once the 180-minute time limit is up, please send your code over to us by replying to the interview invite email.

Resources

There are also no constraints on the resources you may use. Feel free to consult any online resources such as Google, StackOverflow and official documentations.

Part II: Discussion (40 min)

Code Walkthrough

For the first part of the interview, we would like you to walk us through your code. Explain to us the overall design and business logic, and touch on how you handled the idiosyncrasies and edge cases. You are also encouraged to elaborate on any interesting design decisions you made. If you have enriched the test suite, do also let us know what cases you are testing for.

Code QnA

The second part of the interview will be more interactive. We will ask you some questions regarding your code; you can expect the question to generally fall within the scope of the five grading criteria described above. We may also send you additional sample requests for testing.