χ-CHANGE SETUP AND CLIENT DOCUMENTATION

THE PLATFORM DEV TEAM

1. INTRODUCTION

The χ-change Platform is the University of Chicago Financial Market Program's proprietary in-house trading platform. As can be seen from other documentation, several of the cases in this year's competition will be run on the platform. The platform consists mainly of two parts: a **server** application (also referred to as the **exchange**) that externally resembles an exchange, and **client** code that can be used to construct **bots** that trade on the exchange. This documentation has several main goals:

(1) To inform you about how to set up the software on your system, including the code to write your own bots and the exchange executable
(2) To help you get started developing bots and provide some basic reference documentation for the client.

If you have any questions or concerns about the platform, please reach out via the associated Piazza channel.

2. SETUP

In order to set up your computer to be able to begin developing your bots, you will need to do the following things:

(1) Download and install Java 11
   - When this process is finished, you should be able to run

     ```
     java --version
     ```

     in your terminal/command prompt and see something like `Java 11P` displayed
(2) Download and install Python 3.8
   - Instructions can be found at `https://www.python.org/downloads/release/python-381/`
   - When this process is finished, you should be able to run

     ```
     python --version
     ```

     in your terminal/command prompt and see the result `Python 3.8.1` displayed
(3) Download the `competitor-files-1.0.0.zip` file, and extract its contents to a folder where you want to begin development of your bot.
(4) Navigate to the folder where you extracted the contents of the above zip file in a terminal or command prompt window. If you check the contents of this directory (using the command `dir` on Windows machines or `ls` on Mac OS or Linux machines), you should see several files including `requirements.txt` and `example_bot.py`

(5) Use your terminal/command prompt to navigate to the folder chosen in the previous step. In this folder, type

```
python -m venv venv/
```

- This creates a **virtual environment** in the folder `venv/`. A virtual environment is a self-contained installation of Python where you can install packages and modify settings.

(6) Next, activate this virtual environment

- If you're on a Mac or Linux computer, type something like the following into the terminal:

```
source venv/bin/activate
```

- If you're on a Windows computer, type something like:

```
.\venv\Scripts\activate.bat
```

- After typing this, you should see `(venv)` somewhere in your prompt. This indicates that the virtual environment is currently active.

(7) Now that your virtual environment is activated, type the following command:

```
pip install -r requirements.txt
```

- This loads all of the libraries that you will need to run the test trading bot that we've provided into the virtual environment you just created

The setup process is now complete, and you can begin developing your bots.

3. Running your Bot

In order to help you test the performance of your bots, we have provided you with a functioning version of exchange as well as the resources necessary to start a mock case. In order to start the exchange, simply run

```
java -jar xchange-1.0.0.jar --config=configs/case_1_config.yaml
```

to start a test run of case 1 and

```
java -jar xchange-1.0.0.jar --config=configs/case_2_config.yaml
```

to start a test run of case 2.

Once you have installed the dependencies as detailed in the previous section, run the provided example bot with the following:

```
python example_bot_case_1.py --host {host} --port {port} \
        --username {user} --password {pass}
```

where the fields `{host}` and `{port}` will be provided by us (and allow you to connect to a specific exchange that's running) and the `{user}` and `{pass}` can be whatever you want. If you are running the exchange locally, you do not need to specify a host and port parameter, as they default to localhost:9090.
Your bot will automatically register with the provided username and password. If your bot stops in the middle of a round, you can reconnect using the same credentials and your positions and PnL will carry over.

4. Viewing the Market

You can see the current state of the ladders by navigating to `{host}:8080` in your browser. If you are running the xchange.jar file locally, you should use `localhost`; otherwise, specify the IP address of the machine running the exchange. To see current Greek calculations, navigate to `{host}:8080/greeks.html`.

Appendix A. Documentation

Your trading bot should implement the `CompetitorBot` superclass. This will allow you to call the following methods from within your bot:

```
self.start(host, port, username, password)
```

Registers the client with the exchange as a competitor (if not registered already), waits for the case to start, and starts streaming exchange updates once the round starts

Parameters:

- `host (str)`: The address of the exchange server
- `port (str)`: The port of the server to access
- `username (str)`: The username to be used for registration
- `password (str)`: The password to be used for registration

```
self.place_order(order_type, order_side, qty, asset_code, px)
```

Place an order on the exchange

Parameters:

- `order_type (OrderType)`: The type of order this is
- `order_side (OrderSize)`: The side of the order
- `qty (int)`: The # of lots of the asset to buy
- `asset_code (str)`: The code of the asset
- `px (str)`: The price to buy/sell at

Returns:

- `Tuple[bool, order_book.PlaceOrderResponse]`: Whether the placement was successful and the full response

```
self.modify_order(order_id, order_type, order_side, qty, asset_code, px)
```

Modify an order that you've already placed (equivalent to atomic cancel + place)

Parameters:

- `order_id (str)`: The ID of the order to replace
- `order_type (OrderType)`: The type of order this is
- `order_side (OrderSize)`: The side of the order
- `qty (int)`: The # of lots of the asset to buy
- `asset_code (str)`: The code of the asset
- `px (str)`: The price to buy/sell at

Returns:

- `Tuple[bool, order_book.ModifyOrderResponse]`: Whether the modify was successful and the full response

```
self.cancel_order(order_id, asset_code)
```

Cancel an order

Parameters:

- `order_id (str)`: The ID of the order to cancel
- `asset_code (str)`: The code of the asset associated with this order

Returns:

- `Tuple[bool, order_book.CancelOrderResponse]`: Whether the cancel was successful and the full response

```
handle_round_started()
```

Handle a round being started. Is only called if registration was successful.

```
handle_exchange_update(exchange_update_response)
```

This is the method you need to implement to handle any exchange updates. Note that you will be getting lots of exchange updates, and this method will be called sequentially on received updates, so you should make sure this runs in a timely fashion. A slow implementation may act on stale prices, which can lead to bad PnL.