CASE 2: PAIRS TRADING

1. Introduction

This case examines how an understanding of the relationship between securities can be used to attain profits when prices deviate from the equilibrium.

Each team will be allowed to trade a portfolio of stocks in each round of the case. Your algorithm should be able to identify the profitable trading pairs within the portfolio of securities, discover the underlying relationship and profit from it, at the same time managing your exposure to secure the profit before the deviation reverts.

The case will consist of three rounds with 1,000 ticks each. (1) The first round will consist of only two securities with a pre-established relationship. Your job is to study it and profit from any deviations from its equilibrium. (2) In the second round, there will be three tradable securities so your job extends to being able to identify the *one* trading pair (and the third security, which has no meaningful relationship with the other two) and profit from it. (3) The third round will have five tradable securities available with *two* distinct trading pairs for your algorithm to tackle. Hence, you should also be aware of the trading opportunity between the two trading pairs for the last round.

Securities	HURON	SUPERIOR	MICHIGAN	ONTARIO	ERIE
Starting Price for Round 1	100	100	-	-	-
Starting Price for Round 2	100	100	100	-	-
Starting Price for Round 3	100	100	100	100	100

2. Addtional Details

To discourage running algorithm that only trades one particular security instead of attempting to discover a profitable pair, data for each individual security will be generated from a process where the expected returns are 0.

In addition, you must maintain a net zero position at all times. Hence, when you buy a security, you have to simultaneously sell another security in the same amount (and vice versa).

An absolute position limit (the sum of the absolute value of all of your open positions) will also be enforced. That is, if you have a long position of HURON of 10 lots and a short position of SUPERIOR of 10 lots, your absolute position will be 20. This constraint means your algorithm should allocate your exposure to where it thinks the most profitable trades will lie. The absolute position limit for Round 1 will be 40, Round 2 with 60 and Round 3 with 100.

There will be a $1 fixed bid-ask spread around the price of each security throughout the entire competition, so the starting bid ask price for all securities will be 99.5, 100.5, respectively.

Participants will be given sample data generated from our algorithm to test run their program, but we may change the parameters used on the competition day. You will be provided with adequate capital at the beginning of each round to establish positions within the limits set forth above. Positions and PnL are not carried over between rounds. Any open positions will be liquidated at the closing price of each security at the end of each round. Teams will have time between rounds to make changes to their parameters used in their algorithm if they choose.

3. Scoring

Participants' scores in each round of the case will be determined based on PnL, where PnL is defined to include profits and losses from closed positions, as well as profits and losses from any open positions at the end of the round marked to the closing price of each security of that round.

The weighting of the scores from each round to calculate the final score of the case will be announced prior to the start of the competition.

4. Case Objects & Interface

The following Java class objects are implemented in the util package and should be used in your program based on the interface defined:

```
public enum Ticker {
    HURON, SUPERIOR, MICHIGAN, ONTARIO, ERIE
}

public static class Quote {
    public Quote(Ticker ticker, double bid, double ask)
}

public enum OrderState {
    DEFAULT, FILLED, REJECTED
}

public static class Order {
    public Order(Ticker ticker, int quantity, OrderState state)
}
```

Within `PairsUtil.java`, the following helper function is available to you:

```
public static Order[] initiateOrders(Ticker[] symbols)
```

Your program needs to expose the following functions:

```
public void currentSymbols(Ticker[] symbols);
```

This method will be called at the beginning of each round and an array of Tickers used for that round will be passed down to you through the argument. You should use `initiateOrders(Ticker[] symbols)` provided in the util to initiate the array of orders that will be returned to the system.

```
public Order[] priceUpdate(Quote[] quotes);
```

This method is called when new quotes for the stocks are distributed. New price information will be passed on via the quotes array argument. You implementation should return an array of orders indicating your actions for each of the stocks using the `quantity` variable of the `order`object. 1 for buy, -1 for sell and 0 for inaction.

```
public void ordersConfirmation(Order[] orders);
```

This method is invoked to confirm the orders you submitted were filled. The same array of `order` that you submit to the system is returned to you with the updated `OrderState`. If the order is rejected, you should check your position/limit.

Note that although you can always expect the order of the tickers in `Quote[]` to be

HURON, SUPERIOR for Round 1,

HURON, SUPERIOR, MICHIGAN for Round 2 and

HURON, SUPERIOR, MICHIGAN, ONTARIO, ERIE for Round 3,

please double check the Ticker before you proceed with your algorithm and in your `Order[]`, please return them in the SAME order as the `Quote[]`.