Thank you for your interest!

The purpose of the following task is to test your understanding of the requirements, programming skills, and common sense. We also hope you will enjoy it.

The task is to implement a simple exchange which can process orders of buyers and sellers (in other words, traders). The task is self contained and doesn't require any additional / external background in finance or in trading.

General task requirements and instructions

- Avoid usage of external libraries unless it's really necessary and beneficial
- No need to handle errors unless specifically required
- Avoid over-engineering and over-generalization of the solution
- Performance of all methods required by the task matters. However, if a significant performance boost requires much more effort or would make the solution much more complex, you can only describe it without its actual implementation.
- Use Java 8
- Email your solution to hr@bookmap.com. Please include:
    - The link to the corresponding job posting
    - Source code of your solution (ideally, a link to an online repository)
    - You may also include a cover letter, description / explanation of your solution, how did you test it, etc.
- Anything that isn't covered by the requirements is up to you to decide how to implement, so please consider carefully the alternatives. Also, providing a list of your assumptions along with the task may help us to examine your solution. For example (though our task has no relation to image processing): "I assumed that input file is a valid JPEG image with width and height in range of 200-500 px".

Task details

Exchange is a centralized trading venue which processes buy and sell orders sent by traders.

Order

An order is a set of parameters:

- **Side**: Either Buy or Sell
- **Price**: This instructs the exchange not to execute the order at a worse price than this
- **Size**: Amount of units to buy or sell (e.g. barrels of oil, apples, or tons of gold).

Matching orders

We say that two orders are matching when they have opposite Buy/Sell side and the price of the Buy order is higher than, or equal to, the price of the Sell order. When this happens, the order
with smaller size gets fully executed (its size becomes zero) while another (with larger or equal size) is at least partially executed.

Resting orders

An order with a size greater than zero that doesn't match any of other resting orders is called a resting order. Such orders are kept by the exchange for their potential match with future orders.

Processing of incoming orders

New orders of traders are processed strictly according to the order of their arrival to the exchange. When a new order arrives, the exchange attempts to execute it immediately as long as it matches any of the resting orders. The matching must be done according to price priority, i.e. starting from the best price from the perspective of the newly arrived order. When there are multiple resting orders at the best price they should be processed in the order they were placed.

Cancellation

The exchange allows traders to cancel their resting orders

TODO

- Create a package called **exchangetask** with the 2 interfaces below and a class RequestRejectedException that extends java.lang.Exception.
- Add to the package a class named **Exchange** which implements both interfaces and processes orders according to the description above. Decide which actions should or shouldn't be rejected and throw RequestRejectedException accordingly.

```
public interface ExchangeInterface {
        public void send(long orderId, boolean isBuy, int price, int size) throws  RequestRejectedException;
        public void cancel(long orderId) throws RequestRejectedException;
}
```

```
public interface QueryInterface {
        // Return sum of sizes of resting orders at <price> or zero
        public int getTotalSizeAtPrice(int price);
        // Return the highest price with at least one resting Buy order
        public int getHighestBuyPrice();
        // Return the lowest price with at least one resting Sell order
        public int getLowestSellPrice();
}
```

Bonus task (Optionally)

Implement the following interface which allows traders to modify price and/or size of their working orders.

```
public interface  AdvancedExchangeInterface  extends  ExchangeInterface  {
        public void modify(long oid, int newPrice, int newSize,
                  boolean keepPositionOnSizeDecrease) throws RequestRejectedException;
}
```

If keepPositionOnSizeDecrease is true and newPrice is equal to old price and newSize is smaller than the old size of the resting order, then the order should be kept at the same position in the queue at the given price. Otherwise modified order should 'lose' the position and be placed at the end of the queue at the given price just as if it was a new order.

Your solution will be tested with a semi-automated test suit which generates orders with unique <orderId> parameter. We may reject a solution that requires too many manual adjustments to be tested.