Background

Thank you for your interest!

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

This is a practical task which requires basic Java skills. The task is to implement a simple exchange which can receive and process actions of buyers and sellers (in other word, traders) such as sending new orders and canceling or modifying their previously sent orders. The task is self contained so you don't need any additional / external background in finance or in trading.

Hiring procedure

Applicants who will submit a high quality solution to the following task, will be invited to a technical interview (typically via a skype call). Those who successfully pass the I technical interview will go through an administrative interview and salary negotiation. We may also offer you to perform another task before progressing. In that case it will be a paid task.

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 matters. However, if you feel that a significant performance boost requires much more effort to implement or would make the solution much more complex, you can describe it in comments or in a doc/email without its actual implementation.
- Use Java 8
- Email your solution to hr@bookmap.com. Please include:
    - The link to the corresponding job posting
    - Your solution in a form of either a zip file, a link to the zip file, or 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. But it's best to provide the list of your assumptions along with the task. 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".
- You are also welcome to email your questions. However, please note that in some cases the questions may also affect the score of your solution in both directions.

The Task

Exchange is a centralized trading venue which processes buy and sell orders and other actions (such as request to cancel a previously sent order) that it receives from traders. Actions are processed strictly according to the order of their arrival to the exchange. Some orders can be matched instantly. Others are kept by the exchange for potential future match.

Order

An order is a set of parameters:

- **Side**: Either Buy or Sell
- **Price**: This instructs the exchange that the order must not be executed at a worse price than specified
- **Size**: This tells how many units of the asset to buy or sell (e.g. barrels of oil, apples, or tons of gold).

Orders processing algorithm

We say that two orders are matched (or executed) 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 (i.e. its size becomes zero) while another (with larger or equal size) is at least partially executed. Orders that can't be instantly matched and have size greater than zero are kept by the exchange for their potential execution in future. We call such orders **resting orders** (or **working orders**). When a new order arrives to the exchange, it may be matched with more than one resting order. The matching priority must be according to the best available price (from the perspective of the newly arrived order). Also, traders can request to cancel or modify their working orders.

TODO

Implement a class named **Exchange** located in **exchangetask** package which implements the following interfaces (also located in exchangetask package) and processes incoming actions of traders as described above. Reject cancellation requests of non-resting orders by throwing corresponding exception. Assume that each order has its unique Order ID assigned by some preprocessing module.

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

In order to test the integrity of your solution please also implement the following query interface. If you created your own test suit, you may also attach it.

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

Bonus task (optionally)

Implement the following interface which allows traders to modify price and/or size of their working orders. Reject modification requests of non-resting orders.

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