In this assignment – assignment ORDER_BOOK – you will implement the exchange side of a simplified order book. Most exchanges support many different types of order related messages, but, for the sake of example, we will keep things simple.

The main object in our system is an object of class Exchange. You will write the Exchange class, as well as others. An exchange has multiple order books, one for each ticker. The tickers – “IBM”, “PFE”, “DELL” – represent stocks, for each of which the exchange maintains an order book.

Each order book looks like the presentation that I showed in class. (See short presentation previously uploaded to the group site.)

The method through which an exchange receives messages from its clients is

```
public void receiveMessage( OrderMessage orderMessage ) {…}
```

In the real world, messages would not be sent to an exchange object directly – the exchange is not running on the same computer as your code – but would be read from some wired or wireless connection. Some kind of translator object would un-pack the raw data of a message, instantiate the appropriate object and call a method similar to the above.

We want our exchange to support three types of incoming messages:

Add

- Fields
  - clientId – This id uniquely identifies the client and allows the exchange to send messages back to the client.
  - clientMessageId – This id is used by the client to uniquely identify this message. When the exchange sends back a response to this message, it will use this id.
  - ticker – A ticker such as “IBM”, “MSFT”, etc.
  - side – Long or short
  - quantity – A positive integer
  - price – The highest (lowest) price for buy orders (sell orders) at which the client wants an execution
  - type – We will implement three types of orders
    - Ordinary limit order
    - Fill or kill (FOK) orders require that the order be immediately filled in its entirety. If this is not possible, the order is cancelled.
    - Immediate or cancel (IOC) orders require that any part of an order that can be filled immediately is filled, and any remaining shares are cancelled.
- Purpose – This message places a new order into the order book. If any of the parameters are wrong, the exchange should send out a Rejected message. (See below)

Cancel

- Fields
  - clientId – An id that uniquely identifies the client
  - exchangeOrderId – The id of an existing order, an order that is supposed to reside somewhere in the order book. The exchange previously gave this id to the client when it sent the client an Accepted message in response to the client's Add

message.

- Purpose – If an order is unfilled or partially unfilled, the remainder of the order can be cancelled using this message. If the order doesn't exist or has been fully filled or previously cancelled, the exchange should send out a Rejected message.

Replace

- Fields
  - clientId – An id that uniquely identifies the client
  - exchangeOrderId – This is the id of an existing order, an order that is supposed to reside somewhere in the order book
  - newQuantity – Clients can reduce (but not increase) the quantity of an order that has not been executed
- Purpose – This message allows clients of the exchange to reduce the quantity of an existing order. If the order doesn't exist because the id is erroneous or the order has been fully filled or previously cancelled, the exchange should send out a Rejected message. (See below.)

Our exchange will support three types of messages to clients (outgoing messages):

Rejected

- Fields
  - clientId – This id identifies the client to whom the order is being sent.
  - clientMessageId – This is the id that allows the client to uniquely identify the message on the client's side.
  - description – Write a human readable description of why the message was rejected.
- Purpose – There is something wrong with the message to the exchange to which this message refers via its clientMessageId field.

Accepted

- Fields
  - clientId – This id identifies the client to whom the order is being sent.
  - exchangeOrderId – If the client message to which this message is a response resulted in the creation of a new order, this field uniquely identifies that order on the exchange side.
  - clientMessageId – This is the id that allows the client to uniquely identify the message on the client's side
- Purpose – This message tells a client that the message referenced by clientMessageId has been accepted. Additionally, if that message resulted in a new order, the client is told the order's id on the exchange side, exchangeOrderId.

Fill

- Fields
  - quantity – Quantity of the order that has been filled when another order matched with it.
  - exchangeOrderId – This is an id that uniquely identifies the order on the exchange side.
- Purpose – This message reports a fill or partial fill to a client. Remember, there are two sides to a transaction – the order that was in the book and the order that matched with the order that was in the book. A report must be sent to both parties. They both receive a

fill.

You will not be required to implement client side processing of messages from the exchange. You can just assume that the exchange has a method called:

```
public void sendToClient( MessageFromExchange msgFromEchange ) {…}
```

There is one difference between the presentation and the order types listed above. As do many exchanges, our exchange has no market orders. We have only marketable limit orders, which is to say that if we set our price high enough (low enough) for buy orders (sell orders) they will match with all existing sell orders (buy orders) already in the book.

On the exchange side, you will have to use your knowledge of data structures to implement order books in the most efficient manner possible. You now know about data structures such as ArrayList, LinkedList, TreeMap, HashMap, and others. Use these to good effect.

For example, when an Add message gets to an exchange, it should NOT simply walk through the long, long list of resting orders making decisions about what it should or shouldn't do. That is horribly inefficient. The same is true of Cancel messages: they should quickly and efficiently find the appropriate resting orders and remove them from the order book.

The deliverable for this assignment is... you guessed it, a set of unit tests. You should demonstrate that your code works – and all of its components work – as expected, and implement all of the functionality described above.

Note that market impact is not part of this assignment. Meaning, you are not required to implement a market impact model.