Order book project

# Application Overview

## Functionality:

All functionality from design instructions are implemented

## Unit tests:

94 unit tests passing. 0 failing. (see unitTest.xls document)

# Design:

## Classes

* OrderBookManagerImpl
* OrderBook
* OrderLinkedList, OrderNode
* InstrumentProperty, LevelProperty

## OrderBookManagerImpl

The OrderBookManagerImpl contains two hashmaps, askBookHashMap and bidBookHashMap that acts as a lookup to find different books depending on the side and the instrument. The two hashmaps are used to store the different books. When orders are added, the hashmaps either return the appropriate book or create a new entry and book for the order.

A further hashmap, orderHashMap is used to store a String and Order, where the String is the orderId to act as a lookup function for the order. The orderHashMap is used for many lookupFunctions to return items such as the order price, volume and instrument when the orderId is passed in. A lot of the lookup get requests use a string as an argument so the order hashMap is most useful for this.

A final hashmap, InstrumentPropertyMap is used to store instrument-level properties. The instrumentPropertyMap returns an InstrumentProperty class instance when the instrument is passes as a key. The InstrumentProperty class contains information about the instrument and side as a whole. The information includes the different levels that exist for the given instrument and side. The InstrumentProperty class provides a lookup into the individual levels through the LevelProperty class. LevelProperty provides all the details for the instrument, side, and level (price) that may be queried.

## OrderBook

The OrderBook is used for adding, removing, and modifying orders. The OrderBook extends from TreeMap<Long, OrderLinkedList> to ensure the order books are always kept in a sorted order based upon price (Long). The bid book differs from the ask book in one way: when the one-argument constructor is called, it will call the super(Comparator) constructor where the passed in argument is Comparator.reverseOrder. The constructor will ensure the BidBook orders the levels with the highest price first. See createNewBookForOrdermethod.

## OrderLinkedList, OrderNode

The BidBook and AskBook classes store the value of the treeMap as an OrderLinkedList instance. Each instance relates to a single price, of a single instrument, and a single side. The OrderLinkedList class is my implementation of a LinkedList to store the Orders. The OrderLinkedList contains OrderNode instances, which are classes that store the Order, next Order, and previous Order in the LinkedList.

By using a LinkedList adaptation, I am able to achieve constant time adding to the end of the LinkedList. I am also able to delete orders within the chain without having to update all the other orders.

## InstrumentProperty, LevelProperty

The InstrumentProperty class is used to return values for the get requests for each instrument. The InstrumentProperty class will return the levels for the instrument in the respective side. The LevelProperty class stores a further granularity of information relating to the level of each instrument and side. LevelProperty is used to group the data from different price levels for each instrument and side.

## Time and Space Complexity

Below is a table to define the time and space complexity of the key operations:

The operations of the application perform at constant time complexity with the exception of the modifyOrder, deleteOrder, and getOrdersAtLevel methods. The three methods have a complexity that is equal to the number of Orders at the level it is searching on. The non-constant time complexity is created due to the LinkedList style of storing and ordering Orders at each level.

## Improvements

I would implement exception handling into the application to gracefully handle null pointer exceptions.

The