**Pricer project by Timothy Denisenko**

This project is an example of implementation of simple Order Book and its functions in Java.

Compiled and tested in: Eclipse IDE for Java Developers Version: Kepler Release Build id: 20130614-0229

This project has 3 classes: Pricer.java, OrderBook.java and Test.java

**Update 28 Aug 2013**

* Implemented new order types: Market order, Limit order, Reduce order, Cancel order, Hidden order, IOC and FOK
* Fixed some comparison bugs
* Made the project compatible with "pricer.in" file again
* Added a recursive alphabetic sequence generetor (it has performance issues at large inputs, will reconsider coding a prefixed sequence generator)
* Contructors and input formats are changed according to new order types
* Class descriptions updated

**Pricer.java**

This class is a data structure for storing individual orders in a class. It accepts different kinds of order formats.

**Fields:**

* long timestamp:

Time passed since midnight (00:00) in milliseconds

* String message:

Type of the order

L for Limit, M for Market, R for Reduce, C for Cancel, I for IOC, F for FOK and H for Hidden orders

* static int NUMBER\_OF\_ORDERS:

Number of orders created

* String orderID:

An alphanumeric ID either automatically generated or entered manually

* char side:

'S' for sell(ask) 'B' for buy(bid)

* double price:

Price with a precision of .00

* int size:

Size of the order

**Formats:**

* Limit order, IOC, FOK, Hidden:

Pricer(String message, char side, double price, int size)

Ex.: "F S 33.10 100"

* Reduce order:

Pricer(String message, String orderID, int size)

Ex.: "R gfe 42"

* Cancel order:

Pricer(String message, String orderID)

Ex.: "C rye"

* Market order:

Pricer(String message, char side, int size)

Ex.: "M B 200"

**OrderBook.java**

This class is a data structure for storing Pricer objects together in a class.   
It is sorted (by price and by timestamps when the prices are the same) and synchronized (takes an order at a time) anytime.  
It prints the Order Book once every 100 orders has been processed.  
This class includes 2 main ArrayLists, ListBuy (Bid orders) and ListSell (Ask orders) and 2 sub Arraylists ListTempBuy and ListTempSell (for FOK order structure)  
It adds the order to a list when an object of it has created.  
For ex. OrderBook o = new OrderBook("M B 200");

**Test.java**

This class tests orders from a given file (pricer.in) that includes over 1 million orders.

**Functions of Pricer Project**

**Limit order and Hidden order (adds order to list):**

1. Sell order with a smaller size than the highest buy order. Solution: Remove the sell order and reduce the size of the highest buy order.
2. Sell order with the same size as the highest buy order. Solution: Remove both lowest sell and highest buy order.
3. Sell order with bigger size than the highest buy order. Solution: Decrease the size of sell order by the highest buy order, remove the highest buy order, iterate from the first case.
4. Sell order with a higher price than the highest buy order. Solution: Just add the sell order to the Order Book.
5. Buy order with a smaller size than the lowest sell order. Solution: Remove the buy order and reduce the size of the lowest sell order.
6. Buy order with the same size as the lowest sell order. Solution: Remove both lowest sell and highest buy order.
7. Buy order with bigger size than the lowest sell order. Solution: Decrease the size of buy order by the lowest sell order, remove the lowest sell order, iterate from the first case.
8. Buy order with a lower price than the lowest buy order. Solution: Just add the buy order to the Order Book.

**Market order:**

1. Sell order with a smaller size than the highest buy order. Solution: Reduce the size of the highest buy order.
2. Sell order with the same size as the highest buy order. Solution: Remove the highest buy order.
3. Sell order with bigger size than the highest buy order. Solution: Decrease the size of sell order by the highest buy order, remove the highest buy order, iterate from the first case.
4. Buy order with a smaller size than the lowest sell order. Solution: Reduce the size of the lowest sell order.
5. Buy order with the same size as the lowest sell order. Solution: Remove lowest sell order.
6. Buy order with bigger size than the lowest sell order. Solution: Decrease the size of buy order by the lowest sell order, remove the lowest sell order, iterate from the first case.

**IOC:**

1. Sell order with a smaller size than the highest buy order. Solution: Reduce the size of the highest buy order.
2. Sell order with the same size as the highest buy order. Solution: Remove highest buy order.
3. Sell order with bigger size than the highest buy order. Solution: Decrease the size of sell order by the highest buy order, remove the highest buy order, iterate from the first case.
4. Sell order with a higher price than the highest buy order. Solution: Do nothing.
5. Buy order with a smaller size than the lowest sell order. Solution: Reduce the size of the lowest sell order.
6. Buy order with the same size as the lowest sell order. Solution: Remove lowest sell order.
7. Buy order with bigger size than the lowest sell order. Solution: Decrease the size of buy order by the lowest sell order, remove the lowest sell order, iterate from the first case.
8. Buy order with a lower price than the lowest buy order. Solution: Do nothing.

**FOK:**

1. Sell order with a smaller size than the highest buy order. Solution: Reduce the size of the highest buy order.
2. Sell order with the same size as the highest buy order. Solution: Remove highest buy order.
3. Sell order with bigger size than the highest buy order. Solution: Decrease the size of sell order by the highest buy order, remove the highest buy order, put the removed element to temporary list (listTempBuy), iterate from the first case.
4. Sell order with a higher price than the highest buy order. Solution: Reverse the process by adding all elements from temporary list (listTempBuy) to buy list (listBuy) back, clear the temporary list.
5. Buy order with a smaller size than the lowest sell order. Solution: Reduce the size of the lowest sell order.
6. Buy order with the same size as the lowest sell order. Solution: Remove lowest sell order.
7. Buy order with bigger size than the lowest sell order. Solution: Decrease the size of buy order by the lowest sell order, remove the lowest sell order, put the removed element to temporary list (listTempSell), iterate from the first case.
8. Buy order with a lower price than the lowest buy order. Solution: Reverse the process by adding all elements from temporary list (listTempSell) to sell list (listSell) back, clear the temporary list.

**Reduce order:**

1. Reduce order for sell orders. Solution: Remove order from the Order Book if reduce order's size is equal to or bigger than the order size, reduce the size otherwise.
2. Reduce order for buy orders. Solution: Remove order from the Order Book if reduce order's size is equal to or bigger than the order size, reduce the size otherwise.

**Cancel order:**

1. Cancel the specified order with orderID.
2. Error message if order with given orderID does not exist.