

# Assignment 3

COMP 249, Winter 2016

Due: Wednesday, April 6, 23:55

Teams: Single student or two students from same section

---

## Updates:

*Correction 1 (May 26): Correction to graphic of transaction Version 4. In the original order book, the best offer in the book should have volume 500.*

*Correction 2 (May 26): Correction to the sample run in Part 1. The second call to `book.outputBook()`; should insert the remaining volume (125 shares) as a new offer not a new bid. Thus the next line, `book.outputBBO()`; also changes in reflection of this.*

*Other potential mistakes? Please email [j.clark@concordia.ca](mailto:j.clark@concordia.ca)*

---

**Summary.** In this assignment, you will be creating a simplified version of a stock market. When Alice wants to buy some shares of a stock, she consults an *order book* which contains a list of people willing to buy and sell the stock at a variety of prices sorted from highest price to lowest price. If she sees a price that she is interested in, her request to buy is matched to that person's offer to sell and the trade goes through. If instead the prices are all too high, she can insert her own offer to buy at a certain price into the book. Later someone else may come along and sell to her at her set price. The algorithm that matches buy and sell orders is called a *matching engine*. In the first part of this assignment, you will write an order book object and a matching engine. In the second part, you will create a GUI interface for specifying the details of the order. Although the two parts are thematically related, you do not need to complete them in any particular order: they are independent of each other.

## Part 1: Order Book and Matching Engine

When Alice wants to buy some stocks in a particular company, she creates an order called a *bid* or *buy*. A bid contains her ID, the price she is willing to pay, and the number of stocks (called the *volume*) she wants to buy at that price. When Alice wants to sell some stocks, she creates an order called an *offer*, *sell*, or an *ask*. It is otherwise the same as a bid: it contains her ID, price, and volume. (Orders of these types are called *limit orders* to distinguish them from other more complicated types of orders that are possible but will not be considered in this assignment.)

Alice submits her order (whether a bid or offer) to the matching engine, and the matching engine consults its order book for that company's stock to see if it can match Alice's order to someone else's order. If it can't, it puts Alice's order in the book.

To illustrate this and the possible corner cases, consider the following examples. Each example starts with the *same* original order book, sorted by price (from highest price to lowest) and shows how it is updated depending on which of 6 different examples of orders come in.

Order Version 1 (see Figure): The new bid is between the best bid and the best offer. Thus it does not match any existing offers in the book. It is inserted as a new bid in the book and it becomes the *best bid* (the highest price in the book someone is willing to pay for the stock).

*See next page for examples...*

Original Order Book			Updated Order Book		
<i>Type</i>	<i>Price</i>	<i>Volume</i>	<i>Type</i>	<i>Price</i>	<i>Volume</i>
Offer	155.00	300	Offer	155.00	300
Offer	152.50	120	Offer	152.50	120
Offer (Best)	152.00	100	Offer (Best)	152.00	100
Bid (Best)	148.00	75	Bid (Best)	150.00	500
Bid	147.00	200	Bid	148.00	75
Bid	146.60	100	Bid	147.00	200
Bid	146.50	50	Bid	146.60	100
New Order — Version 1			Bid	146.50	50
<i>Type</i>	<i>Price</i>	<i>Volume</i>			
Bid	150.00	500			

Original Order Book			Updated Order Book		
<i>Type</i>	<i>Price</i>	<i>Volume</i>	<i>Type</i>	<i>Price</i>	<i>Volume</i>
Offer	155.00	300	Offer	155.00	300
Offer	152.50	120	Offer	152.50	120
Offer (Best)	152.00	100	Offer	152.00	100
Bid (Best)	148.00	75	Offer (Best)	151.00	500
Bid	147.00	200	Bid (Best)	148.00	75
Bid	146.60	100	Bid	147.00	200
Bid	146.50	50	Bid	146.60	100
New Order — Version 2			Bid	146.50	50
<i>Type</i>	<i>Price</i>	<i>Volume</i>			
Offer	151.00	500			

Original Order Book			Updated Order Book		
<i>Type</i>	<i>Price</i>	<i>Volume</i>	<i>Type</i>	<i>Price</i>	<i>Volume</i>
Offer	155.00	300	Offer	155.00	300
Offer	152.50	120	Offer (Best)	152.50	120
Offer (Best)	152.00	100	Bid (Best)	150.00	500
Bid (Best)	148.00	75	Bid	148.00	75
Bid	147.00	200	Bid	147.00	200
Bid	146.60	100	Bid	146.60	100
Bid	146.50	50	Bid	146.50	50
New Order — Version 3					
<i>Type</i>	<i>Price</i>	<i>Volume</i>			
Bid	152.10	100			

Original Order Book			Updated Order Book		
<i>Type</i>	<i>Price</i>	<i>Volume</i>	<i>Type</i>	<i>Price</i>	<i>Volume</i>
Offer	155.00	300	Offer	155.00	300
Offer	152.50	120	Offer (Best)	152.50	120
Offer (Best)	152.00	500	Bid (Best)	152.10	400
Bid (Best)	148.00	75	Bid	150.00	500
Bid	147.00	200	Bid	148.00	75
Bid	146.60	100	Bid	147.00	200
Bid	146.50	50	Bid	146.60	100
New Order — Version 4			Bid	146.50	50
<i>Type</i>	<i>Price</i>	<i>Volume</i>			
Bid	152.10	500			

Original Order Book			Updated Order Book		
<i>Type</i>	<i>Price</i>	<i>Volume</i>	<i>Type</i>	<i>Price</i>	<i>Volume</i>
Offer	155.00	300	Offer	155.00	300
Offer	152.50	120	Offer	152.50	120
Offer (Best)	152.00	100	Offer	152.00	100
Bid (Best)	148.00	75	Offer (Best)	140	75
Bid	147.00	200			
Bid	146.60	100			
Bid	146.50	50			
New Order — Version 5					
<i>Type</i>	<i>Price</i>	<i>Volume</i>			
Offer	140	500			

Original Order Book			Updated Order Book		
<i>Type</i>	<i>Price</i>	<i>Volume</i>	<i>Type</i>	<i>Price</i>	<i>Volume</i>
Offer	155.00	300	Offer	155.00	300
Offer	152.50	120	Offer	152.50	200
Offer (Best)	152.00	100	Offer	152.50	120
Bid (Best)	148.00	75	Offer (Best)	152.00	100
Bid	147.00	200	Bid (Best)	150.00	500
Bid	146.60	100	Bid	148.00	75
Bid	146.50	50	Bid	147.00	200
New Order — Version 6			Bid	146.60	100
<i>Type</i>	<i>Price</i>	<i>Volume</i>	Bid	146.50	50
Offer	152.50	200			

Order Version 2: Similarly, if a new offer comes in instead, but is still between the best bid and best offer, it does not match any bids and is inserted into the book. It becomes the best offer (the lowest price in the book someone is willing to sell the stock for).

Order Version 3: Consider instead a bid of \$152.10 for 100 shares. Someone in the book is offering 100 shares for sale at \$152.00. Thus these buyers and sellers are matched together, and the 100 shares are sold (the exchange keeps the difference in price). The updated book removes the offer at \$152.00 and does not insert anything about the new bid since both are taken care of fully.

Order Version 4: Consider what happens if the bid for Version 3 was for 500 shares instead of 100 shares. The bid still matches to the offer of 100 shares at \$152.00 and these 100 shares trade. However the buyers still wants another 400 ( $500 - 100 = 400$ ) shares at \$152.10 and there is no one willing to sell for that low of a price. Thus the book is updated as follows (1) the matched offer of 100 @ \$152.00 is removed from the book, (2) the bid for the remaining 400 @ \$152.00 is added to the book, and (3) the best bid is updated.

Order Version 5: Consider now what happens if a very large order comes in at a very good price: in this case, a large sell order at a very low price. The offer of 500 shares at \$140.00 matches to more than one bid. The best bid is 75 shares @ \$148 and these trade. However the order still has 425 shares left to sell. The next best bid is 200 shares @ \$147.00. Since \$147 is still greater than \$140, then these 200 shares trade. In general, this procedure continues until either (a) the sell order is fully sold (in this case, nothing new is added to the book, the matched bids are removed from the book, and if the last bid to match wants more shares than the offer has left to sell, it remains in the book with a lower volume) or (b) all the shares for greater than or equal to \$140 have been bought and there is still left over shares to be sold at \$140 (in this case, the remaining volume is added to the book as the new best offer — this is what happens in Version 5).

Order Version 6: In this case, an offer comes in that is the exact same price as another offer in the book. When two offers have the same price, the first offer in the book should be traded first (this is called price-time priority). The convention when matching a new bid to the existing offers is to start with the best offer and execute as many trades as possible moving from the best offer to the outside of the book. Thus new orders at the same price as existing orders should be placed in the book further from the middle of the book. Concretely this means a new offer at the same price is placed immediately above the other offers at the same price, and a new bid at the same price is placed immediately below any other bids at the same price.

## Implementation

This project has an open design, meaning we will not tell you all the specifics about what objects and methods to implement. Instead, we will give you a set of requirements and you can have flexibility in how you meet the requirements. Your goal is to produce a working implementation.

**Orders:** Create a parent `Order` object that is not specifically a bid or offer but contains all the common members a bid and an offer would have. Then create a `BidOrder` object and an `OfferOrder` object, both which extend the parent offer object (depending on your design, they may not add any new instance variables but will allow you to tell the types of orders apart using `instance of` and/or `getClass()`). An order of either type should have a string `ID` for the person submitting the order, a double `price`, and an int `volume`. They should have appropriate getters and setters for the variables, a `toString` method (which we will return to below when discussing the `anonymous` interface), and other standard methods. In this assignment, we will assume there is only one stock being traded and so we will not include it as a variable in any `Order` object.

**Order Book:** Create an `OrderBook` object as a linked data structure. Specifically, it should be a doubly linked list where unmatched `Order` objects can be inserted, and the list is always sorted from largest price to smallest price. The nodes of the list should be implemented as an inner class within the order book object. Because of the way orders are matched, the resulting list will always have all the offers at the start of the list and then all the bids being at the end of the list.

While every order in the book is important, two orders of particular importance. They are called the *best bid* and *best offer*. The best bid is the bid with the largest price and the best offer is the offer with the lowest price. These two orders will be beside each other, somewhere in the middle of the order book. The difference in price between the best bid and best order is called the *spread*. For the doubly linked list in the `OrderBook` object, always maintain a pointer to the `BestBid` node and the `BestOffer` node. This data structure will allow you to jump directly to either the best bid or the best offer and walk through the book in either direction.

Additionally, your `OrderBook` object should implement the following methods:

- `public void outputBook():` print out the `OrderBook` (note interface requirements below)
- `public void outputBBO():` print out the best bid and offer (BBO)

**Iterators:** You may find it convenient to create an iterator object for your `OrderBook`. This is optional.

**Interfaces:** Order books are generally public and visible to any trader at any time (sometimes order books are kept private and this called a *dark pool* but we will consider open order books only in this assignment). However the names of the individuals with orders in the public is kept private until a trade is actually executed — then their identities are revealed because buying a share makes you an owner in the company and the company needs to know who you are. In order to ensure a coding convention that recognizes that names are not to be revealed until after a trade is complete, create a new interface called `anonymous`. Each order object and the order book itself should implement `anonymous`. The `anonymous` interface does two things: (1) it advises that `toString()` should never print out the names in the orders and (2) it requires that a second method, `printFullDetails()` be implemented that will print out the name, along with the other details. You can set the return type of `printFullDetails()` to whatever is most convenient to your implementation. It can return a `String` like `toString()` to be output by the calling object, or it can directly print the details to `System.out` itself and return `void`.

**Matching Engine:** The matching engine can be implemented in your main method. Optionally, you can make it a helper method or you can make it its own object.

The matching engine should take a `BidOrder` or `OfferOrder` object, attempt to match it to the orders in the book and report on any resulting trades.

Consider a `OfferOrder` (the `BidOrder` case is similar). The matching algorithm will go through the following steps:

- I. Is the offer price lower than the BestBid? Yes.
  - A. Match the order to the best bid, the second best bid, etc. until:
    1. The full volume of the OfferOrder has been satisfied. The last bid to match to the offer may still exist in the book but with a lower volume.
    2. No bids are left in the book. In this case, leave the remaining volume of the order as a new (best) offer.
    3. All the bids that are left in the book have a lower price than the offer. In this case, leave the remaining volume of the order as a new (best) offer.
  - B. For each matched order, report the trade (including identities of the buyers and seller).
  - C. Update the BestBid
- II. Is the offer price lower than the BestBid? No, or there are no bids in the book yet.
  - A. Is the offer price lower than the BestOffer? Yes.
    1. Set the offer as the new BestOffer.
  - B. Is the offer price lower than the BestOffer? No it is equal.
    1. For two (or more) offers at the same price, the order that comes first gets first priority. This means placing new orders closer to the outside of the book (further from the best bid/offer).
  - C. Is the offer price lower than the BestOffer? No it is greater.
    1. Find the right place in the book for insertion.
  - D. Is the offer price lower than the BestBid? No, there are no bids in the book yet.



1. Set the offer as the new BestOffer.

**Output:** At a minimum, you should print out the details (except the name) of a new order that is executed, and print out all of the orders from the order book that are matched (fully or partially) with the new order (print all details, including the name, in this case). While this is a minimum, you are free to print out a more verbose transcript of what happened when the new order came in — in fact, it will make debugging much easier.

A sample run is provided below. You do not have to match the format of the function calls or the sample output exactly, just provide something that is functionally equivalent. For example, in our implementation, we found it easiest to decrement the volume of the order as it matched each order in the book.

Code	Output
<code>// OrderBook object is called book book.outputBook();</code>	Order book: Off: 155.0 300 Off: 152.5 120 Off: 152.0 100 Bid: 148.0 75 Bid: 147.0 200 Bid: 146.6 100 Bid: 146.5 50
<code>Order o = new OfferOrder("Alice", 146.60, 500); execute(o,book);</code>	Matching new order: Off: 146.6 500  Match found: Off: 146.6 500 Alice Bid: 148.0 75 Feng  Match found: Off: 146.6 425 Alice Bid: 147.0 200 Bob  Match found: Off: 146.6 225 Alice Bid: 146.6 100 Ishana  Done.
<code>book.outputBook();</code>	Order book: Off: 155.0 300 Off: 152.5 120 Off: 152.0 100 Off: 146.6 125 Bid: 146.5 50
<code>book.outputBBO();</code>	Best Bid & Offer: Off: 146.6 125 Bid: 146.5 50

## Part 2: Order Creator GUI

In this section, you will create a GUI using Swing for entering orders. Specifically, you will design an interface with 3 inputs: price, volume and ID. The interface will have two buttons: submit and reset. And finally, the interface will have a status bar at the bottom. Note: the interface will not distinguish between Bid and Offers, as that would be most logically implemented with Radio buttons which are not discussed in COMP 249 (but are not too difficult if you want to try it on your own).

The interface does not have to exactly match the provided example (in terms of layout, colors, etc) but should have a fully functional design. Maintain three variables: double `price`, int `volume`, and String `name`. When proper information is entered and the user hits the submit button, you should read the information from the input fields into these variables. While you will not do anything in this assignment beyond print out the information on the status bar, storing this information in the proper variables will enable interaction with your `OrderBook` in Part I.

*See next page for examples...*

A screenshot of a macOS-style window titled "Order Generator". It contains three text input fields with placeholder text: "Enter price here.", "Enter volume here.", and "Enter name here.". Below the fields are two buttons, "Submit" and "Reset". At the bottom, a status bar displays "Last Order: none".

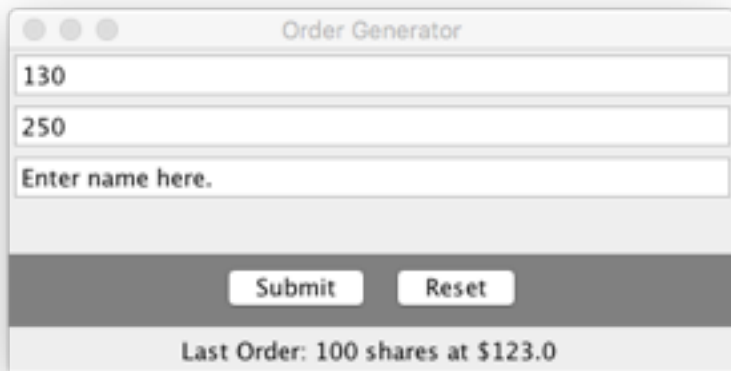
Opening interface prior to any text being entered.

A screenshot of the "Order Generator" window. The input fields now contain the text "123.00", "100", and "Jeremy Clark". The "Submit" and "Reset" buttons remain below the fields, and the status bar at the bottom still shows "Last Order: none".

After typing in the information and prior to pressing the submit button.

A screenshot of the "Order Generator" window after a submission. The input fields are now empty, showing the placeholder text again. The "Submit" button is highlighted with a blue border. The status bar at the bottom has been updated to show "Last Order: 100 shares at \$123.0".

After pressing the submit button. The interface is cleared, and the status bar at the bottom is updated. It does not include the name.



Order Generator

130

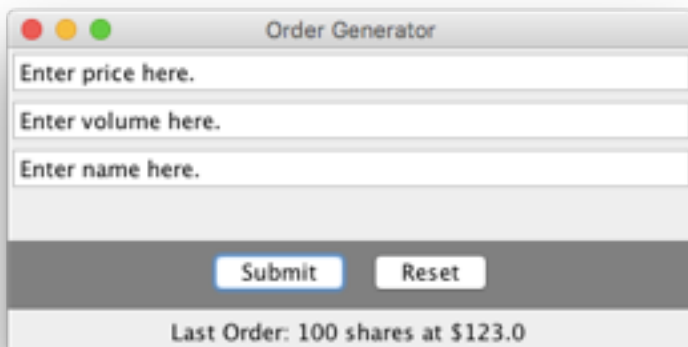
250

Enter name here.

Submit Reset

Last Order: 100 shares at \$123.0

Partial information is typed in and then the reset button is pressed leading to...



Order Generator

Enter price here.

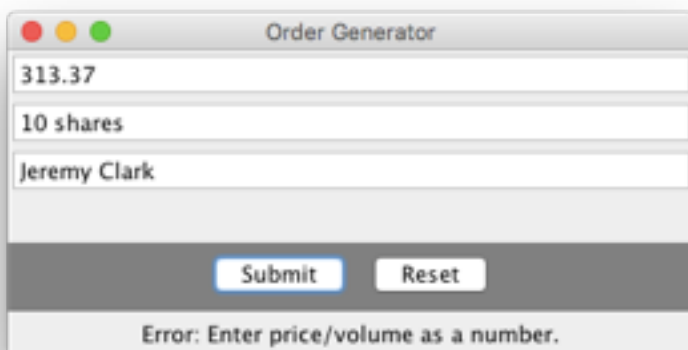
Enter volume here.

Enter name here.

Submit Reset

Last Order: 100 shares at \$123.0

... after pressing reset button



Order Generator

313.37

10 shares

Jeremy Clark

Submit Reset

Error: Enter price/volume as a number.

Entering non-numeric characters in either the price or volume field, and pressing submit, should produce the following error in the status bar.

## Evaluation

Question 1 (15 pts)	
Implementation of Class Order	2 pt
Implementation of Class OfferOrder	1 pts
Implementation of Class BidOrder	1 pts
Implementation of Linked list OrderBook	5 pts
Interface anonymous	1 pt
Matching Engine	5 pts

Question 2 (5 pts)	
Window	3 pts
Interface	2 pts

## Submission

Create one zip file containing all source files and Javadoc files for your assignment using the following naming convention:

- If the assignment is done by 1 student:  
The zip file should be called a#\_studentID, where # is the number of the assignment and studentID is your student ID number.
- If the assignment is done by 2 students:  
The zip file should be called a#\_studentID1\_studentID2, where # is the number of the assignment, and studentID1 and studentID2 are the student ID numbers of each student.

Check your section's course webpage for instructions on where and how to submit your assignment.