

## SE 350/450 - Assignment 2

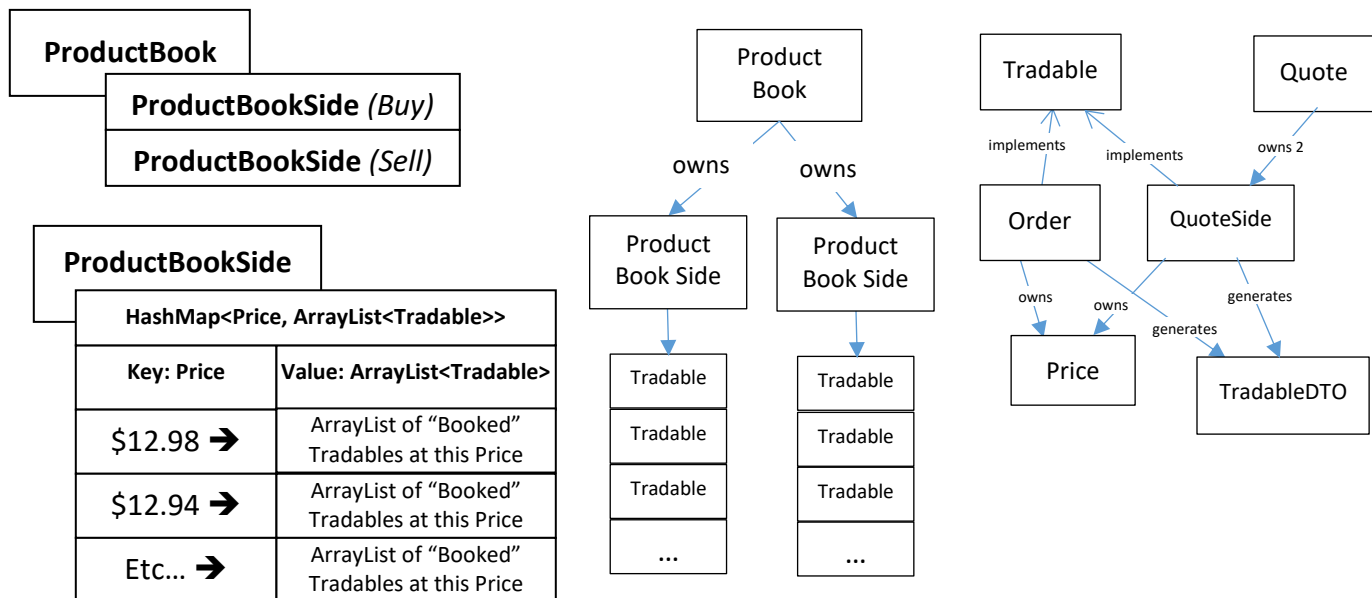
### Object-Oriented Coding

### Tradables & Product Books (200 pts)

#### Objectives

Create well-written object-oriented classes that represent a Product Book, Product Book Side, Order, and other supporting classes. You should also create any new exception classes if needed to handle invalid situations.

#### Classes and their relationships



#### 1) Tradable (interface)

The Tradable interface defines the behaviors that any object that can be traded must implement. Currently the implementers are Order and QuoteSide. These behaviors are as follows:

- String getId();
- int getRemainingVolume();
- void setCancelledVolume(int newVol);
- int getCancelledVolume();
- void setRemainingVolume(int newVol);
- TradableDTO makeTradableDTO();
- Price getPrice();
- void setFilledVolume(int newVol);
- int getFilledVolume();
- BookSide getSide();
- String getUser();
- String getProduct();
- int getOriginalVolume();

#### 2) Order class → implements Tradable

The Order class represents an order to buy or sell a volume (i.e., quantity) of stock at a certain price. The order will also maintain its processing state, detailed below.

- The Order class should have a public constructor that accepts and sets the parameters where indicated below.

- An Order must maintain certain data values – shown below (*remember to make these private to enforce information hiding*):
  - String user: A 3-letter user code – must be 3 letters, no spaces, no numbers, no special characters (i.e., XRF, BBT, KNT, etc.). This should be passed to the constructor. Cannot be changed once set.
  - String product: The stock symbol for the order (1 to 5 characters) – must from 1 to 5 letters/numbers, they can also have a period “.” In the symbol. Otherwise, -no spaces, no special characters (i.e., WMT, F, 3M, GOOGL, WOOF, AKO.A, etc.). This should be passed to the constructor. Cannot be changed once set.
  - Price price: A Price object – cannot be null. This should be passed to the constructor. Cannot be changed once set.
  - BookSide side: The “side” of the order – BUY or SELL. The BookSide type should be an “enum” with values BUY & SELL. Do not allow “side” to be null. This should be passed into the constructor. Cannot be changed once set.
  - int originalVolume: The quantity of the stock being ordered. <Must be greater than 0, and less than 10,000. This should be passed into the constructor. Cannot be changed once set.
  - int remainingVolume – This should be initialized to the originalVolume value. This value is changeable as the order is filled.
  - int cancelledVolume – This should be initialized to zero. This value is changeable when any part of the order is cancelled.
  - int filledVolume - This should be initialized to zero. This value is changeable when part of the order is filled.
  - String id: The tradable (order) id – unique to each order. This should *not* be passed into the constructor, it should be generated and set *in* the constructor. This should be generated as: user + product + price (String) + nanotime (from System.nanoTime()). Cannot be changed once set.  
Example: “XRF” + “WMT” + “\$139.59” + “157357272154300” = “XRFWMT\$139.59157357272154300”
- An Order must be able to perform certain behaviors – shown below. Many (but not all) of these are part of the Tradable interface.
- Create accessors and modifiers (that validate incoming parameters) for all private data members. *Modifiers for data attributes that should not be changed should be made private. Modifiers for data attributes that can be changed should be made private.*
- Override the *toString* method to generate a String like this:
  - **USER SIDE** order: **PRODUCT** at **PRICE**, Orig Vol: **ORIGINAL VOLUME**, Rem Vol: **REMAINING VOLUME**, Fill Vol: **FILLED VOLUME**, CXL Vol: **CANCELLED VOLUME**, ID: **ID**  
Example
  - **AXE SELL** order: **TGT** at **\$134.00**, Orig Vol: **50**, Rem Vol: **50**, Fill Vol: **0**, CXL Vol: **0**, ID: **AXETGT\$134.00506492572504400**
- public TradableDTO makeTradableDTO() ➔ Generates and returns a DTO object (described later) for the current order

### 3) Quote class (*does NOT implement Tradable*)

A Quote is made up of the BUY and SELL prices & quantities that a trader is willing to buy and sell a stock. The BUY part is referred to as the buy-side, the SELL referred to as the sell-side. The buy-side represents the price and quantity (referred to as the “volume”) at which the trader will buy the stock; the sell-side represents the price and quantity at which the trader will sell the stock.

- A Quote must maintain certain data values – shown below (*remember to make these private to enforce information hiding*):
  - String user: A 3-letter user code – must be 3 letters, no spaces, no numbers, no special characters (i.e., XRF, BBT, KNT, etc.). This should be passed to the constructor. Cannot be changed once set.
- String product: The stock symbol (1 to 5 characters) – must from 1 to 5 letters/numbers, they can also have a period “.” In the symbol. Otherwise, -no spaces, no special characters (i.e., WMT, F, 3M, GOOGL, WOOF, AKO.A, etc.). This should be passed to the constructor. Cannot be changed once set.
- QuoteSide buySide: A reference to the BUY side of this quote. This object is NOT passed in, it is **created** and set in the constructor. Cannot be changed once set.
- QuoteSide sellSide: A reference to the SELL side of this quote. This object is NOT passed in, it is **created** and set in the constructor. Cannot be changed once set.
- The Quote class should have a public constructor that accepts the following parameters:
  - String symbol, Price buyPrice, int buyVolume, Price sellPrice, int sellVolume, String userName
  - These parameters should be used in the constructor as follows:
    - The *symbol* and the *username* parameters should be used to set the *product* and *user* data fields.
    - The 2 QuoteSide references should be set by creating new QuoteSide objects. These should be created by passing these required parameters:
      - buySide: userName, symbol, buyPrice, buyVolume, BookSide.BUY
      - sellSide: userName, symbol, sellPrice, sellVolume, BookSide.SELL
  - The Quote class needs only these 3 public methods:
    - public QuoteSide getQuoteSide(BookSide sideIn): This method returns a reference to the specified QuoteSide (BUY or SELL side). *NOTE – Normally we would not return a reference to these objects, as they are owned by the Quote. However, as the QuoteSides are used widely across the application the creation of QuoteSide copies would quickly become unwieldy, this slight bending of the OO rules is preferable to the proliferation of unnecessary objects.*
    - public String getSymbol(): Return the quote’s *symbol* data value.
    - public String getUser(): Return the quote’s *user* data value.

#### 4) QuoteSide class → implements Tradable

The QuoteSide class represents one side of a quote - to buy or sell a volume (i.e., quantity) of stock at a certain price. The QuoteSide will also maintain its processing state, detailed below.

- The QuoteSide class should have a public constructor that accepts and sets the parameters where indicated below.
- A QuoteSide must maintain certain data values – shown below (*remember to make these private to enforce information hiding*):
  - String user: A 3-letter user code – must be 3 letters, no spaces, no numbers, no special characters (i.e., XRF, BBT, KNT, etc.). This should be passed to the constructor. Cannot be changed once set.
  - String product: The stock symbol for the quote side (1 to 5 characters) – must from 1 to 5 letters/numbers, they can also have a period “.” In the symbol. Otherwise, -no spaces, no special characters (i.e., WMT, F, 3M, GOOGL, WOOF, AKO.A, etc.). This should be passed to the constructor. Cannot be changed once set.

- Price price: A Price object – cannot be null. This should be passed to the constructor. Cannot be changed once set.
- int originalVolume: The quantity of the quote side. <Must be greater than 0, and less than 10,000. This should be passed into the constructor. Cannot be changed once set.
- int remainingVolume – This should be initialized to the originalVolume value. This value is changeable as the quote side is filled.
- int cancelledVolume – This should be initialized to zero. This value is changeable when any part of the quote side is cancelled.
- int filledVolume - This should be initialized to zero. This value is changeable when part of the quote side is filled.
- String id: The quote side id – unique to each quote side. This should *not* be passed into the constructor, it should be generated and set *in* the constructor. This should be generated as: user + product + price (String) + nanotime (from System.nanoTime()). Cannot be changed once set.  
Example: “XRF” + “WMT” + “\$139.59” + “157357272154300” = “XRFWMT\$139.59157357272154300”
- A QuoteSide must be able to perform certain behaviors – shown below. Many (but not all) of these are part of the Tradable interface.
  - Create accessors and modifiers (that validate incoming parameters) for all private data members. *Modifiers for data attributes that should not be changed should be made private. Modifiers for data attributes that can be changed should be made public.*
  - Override the toString method to generate a String like this:
    - **USER SIDE** side quote for **PRODUCT: PRICE**, Orig Vol: **ORIGINAL VOLUME**, Rem Vol: **REMAINING VOLUME**, Fill Vol: **FILLED VOLUME**, CXL Vol: **CANCELLED VOLUME**, ID: **ID**  
Example  
○ **AXE BUY** side quote for **TGT: \$134.00**, Orig Vol: **50**, Rem Vol: **50**, Fill Vol: **0**, CXL Vol: **0**, ID: **AXETGT\$134.00506729337603300**
- public TradableDTO makeTradableDTO() ➔ Generates and returns a DTO object (described later) for the quote side.

## 5) TradableDTO class

The TradableDTO class contains a copy of an order or quote side’s data – used to transfer/return data about the order or quote side without giving out a reference to the actual object (good info hiding!).

- This class should be implemented as a *java record*, and should have the same data fields as the Order/QuoteSide classes:
  - String user, String product, Price price, int originalVolume, int remainingVolume, int cancelledVolume, int filledVolume, BookSide side, String tradableId
  - A secondary constructor should be created that accepts a Tradable object, and uses the tradables’ data values to call the primary constructor, passing it the individual data value.

## 6) ProductBook class

A ProductBook maintains the Buy and Sell sides of a stock’s “book”. A stock’s “book” holds the buy and sell tradables for a stock that are not yet tradable. The buy-side of the book contains all buy tradables that are not yet tradable in

descending order. The sell-side of the book contains all sell tradables that are not yet tradable in ascending order. Many of the functions owned by the ProductBook class are designed to simply pass along that same call to either the Buy or Sell side of the book. The “book” is often visually represented as follows:

MSFT (Microsoft) Book			
BUY		SELL	
1000	\$29.05	\$29.07	1000
120	\$29.01	\$29.08	120
210	\$28.98	\$29.10	210
80	\$28.94	\$29.12	80
...	...	...	...

- The ProductBook class should have a public constructor that accepts and sets the parameters as indicated below.
- The ProductBook will need the following data elements to properly represent a product’s book:
  - String product: The stock symbol that this book represents (1 to 5 characters) – must be 1 to 5 letters/numbers, they can also have a period “.” in the symbol (i.e., WMT, F, 3M, GOOGL, WOOF, AKO.A, etc.). Otherwise, no spaces, no special characters. This should be passed to the constructor and cannot be changed once set.
  - ProductBookSide buySide: ProductBookSide object that holds the Buy side of this book. Should be created new and set in the constructor. Cannot be changed once set.
  - ProductBookSide sellSide: ProductBookSide object that holds the Sell side of this book. Should be created new and set in the constructor. Cannot be changed once set.
- The ProductBook will need the following methods to properly represent a product’s book:
  - public TradableDTO add(Tradable t): First print a message indicating that you are adding a tradable, like this: System.out.println("\*\*\*ADD: " + t); Then throw (and propagate) an exception if the Tradable passed in is null. Call the “add” method of the ProductBookSide the tradable is for (BUY or SELL) and save the TradableDTO that is returned from the “add” call. Call “tryTrade” (seen below), then return the TradableDTO you were holding (from the “add” call)..
  - public TradableDTO[] add(Quote qte): Throw (and propagate) an exception if the Quote passed in is null. Then do the following:
    - First, call removeQuotesForUser (passing the user id as the parameter).
    - Call the “add” method of the BUY ProductBookSide, passing the BUY-side of the quote, and save the TradableDTO that is returned from the “add” call.
    - Call the “add” method of the SELL ProductBookSide, passing the SELL-side of the quote, and save the TradableDTO that is returned from the “add” call.
    - Call “tryTrade” (seen below)
    - Return both TradableDTOs in an array of TradableDTOs (BUY side DTO is index 0, SELL side DTO is index 1).
  - public TradableDTO cancel(BookSide side, String orderId): [Method is used for Orders] Call the “cancel” method of the ProductBookSide the order is for (BUY or SELL) and return the resulting TradableDTO.
  - public TradableDTO[] removeQuotesForUser(String userName): [Method is used for Quotes] This method should do the following:
    - Call the “removeQuotesForUser” method of the BUY ProductBookSide, passing the String userName, and save the TradableDTO that is returned.
    - Call the “removeQuotesForUser” method of the SELL ProductBookSide, passing String userName, and save the TradableDTO that is returned.
    - Then return both TradableDTOs in an array of TradableDTOs (BUY side DTO is index 0, SELL side DTO is index 1).
  - public void tryTrade(): Checks to see if the book sides are tradable, and if so, perform the trades. If not, it does nothing. See diagram in *Appendix A* for how this should work.

- `public String getTopOfBookString(Side side)`: This method returns a String summary of the top-of-book for the specified side. The String should be returned as follows:
  - Top of BUY book: Top of BUY book: \$122.50 x 75 -OR- Top of SELL book: Top of SELL book: \$122.90 x 100
- `public String toString()`: Override the `toString` method to generate a String containing a summary of the book content as follows (be sure to let the `ProductBookSides` generate their part of the String).

Product: WMT

Side: BUY

Price: \$9.95

CCC order: BUY AMZN at \$9.95, Orig Vol: 70, Rem Vol: 70, Fill Vol: 0, CXL Vol: 0, ID: CCCAMZN\$9.95186495726402400

Price: \$9.90

DDD quote: BUY AMZN at \$9.90, Orig Vol: 25, Rem Vol: 25, Fill Vol: 0, CXL Vol: 0, ID: DDDAMZN\$9.90186495726811600

Side: SELL

Price: \$10.10

DDD quote: SELL AMZN at \$10.10, Orig Vol: 120, Rem Vol: 10, Fill Vol: 110, CXL Vol: 0, ID: EEEAMZN\$10.10186495727149200

Price: \$10.20

FFF order: SELL AMZN at \$10.20, Orig Vol: 45, Rem Vol: 45, Fill Vol: 0, CXL Vol: 0, ID: FFFAMZN\$10.20186495727529400

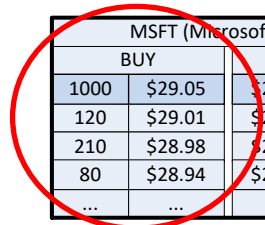
Price: \$10.25

GGG order: SELL AMZN at \$10.25, Orig Vol: 90, Rem Vol: 90, Fill Vol: 0, CXL Vol: 0, ID: GGGAMZN\$10.25186495727930400

## 7) ProductBookSide class

A `ProductBookSide` maintains the contents of one side (Buy or Sell) of a stock product “book”. A product book “side” holds the buy or sell orders for a stock that are not yet tradable. The “book” is often visually represented as shown below (with the product book buy side circled).

- The buy-side contains all buy-side tradables in descending order.
- The sell-side contains all sell-side tradables in ascending order.



BUY		SELL	
1000	\$29.05	\$29.07	1000
120	\$29.01	\$29.08	120
210	\$28.98	\$29.10	210
80	\$28.94	\$29.12	80
...	...	...	...

- The `ProductBookSide` class should have a public constructor that accepts and sets the parameters as indicated below.
- The `ProductBookSide` will need the following data elements to properly represent a product’s book side:
  - `BookSide side`: The “side” of the `ProductBookSide` – BUY or SELL (enum), do not allow null. This should be passed into the constructor. Cannot be changed once set.
  - `final TreeMap<Price, ArrayList<Tradable>> bookEntries`: Contains the tradables at each price that make up this book side. Set to a new `TreeMap` in the constructor. The diagram below shows how this is represented:

BUY Side		TreeMap< Price, ArrayList<Tradable>>
Key: Price		Value: ArrayList<Tradable>
\$12.98	→	[Tradable, Tradable, Tradable, Tradable]
\$12.95	→	[Tradable]
\$12.90	→	[Tradable, Tradable, Tradable]

- The ProductBookSide will need the following methods to properly represent a product's book side:
  - public TradableDTO add(Tradable o): Add the incoming Tradable to the bookEntries HashMap. If the price for the Tradable does not exist as a key in the HashMap, add the price to the HashMap as the key and a new ArrayList<Tradable> as the value. Then add the Tradable to that ArrayList, and return an TradableDTO made from the new tradable.
  - public TradableDTO cancel(String tradableId): Find the tradable using the tradable id passed in (search the ArrayLists at each price key in the bookEntries HashMap to find it). If you find it, first print a message indicating that you are cancelling a tradable, like this: System.out.println("\*\*CANCEL: " + t); [where "t" is the tradable you found]. Then remove the tradable from the ArrayList, add the tradable's remaining volume to the tradable's cancelled volume, then set the remaining volume to zero. If the ArrayList is empty after removing the tradable, remove the price and the empty ArrayList from the bookEntries HashMap. Finally, return a TradableDTO made from the cancelled tradable. If the tradable is not found, return null.
  - public TradableDTO removeQuotesForUser(String userName): [Used for Quotes] Find the quote using the userName passed in (search the ArrayLists at each price key in the bookEntries HashMap to find a quote matching the userName passed in). If you find it, call the "cancel" method (passing the Tradable's Id) and save the TradableDTO received back from the cancel method. If the ArrayList is empty after removing the quote, remove the price and the empty ArrayList from the bookEntries HashMap. Finally, return a TradableDTO made from the cancelled quote. If no quote is not found, return null.
  - public Price topOfBookPrice(): This method should return the first key in the bookEntries TreeMap. (i.e., bookEntries.firstKey). If the bookEntries TreeMap is empty, return null.
  - public int topOfBookVolume(): If the ProductBookSide is the BUY side, return total of all tradable volumes at the highest price in the bookEntries TreeMap. If the ProductBookSide is the SELL side, return total of all tradable volumes at the lowest price in the bookEntries TreeMap.
  - public void tradeOut(Price price, int vol): Trade out any tradables at or better than the Price passed in, up to the volume value passed in. See diagram in *Appendix B* for how this should work.
  - public String toString(): Override the toString method to generate a String containing a summary of the book side content as follows.

#### BUY side example:

```
Side: BUY
Price: $9.95
    CCC order: BUY AMZN at $9.95, Orig Vol: 70, Rem Vol: 70, Fill Vol: 0, CXL Vol: 0, ID: CCCAMZN$9.95189383477035100
Price: $9.90
    DDD quote: BUY AMZN at $9.90, Orig Vol: 25, Rem Vol: 25, Fill Vol: 0, CXL Vol: 0, ID: DDDAMZN$9.90189383477465600
```

#### SELL side example:

```
Side: SELL
Price: $10.10
    DDD quote: SELL AMZN at $10.10, Orig Vol: 120, Rem Vol: 10, Fill Vol: 110, CXL Vol: 0, ID: EEEAMZN$10.10189383478470500
Price: $10.20
    FFF order: SELL AMZN at $10.20, Orig Vol: 45, Rem Vol: 45, Fill Vol: 0, CXL Vol: 0, ID: FFFAMZN$10.20189383478919000
Price: $10.25
    GGG order: SELL AMZN at $10.25, Orig Vol: 90, Rem Vol: 90, Fill Vol: 0, CXL Vol: 0, ID: GGGAMZN$10.25189383479288700
```

**NOTE:** If the product book side is empty, generate the string as follows to show that it is empty:

```
Side: BUY           - OR -           Side: SELL
    <Empty>                <Empty>
```



## 8) Testing

Create a new empty class called “Main” and copy into that class the provided “main” method. The expected results are also provided in a text file. If your application generates the correct expected results using these tests, it is working. If not, you’ll want to look into what has gone wrong and fix the problem ASAP to complete the assignment.

## 9) Project Assistance

If you are stuck on some design or code related problem *that you have exhaustively researched and/or debugged yourself*, you can email me a ZIP file of your entire project so that I can examine the problem. All emailed assistance requests must include a detailed description of the problem, and the details of what steps you have already taken in trying to determine the source of the problem.

## 10) Submissions & Grading

When submitting, you should submit a ZIP file of your entire project so that I can compile and execute it on my end. To reduce the size of the zipped file, please remove the “out” folder from your project (if present) before zipping. (This folder is automatically regenerated each time you compile/run so It’s safe to delete)

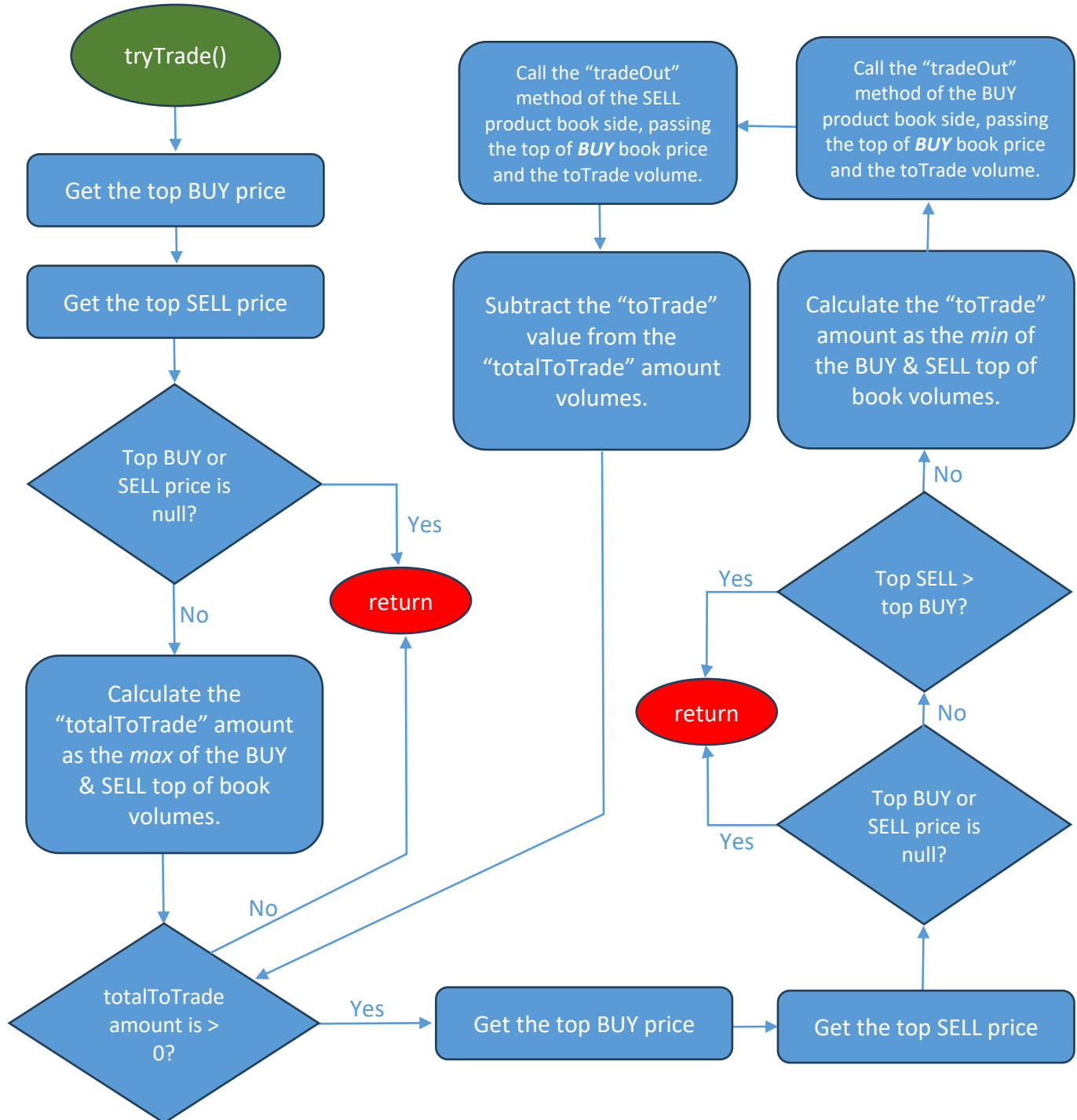
The following are the key points that will be examined in Project when graded:

- Good object-oriented design & implementation.
  - Proper use of java language conventions.
  - Proper object-oriented coding practices.
  - Correct, accurate application execution.
- 
- Submissions must reflect the concepts and practices we cover in class, and the requirements specified in this document.
  - Please review the Academic Integrity and Plagiarism section of the syllabus before, during, and after working on this assignment. All work must be your own.
  - Late submission up to 1 week late will be accepted, though late submissions will incur a 10% penalty (i.e., 10 points). *No late submissions will be accepted for grading after that 1-week time period has passed.*

*If you do not understand anything in this handout, please ask. Otherwise, the assumption is that you understand the content.*



Appendix A: ProductBook tryTrade()



**Appendix B: ProductBookSide tradeOut(Price price, int vol)**
