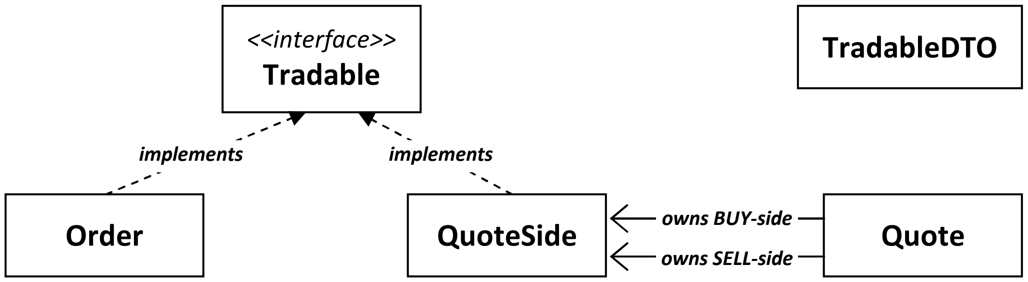
Course Programming Project

DePaul Stock Exchange (DSX)

Phase 2

Tradable, TradableDTO, Order, Quote & QuoteSide

Phase 2 of the Course Programming project involves the creation of several (5) classes that will represent the major tradable components of our trading system. These Tradable components will represent up the orders and quotes in the product "books", and they will trade with each other when appropriate. The following shows the Phase 2 classes and their relationships:



MSFT (Microsoft) Book			
BUY		SELL	
500	$29.97	$30.01	500
100	$29.90	$30.05	250

The entries in each Book side (BUY/SELL) are maintained as "Tradable" objects – they implement the Tradable interface. Anything that implements the Tradable interface can be added to a product's book. In our application Orders and the buy and sell side of a Quote are Tradable objects, so they can be added to the book.

The classes that make up Phase 2 are key classes to the overall application, but they do not include much functionality. They exist to hold information. As such, they are not particularly complex to write. BE SURE to follow the various object-oriented guidelines we have covered in class in this (and all) Phases. For example:

- Data members should be private
- Data members should be accessed via accessors (get methods)
- Data members should be set via modifier (set methods)
- ANY method that accepts a parameter should verify the integrity of the parameter, and throw an appropriate exception(s) if the parameter does not have an acceptable value.

1) Tradable (Interface)

The Tradable interface will be used as the common generic type for entities representing a BUY or SELL request that can be traded in our system. For example, if a user submits a BUY order for 100 shares of INTC at $23.27 – that is an entity representing a BUY request – so it will be considered a "Tradable" item. If a user submits a Quote to BUY 500 shares of JPM at $38.95 and SELL 500 shares of JPM at $39.00 – then 2 Tradable items are involved (remember "quotes" are made up of both a BUY and SELL side). The BUY side of the quote is a "Tradable" entity, and the SELL side of the quote is a "Tradable" entity. The Tradable interface defines behaviors that all entities that can be booked and traded in our system must be able to perform.

Note, that many of the entities in this project need to maintain an indicator of which "side" they represent: BUY or SELL. How you represent this is up to you – enumerated type, constants, etc. In this and other handouts, I will refer to this "side" attribute with the gener type name of "BookSide". Wherever you see "BookSide", you just nee to keep in mind that this refers to the type you have decided to use tc represent the BUY/SELL side.

Required Tradable Interface Behaviors *(and a general description of their purpose):*

- String getProduct() – Returns the product symbol (i.e., IBM, GOOG, AAPL, etc.) that the Tradable works with.
- Price getPrice() – Returns the price of the Tradable.
- int getOriginalVolume() – Returns the original volume (i.e., the original quantity) of the Tradable.
- int getRemainingVolume() – Returns the remaining volume (i.e., the remaining quantity) of the Tradable.
- int getCancelledVolume() – Returns the cancelled volume (i.e., the cancelled quantity) of the Tradable.
- void setCancelledVolume(int newCancelledVolume) – Sets the Tradable's cancelled quantity to the value passed in. This method should throw an exception if the value is invalid (i.e., if the value is negative, or if the value is inconsistent with the original volume, and the remaining volume).
- void setRemainingVolume(int newRemainingVolume) – Sets the Tradable's remaining quantity to the value passed in. This method should throw an exception if the value is invalid (i.e., if the value is negative, or if the value is inconsistent with the original volume, and the cancelled volume).
- String getUser()– Returns the User id associated with the Tradable.
- *BookSide* getSide() – Returns the "side" (BUY/SELL) of the Tradable.

- boolean isQuote()– Returns true if the Tradable is part of a Quote, returns false if not (i.e., false if it's part of an order)
- String getId()– Returns the Tradable "id" – the value each tradable is given once it is received by the system.

2) TradableDTO class

The TradableDTO class is based upon the "Data Transfer Object" design pattern (be sure to read the "Data Transfer Object" design pattern notes if you have not already done so). The TradableDTO will act as a "data holder" that holds selected data values from a "Tradable" object. Data elements held by the TradableDTO should be as follows (note that the content of the TradableDTO follows the Tradable interface functionality):

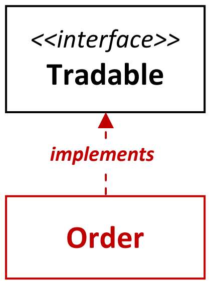
- *public* String product - The product (i.e., IBM, GOOG, AAPL, etc.) that the Tradable works with
- *public* Price price - The price of the Tradable
- *public* int originalVolume - The original volume (i.e., the original quantity) of the Tradable
- *public* int remainingVolume - The remaining volume (i.e., the remaining quantity) of the Tradable
- *public* int cancelledVolume - The cancelled volume (i.e., the cancelled quantity) of the Tradable
- *public* String user - The User id associated with the Tradable.
- *public* BookSide side - The "side" (BUY/SELL) of the Tradable.
- *public* boolean isQuote – Set to true if the Tradable is part of a Quote, false if not (i.e., false if it's part of an order)
- *public* String id - The Tradable "id" – the value each tradable is given once it is received by the system

NOTE: It is advisable to add a "toString" to the TradableDTO for debugging purposes.

3) Order class

An Order represents a request from a user to BUY or SELL a specific quantity of certain stock either at a specified price, or at the current market price. For example, "BUY 100 shares of EBAY at $48.17", or "SELL 250 shares of GE at the Market Price". The Order class should implement the Tradable interface, as it represents a BUY or SELL request that can be traded in our system. Order objects must possess data members to represent the following information:

- String data: The user name related to this order, the product (stock symbol) that the order specifies, and the "id" of the order (assigned by the system).
- Price: The Price object holding the price for this order.
- Numeric data: The values for the original, remaining, and cancelled volumes of the order.
- Other: The "side" of the order (BUY/SELL).



Required Order Class Behaviors:

- *public* constructor Order(String userName, String productSymbol, Price orderPrice, int originalVolume, BookSide side)– Creates an Order object. The 5 parameters passed in should be used to set the corresponding Order data members. The constructor should also set the Order's *remaining volume* to the *original volume* value, since all Orders start with the remaining volume equal to the original volume. NOTE, the Order's "id" should be set in the constructor to the following combined String:

  *id = the user name + the product symbol + the order price + the current time in nanoseconds***.

  Example:
  - the user name = "REX"
  - the product symbol = "AMZN"
  - the order price = $257.09
  - the current time in nanoseconds = *1684142189815011*

  id = "REX" + "AMZN" + "$257.09" + 1684142189815011

  id = "REXAMZN$257.091684142189815011"

  ***The current time in nanoseconds can be obtained by calling "System.nanoTime()" in your code.*

- The remainder of the Order class behaviors consists of the Tradable interface-required behaviors (listed earlier). Those methods should "get" or "set" the data member they are designed to work with *(i.e., "String getProduct()" should return the product symbol "int getRemainingVolume()" should return the remaining volume value, "setRemainingVolume(int newRemainingVolume)" should set the remaining volume to the new value passed in, etc).*

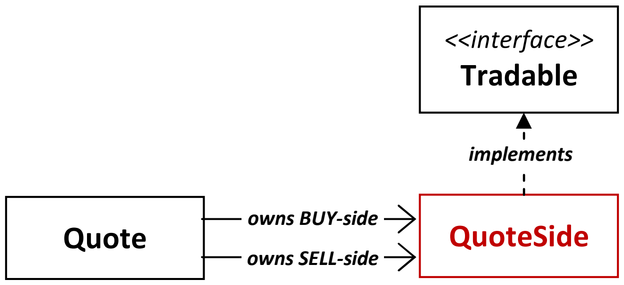
- public String toString() – Generates a String representing the key values in this order. Recommended String value generated by your toString:

  USER1** order: **BUY 250 GE** at **$21.59** (Original Vol: **250**, CXL'd Vol: **0**), ID: **USER1GE$21.591684416944495943

4) QuoteSide class

QuoteSide represents the price and volume for one side (BUY or SELL) of a Quote. One side of a Quote (the QuoteSide object) represents the price and volume of certain stock that the user is willing to BUY or SELL shares. For example: "BUY 200 shares of FB at $19.42". The QuoteSide class should implement the Tradable interface, as it represents a BUY or SELL request that can be traded in our system. QuoteSides must possess data members to hold the following information:

- String data: The user name related to this order, the product (stock symbol) that the order specifies, and the "id" of the order (assigned by the system).
- Price: The Price object holding the price for this order.
- Numeric data: The values for the original, remaining, and cancelled volumes of the order.
- Other: The "side" of the order (BUY/SELL).



Required QuoteSide Class Behaviors:

- *public* constructor QuoteSide (String userName, String productSymbol, Price sidePrice, int originalVolume, BookSide side)– Creates a QuoteSide object. The 5 parameters passed in should be used to set the corresponding QuoteSide data members. The constructor should also set the QuoteSide's *remaining volume* to the *original volume* value, since all QuoteSides start with the remaining volume equal to the original volume. NOTE, the QuoteSide's "id" should be set in the constructor to the following combined String:

  *id = the user name + the product symbol + the current time in nanoseconds***.

  Example:
  - the user name = "ARI"
  - the product symbol = "FB"
  - the current time in nanoseconds = *1684142189816542*

  id = "ARI" + "FB" + 1684142189816542

  id = "ARIFB1684142189816542"

  ***The current time in nanoseconds can be obtained by calling "System.nanoTime()" in your code.*

- *public* constructor QuoteSide (QuoteSide qs)– Creates a QuoteSide object using the QuoteSide passed in "qs" to set the values of the new QuoteSide object (this is a "copy" contructor). *All* values in the QuoteSide you are creating should be set to the values held in the "qs" QuoteSide. (i.e., set the user name of the new QuoteSide to the user name value in the QuoteSide "qs" passed in, set the price of the new QuoteSide to the price in the QuoteSide "qs" passed in, etc).

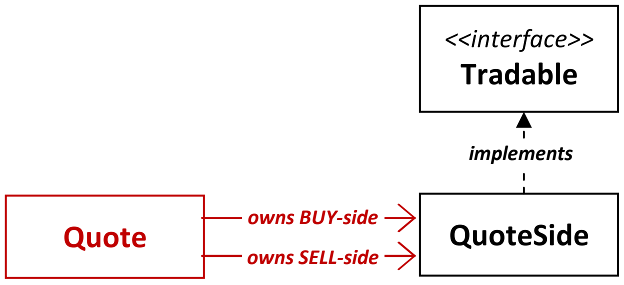
- The remainder of the QuoteSide class behaviors consists of the Tradable interface-required behaviors (listed earlier). Those methods should "get" or "set" the data member they are designed to work with *(i.e., "String getProduct()" should return the product symbol "int getRemainingVolume()" should return the remaining volume value, "setRemainingVolume(int newRemainingVolume)" should set the remaining volume to the new value passed in, etc).*

- public String toString() – Generates a String representing the key values in this QuoteSide. Recommended String value generated by your toString:

  ***$21.56** x **100** (Original Vol: **100**, CXL'd Vol: **0**) [**USER2GE1756426242224751**]*

5) Quote class

A Quote represents the prices and volumes of certain stock that the user is willing to BUY or SELL shares. For example: "BUY 200 shares of FB at $19.42 and SELL 200 shares of FB at $19.48". The Quote class consists of a String user name, a String product symbol, and 2 QuoteSide objects (one for the BUY side and one for the SELL side). A Quote itself is not a "Tradable" item as it represents both the BUY and SELL sides, each of which is separately traded. Quote should not implement the Tradable interface. The individual QuoteSide objects are traded so the QuoteSide class should implement the Tradable interface:



Required Quote Class Behaviors:

- *public* constructor Quote(String userName, String productSymbol, Price buyPrice, int buyVolume, Price sellPrice, int sellVolume)– Creates an Quote object. The userName and productSymbol parameters should be used to set the corresponding Quote data members. The buyPrice and buyVolume should be used (along with the username, productSymbol and a BUY side indicator) to create the buy-side QuoteSide object. The sellPrice and sellVolume should be used (along with the username, productSymbol and a SELL side indicator) to create the sell-side QuoteSide object.

- *public String getUserName()* – Returns the Quote's user name.

- *public String getProduct ()* – Returns the Quote's product symbol.
- *public QuoteSide getQuoteSide (BookSide sideIn)* – Returns a ***copy*** of the BUY or SELL QuoteSide object, depending upon which side is specified in the "sideIn" parameter.
- public String toString() – Generates a String representing the key values in this quote. Recommended String value generated by your toString:

  ***USER2** quote: **GE $21.56** x **100** (Original Vol: **100**, CXL'd Vol: **0**) [**USER2GE1756426242224751**] - **$21.62** x **100** (Original Vol: **100**, CXL'd Vol: **0**) [**USER2GE1756426242242844**]*

  (The above String should appear on a single line. I had to break it into 2 lines to fit on this page)

Testing Phase 2

A test "driver" class with a "main" method will be provided that will exercise the functionality of your Tradable, TradableDTO, Order, Quote & QuoteSide classes. This will not exhaustively test your classes but successful execution is a good indicator that your classes are performing as expected. The output of the test driver is shown below:

Phases & Schedule

The Course Programming Project will be implemented in phases, each with a specific duration and due date as is listed below. Detailed documents on each phase will be provided at the beginning of the phase.

Phase 1 (1 Week) 9/17 – 9/24:

- Price & Price Factory

Phase 2 (1 Week) 9/24 – 10/1:

- **Tradable & Tradable DTO**
- **Order**
- **Quote & Quote Side**

Phase 3 (2 Weeks) 10/1 – 10/15: [Midterm 10/8]

- Current Market Publisher
- Last Sale Publisher
- Ticker Publisher
- Message Publisher
- Fill Message
- Cancel Message
- Market Message
- User (interface)

Phase 4 (2 Weeks) 10/15 – 10/29:

- Product Service
- Book & Book Side
- Trade Processor

Phase 5 (1 Week) 10/29 – 11/5:

- User Implementation
- User Command Service

Phase 6 (1 Week) 11/5 – 11/12:

- User Interface GUI
- Simulated Traders

[Final Exam 11/19]