**Final Project: A Trading System**

**MTH 9815: Software Engineering for Finance**

Note: Please use the C++ coding standards as specified in the following guide:

<https://google-styleguide.googlecode.com/svn/trunk/cppguide.html>

**DUE DATE: FRIDAY, DECEMBER 16, 2016**

You can reach out to me on the forum should you have any questions. You should share your repository (bitbucket or GitHub) with my ID breman\_t.

**INSTRUCTIONS**

Develop a bond trading system for US Treasuries with six securities: 2Y, 3Y, 5Y, 7Y, 10Y, and 30Y. Look up the CUSIPS, coupons, and maturity dates for each security. Ticker is T.

We have a new definition of a Service in soa.hpp, with the concept of a ServiceListener and Connector also defined. A ServiceListener is a listener to events on the service where data is added to the service, updated on the service, or removed from the service. A Connector is a class that flows data into the Service from some connectivity source (e.g. a socket, file, etc) via the Service.OnMessage() method. The Publish() method on the Connector publishes data to the connectivity source and can be invoked from a Service. Some Connectors are publish-only that do not invoke Service.OnMessage(). Some Connectors are subscribe-only where Publish() does nothing. Other Connectors can do both publish and subscribe.

Use the base classes in tradingsystem.zip attached to this thread. You should define the following bond specific classes:

BondTradeBookingService

BondPositionService

BondRiskService

BondPricingService

BondMarketDataService

BondExecutionService

BondStreamingService

BondAlgoExecutionService (no base class – you should create this from scratch)

BondAlgoStreamingService (no base class – you should create this from scratch)

BondInquiryService

BondHistoricalDataService

The following services below should read data from a file – you should create sample data as outlined below. You should read data from the file via a Connector subclass. The Connector should flow data from the file into the Service via the Service.OnMessage() method. Note that you should use fractional notation for US Treasuries prices when reading from the file and writing back to a file in the BondHistoricalDataService (with the smallest tick being 1/256th). Example of fractional is 100-xyz, with xy being from 0 to 31 and z being from 0 to 7 (you can replace z=4 with +). The xy number gives the decimal out of 32 and the z number gives the remainder out of 256. So 100-001 is 100.00390625 in decimal and 100-25+ is 100.796875 in decimal.

Some of the input text files you may create via some script (Python or C++ perhaps). Provide those files for extra credit!

Finally, make sure to provide some sort of README of how to run your program. Your code MUST be compatible with a g++ 5.x compiler.

**BondTradeBookingService**

This should read data from trades.txt. Create 10 trades for each security (so a total of 60 trades across all 6 securities) in the file with the relevant trade attributes. Positions should be across books TRSY1, TRSY2, and TRSY3. Use a subscribe-only Connector to flow these trades into your Service. The BondTradeBookingService should be linked to a BondPositionService via a ServiceListener and send all trades there via the AddTrade() method (note that the BondTradeBookingService should not have an explicit reference to the BondPositionService though or vice versa – link them through a ServiceListener).

**BondPricingService**

This should read data from prices.txt. Create 1,000,000 prices for each security (so a total of 6,000,000 prices across all 6 securites). The file should create prices which oscillate between 99 and 101 (bearing in mind that US Treasuries trade in 1/256th increments). The bid/offer spread should oscillate between 1/128 and 1/64. Use a subscribe-only Connector to flow these prices into your Service. The BondPositionService should should be linked to a BondRiskService via a ServiceListenher and send all positions to the BondRiskService via the AddPosition() method (note that the BondPositionService should not have an explicit reference to the BondRiskService though or versa – link them through a ServiceListener).

**BondPositionService**

The BondPositionService does not need a Connector since data should flow via ServiceListener from the BondTradeBookingService.

**BondRiskService**

The BondRiskService does not need a Connector since data should flow via ServiceListener from the BondPositionService.

**BondMarketDataService**

This should read data from marketdata.txt. Create 1,000,000 order book updates for each security (so a total of 6,000,000 prices) each with 5 orders deep on both bid and offer stacks. The top level should have a size of 10 million, second level 20 million, 30 million for the third, 40 million for the fourth, and 50 million for the fifth. The file should create mid prices which oscillate between 99 and 101 (bearing in mind that US Treasuries trade in 1/256th increments) with a bid/offer spread of 1/128th on the top of book (and widening in the smallest increment from there for subsequent levels). Use a subscribe-only Connector to flow this market data into your Service.

**BondAlgoExecutionService**

This should be keyed on product identifier with value an AlgoExecution object. The AlgoExecution should be a class with a reference to an ExecutionOrder object. BondAlgoExecutionService

should register a ServiceListener on the BondMarketDataService and aggress the top of the book, alternating between bid and offer (taking the opposite side of the order book to cross the spread) – it should send this order to the BondExecutionService via a ServiceListener and the ExecuteOrder() method.

**BondAlgoStreamingService**

This should be keyed on product identifier with value an AlgoStream object. The AlgoStream should be a class with a reference to a PriceStream object. BondAlgoStreamingService

should register a ServiceListener on the BondPricingService and send the bid/offer prices to the BondStreamingService via a ServiceListener and the PublishPrice() method.

**BondExecutionService**

The BondExecutionServicedoes not need a Connector since data should flow via ServiceListener from the BondAlgoExecutionService. The BondExecutionService should write orders to a text file via a publish-only Connector.

**BondStreamingService**

The BondStreamingService does not need a Connector since data should flow via ServiceListener from the BondAlgoStreamingService. The BondStreamingService should write price streams to a text file via a publish-only Connector.

**BondInquiryService**

You should read inquiries from a file called inquiries.txt with attributes for each inquiry (with state of RECEIVED). You should create 10 inquiries for each security (so 60 in total across all 6 securities). Use a subscribe-only Connector to flow these inquiries into your Service. You should register a ServiceListener on the BondInquiryService which sends back a quote when the inquiry is in the RECEIVED state. The BondInquiryService should send a quote of 100 back to a Connector via the Publish() method. The Connector should transition the inquiry to the QUOTED state and send it back to the BondInquiryService via the OnMessage method with the supplied price. It should then immediately send an update of the Inquiry object with a DONE state. Then it moves on to the next inquiry from the file and we repeat the process.

**BondHistoricalDataService**

This service should register a ServiceListener on the following: BondPositionService, BondRiskService, BondExecutionService, BondStreamingService, and BondInquiryService. It should persist objects it receives from these services back into files risk.txt, executions.txt, streaming.txt, and allinquiries.txt via special Connectors for each type with a Publish() method on each Connector. There should be a BondHistoricalDataService corresponding to each data type. When persisting positions, we should persist each position for a given book as well as the aggregate position. When persisting risk, we should persist risk for each security as well as for the following bucket sectors: FrontEnd (2Y, 3Y), Belly (5Y, 7Y, 10Y), and LongEnd (30Y). Use realistic PV01 values for each security.