**Considerations and assumptions**

1. The main interface to the program is through the SuperSimpleStockMarket class. This holds 2 data structures to hold the **processed** trades and **unprocessed** trades
2. The processTrade() method of this class will cause the trade to be added to one of the two data structures above depending on whether the stock symbol exists in the in memory “database”.
3. The **Dividend Yield** and **P/E ratio** is calculated on the main processing flow of the trade. The methods are private in the Trade class. On processing the trade, the Dividend Yield and P/E ratio is calculated and stored in the Trade object for later reporting **along with the timestamp, quantity, buy/sell, and price.** TradeTest.java focuses on asserting these data fields. For P/E ratio, the denominator can be zero, in this case, the P/E ratio of zero is assumed. For Dividend yield, it is highly unlikely that the denominator (price) will be zero, but this has been handled and if the price is zero, the dividend yield of zero is returned.
4. The **Volume Weighted Stock Price** is outside the individual trade processing flow. To allow for testing of the 15-minute window, the time window is made configurable, and, in the tests, seconds is used with sleep to show the window working (due to the sleep, this test will take about a minute to run). **If the ∑quantity in the window is zero**, then it is assumed that zero is returned from this method. These are tested in SuperSimpleStockMarketTest.java.
5. The **Geometric Mean** is also outside the individual trade processing flow and uses the nth root method highlighted below. This is also tested in SuperSimpleStockMarketTest.java. If there are no trades to process, then Zero is returned.
6. If this program were to be used with concurrency, then the two data structures need to be thread safe and access to them synchronised. This has been added for illustrative purposes, but the program was not tested with concurrency.
7. Except for quantity (which is assumed a whole number), all other numerical data types are assumed to have decimal places and the ‘BigDecimal’ data type is used to store them (since the ‘double’ data type loses precision). The ‘Fixed dividend’ which is represented as a percentage is also stored as a BigDecimal for convenience (e.g. 2% is stored as 0.02)
8. The use of BigDecimal introduced an additional complexity with calculating the nth root. Please see comment at the top of NthRoot.java for the rationale of adding this class. This code was copied, and unit tests were added to verify it produces acceptable results (NthRootTest.java). It was tested with non-terminating roots -e.g., √2- and the precision of the results compared with a scientific calculator to an arbitrary precision controlled by the SCALE constant.