**FX Profit Report Engine: Design Document**

The application is written in JAVA8 to make use of features such as lambda expressions and LocalDate/LocalTime features for time comparison.

Data dependencies

* market rates and daily transactions – these are dynamic files that can be updated daily (or even hourly), and thus, these are taken in as external CSV file defined by a properties file. There is a model object and a reader assigned to each dataset so that the data can be easily communicated across different layers such as CSVReader and ReportGenerator.
* mark\_up rates – these are constant data since banks do not change mark up rate on transaction frequently. Given that this is a demo project with no data storage, I used *unmodifiableNavigableMaps* to store two different mark-up for two different client type. The map is unmodifiable so that the mark up values are always constant, and it is navigable so that I can easily search the range using lowerEntry() method provided by NavigableMaps.

Utility Classes with static methods

* CSVUtil: A utility class that can perform generic read/write operations on given csv files. It is used across various readers and report generator to perform read/write actions on csv files.
* PropertiesUtil: A utility class to read and load the input properties file

Enums

* Currency and Client Type are set as ENUM with known values. This is (a) to catch and skip any data error on the input files and (b) to easily skip the header row if there is one.

Main Class

* The backbone class of this project is ReportGenerator, which performs the end-to-end process of reporting the profit.csv from the input.
* The process design is straightforward.
  1. First the application will get the market rates for the day and because market rates are being used several time, it is set as a class level variable – a list of market rate objects, and loaded at init.
  2. Afterwards, the transaction file is read into a list of transaction objects and loop through one by one to get the related market rate, mark up rate and calculate the profit according to the instructions.
  3. Market rate for a given transaction is retrieved by first using java 8 stream-filter on lists to get the relevant market rates. And from this list, LocalTime compareTo() function is used to get the nearest available time. If a transaction doesn’t have the related market rate, the rate is take as 0 to not fail the process and the log will print out ERROR messages stating the particular row that has issue.
  4. Mark Up rate for a given transaction is retrieved by comparing the amount to the lowerEntry of the constant MarkUp maps.
  5. Once, the market rate and mark up rate are determined, the calculations occur and output row in a csv format is created.
  6. Once there is one output for each transaction, the output is then written to the report file with optional headers defined in properties file.