- Run results are generated in CSV format, and are stored in the "result" folder created for this purpose.

- The Client.java class implements the calculation of assets per client, according to the logic described in the specifications.

- The ProductDetail.java class implements in particular a HashMap to store the products (each associated with its portfolio and its total value after forex exchange).

- The CsvIO.java and Utility.java classes are utility classes for processing, in particular Read/Write operations on CSV files.

Ps: Regarding the Forex.csv file, I took the liberty of changing the value of the Euro -> Yen exchange rate from 0.5 to 50, because I assumed that it was a typing error, and that it is more realistic that a euro costs more than a yen, rather than the other way around.

- I also corrected the code to parse the files only once.

- Concerning the design and the implementation, I redesigned the architecture of the code to have a new design that respects the SOLID principles, in particular: the first two:

- The Utility class has thus been removed, and its methods are now part of the "Single Responsibilities" of the abstract class Item, parent of the Client and Portfolio classes. This gives these two classes the properties Open for Extension and Closed for Modification.

- An alternative to this design would have been to go through two interfaces, extended one by the other, and implemented respectively by the Client and Portfolio classes, to implement the Liskov Substitution Principle.

- The CsvIO class has also been removed, to factorize the code, and make the read/write operations abstract from the calling objects.

- Concerning the tests, I did several rounds of TDD (each round corresponds to a commit on the git repo, with a message telling each commit the requirements and the methods tested, and a git add of the failing tests that the TDD round passes). The overall coverage rate is over 90%. The link of the git repo is the following https://github.com/mndben/repositoryCSV

- Last but not least, I manually checked that my code produces the correct values (Below are the checks per client and per portfolio).

```
PTF1 = P1*(10+40+30+10) + P2(20+80) + P3(100+90+80+30+10)
= (EUR10 + USD20 + USD30)*90 + TND5*100 + (JPY180 + CHF20)*310
= EUR35*90 + EUR0.5*100 + EUR83.6*310
= EUR3150 + EUR50 + EUR25916 = 29116

PTF2 = X1*(10+20+40) + X2(20+30+50+60)
= (EUR10 + EUR20)*70 + (GPB20 + GPB40 + GPB60)*160
= EUR30*70 + EUR960*160
= EUR2100 + EUR153600
= 155700
```

```
C1 = 10*P1 + 80*P2 + 20*X1
= 10(EUR10 + USD20 + USD30) + 80*TND5 + 20(EUR10 + EUR20)
= 10*EUR35 + 80*EUR0.5 + 20*EUR30
= 350 + 40 + 600
= 990

C2 = 40*P1 + 100*P3 + 20*X2
= 40(EUR10 + USD20 + USD30) + 100*(JPY50 + JPY60 + JPY20 + JPY10 + JPY40 +
CHF20) +
20(GPB20 + GPB40 + GPB60)
= 40*EUR35 + 100*(JPY180 + CHF20) + 20*GBP120
= 40*EUR35 + 100*(EUR3.6 + EUR80) + 20*EUR960
= 1400 + 8360 + 19200
= 28960

C3 = 30*P1 + 80*P3
= 30(EUR10 + USD20 + USD30) + 80*(JPY50 + JPY60 + JPY20 + JPY10 + JPY40 + CHF20)
= 30*EUR35 + 80*(JPY180 + CHF20)
= 30*EUR35 + 80*(EUR3.6 + EUR80)
= 1050 + 6688
= 7738

C4 = 10*P1
= 10(EUR10 + USD20 + USD30)
= 10*EUR35
= 350

C5 = 20*P2 + 90*P3 + 50*X2
= 20*TND5 + 90*(JPY50 + JPY60 + JPY20 + JPY10 + JPY40 + CHF20) +
50(GPB20 + GPB40 + GPB60)
= 20*EUR0.5 + 90*(JPY180 + CHF20) + 50*GBP120
= 20*EUR0.5 + 90*(EUR3.6 + EUR80) + 50*EUR960
= 10 + 7524 + 48000
= 55534


C6 = 30*P3 + 10*X1
= 30*(JPY50 + JPY60 + JPY20 + JPY10 + JPY40 + CHF20) +
10(EUR10 + EUR20)
= 30*(JPY180 + CHF20) + 10*EUR30
= 30*(EUR3.6 + EUR80) + 10*EUR30
= 2508 + 300
= 2808

C7 = 10*P3 + 40*X1 + 30*X2
= 10*(JPY50 + JPY60 + JPY20 + JPY10 + JPY40 + CHF20) +
+ 40(EUR10 + EUR20) + 30(GPB20 + GPB40 + GPB60)
= 10*(JPY180 + CHF20) + 40*EUR30 + 30*GBP120
= 10*(EUR3.6 + EUR80) + 40*EUR30 + 30*EUR960
= 836 + 1200 + 28800
= 30836

C8 = 60*X2
= 60(GPB20 + GPB40 + GPB60)
= 60*GBP120
= 60*EUR960
= 57600
```