**Observer Pattern**

Hand-in #1

I4SWD – Gruppe 18

Design Dokument

|  |  |
| --- | --- |
| 201607110 | Kasper Juul Hermansen |
| 201400298 | Karsten Winther Johansen |
| 201310517 | Rasmus Lund |

# Overvejelser

Vi er til opgaven blevet bedt om at benytte os af Observer Pattern, som benytter sig af Subjects og Observers.

For at kunne benytte dette design pattern, er vi startet ud med at definere hvilke elementer i opgaven som er Subjects og Observers.

Til vores design er vi nået frem til at Stocks er vores Subject, da Stocks fungere uafhængige af Portfoliet, og Portfoliet er en Observer, da dette er afhængig af tilstanden i Stocks, eller som bruger tilstanden af Subjectet.

StockMarket opføre sig som en observer, men da det er en statisk klasse, kan den derfor ikke blive implementeret efter IObserver interfacet. StockMarket kan ses som et ”design smell” da denne er en statisk klasse, og derfor er svær at teste.

Vi har valgt at bruge vores egne klasser, pga. udfordringen, alt det vi selv har implementeret kunne være gjort med standard klasser som IObservable<T>() og IObserver<>(). Til Portfolio klassen bruger vi pull metode, hvor vi notificerer Portfolio at der er en ændring i klassen Stock, det bliver derefter printet ud på konsollen igennem klassen PortfolioDisplay. Til StockMarket er der brugt en push metode, hvor Stock sender sig selv og StockMarket bruger det object til at printe Stock notifikationer på konsollen.

StockMarket og Portfolio er designet således at notifikationer omkring ændringer i Stock kan slås fra, dette er for at undgå at brugerens input bliver forstyrret af rod på konsollen.

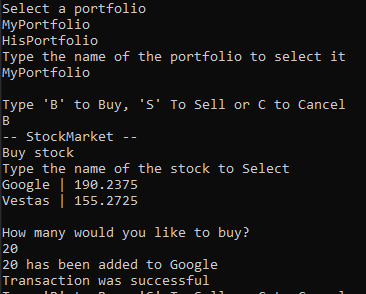
Portfolio bliver printet ud som en tabel, med værdierne: navn på Stock, værdi, antal og total værdi. Stock bliver printet ud som: navn på Stock, nye værdi.

Stock lever et liv af sit eget, og det kræver derfor en til flere tråde at håndtere dette, hvilket også er oprettet. Stock bliver opdateret hvert sekund og dets værdi bliver ændret med max +/- 5% igennem en metode Instability(), som ændre den interne værdi af Stock.

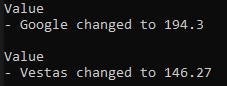
# Tests

Alle Unit tests er vedlagt source koden og kan testes ved gennemløb af solution.

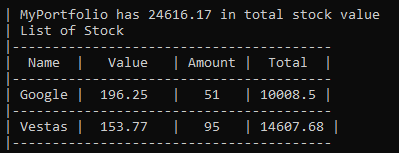
Transaktion i konsollen



Stock notifikationer



Portfolio notifikationer



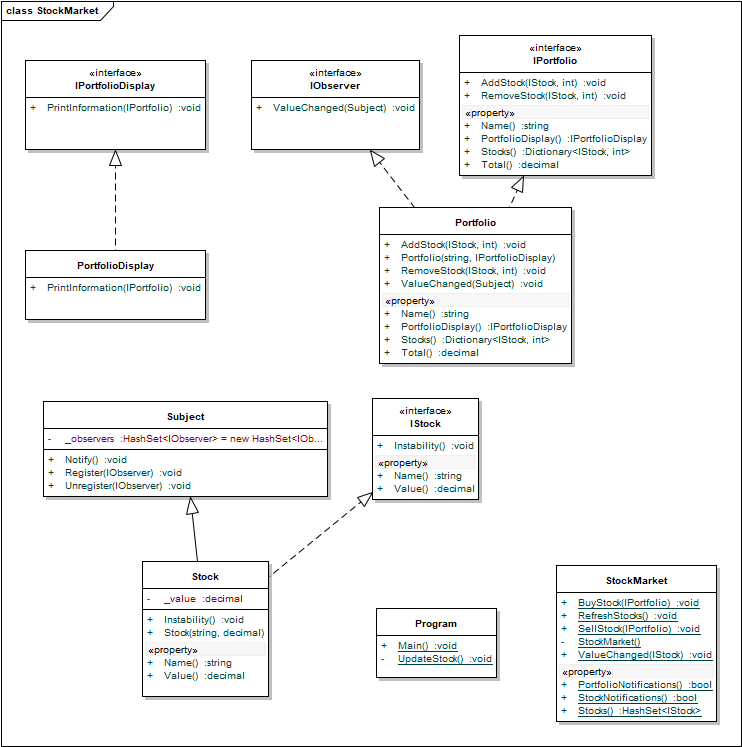
# Diagrammer

Nedenfor ses vores sekvensdiagram for StockMarket. Vi starter med at registrere vores Portfolio til Stock, således at det er muligt at notificere når der er ændringer i Stock. Dette sker vha. Notify()



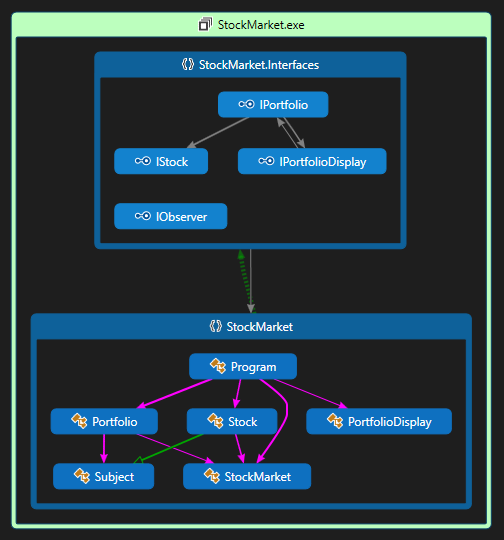
Figur 1 - Sekvensdiagram for StockMarket

Dette sekvensdiagram [1] beskriver hoved princippet bag Observer-Subject Pattern, og er brugt i solution til at løse opgaven med selvstændige Stock og afhængige Portfolio som bruger eller viser disse værdier af Stock.



Figur 2 - Klasse diagram for StockMarket

Til at teste og læse opgaven er der blevet brugt et klassediagram [2] for at planlægge den generelle struktur. Interfaces og Abstraktion er blevet brugt til at efter følge SOLID og DRY. At der ikke bliver duplikeret kode og at kontrakter bliver fulgt, så koden er fleksibel og testbar.



Figur 3 - Codemap for StockMarket

Som det kan ses på codemap, så er de eneste afhængigheder der er værd at overveje Program og Stock, hvilket som beskrevet tidligere godt kunne være et codesmell, eftersom der er mange afhængigheder og som en statisk metode er, ikke at være i stand til at følge en sat kontrakt.