SWT Hand-in 2

Mathias Friis - au547006  
Philip Nygaard Schmidt – au548068  
Daniel Tøttrup – au544366  
Lasse Lilholdt – au545690

Contents

[Refleksion over design 2](#_Toc526346048)

[Interfaces 2](#_Toc526346049)

[Fakes 2](#_Toc526346050)

[Refleksion over tests 2](#_Toc526346051)

[Ingen test af FileLogger og ConsoleRenderer 2](#_Toc526346052)

[Refkelsion over arbejdsdeling 2](#_Toc526346053)

[Refleksion over CI 3](#_Toc526346054)

[Verfikation af tests 3](#_Toc526346055)

[Code Coverage 3](#_Toc526346056)

[Appendix 4](#_Toc526346057)

[Appendix 1 – Klassediagram udkast 4](#_Toc526346058)

[Appendix 2 – Sekvensdiagram udkast for handleNewTrackData 5](#_Toc526346059)

# Refleksion over design

For at skabe en enighed om designet udarbejdes først en skitse til et klasse-diagram som beskrev modelklasser, interfaces samt “hoved-klassen”, ATMclass. Udover diagrammet udarbejdedes et dokument som kort beskrev alle klasser, deres formål og deres hovedfunktionalitet. Diagrammet er vedlagt som Appendix A, og HVAD GØR VI MED DOKUMENTET?

Derudover udarbejdedes et sekvens-diagram som udgjorde en skitse for flowet i ATMclass’es handleNewTrackData. Diagrammet er vedlagt som Appendix B

## Interfaces

I et forsøg på at gøre vores kode mere testbar har vi brugt interfaces til at lave løs kobling i vores system. På denne måde kan de enkelte dele af systemet testes separate. Eksempelvis kan funktionaliteten i Airspace testes udenom resten af systemet. Dette gør det nemmere at lokalisere hvor eventuelle fejl måtte opstå I systemet, hvorefter disse kan udbedres.

## Fakes

Det at vi burger interfaces gør at vi kan fake vores implementeringer af interfaces, så vi kan verificere at de bliver kaldt I de korrekte sammenhænge. Dette gøres simpelt ved at lave en counter, som tæller op for hvert system-kald, og eventuelt attributter til at holde på states af de passede parameter. Efter systemet har kørt i test-sammenhængen kan man så asserte på fake-objektets counter eller attributer.

# Refleksion over tests

## Ingen test af FileLogger og ConsoleRenderer

Vi har undladet at teste FileLogger og ConsoleRenderer.

ConsoleRenderer testes ikke da den blot udskriver data I consolen, og da vi tester i Fake-udgaven at det givne parameter bliver modtaget ordentligt må den antages at virke. Udskrivningen til consolen verficeres istedet ved en visuel test.

FileLogger testes heller ikke, da vi ikke er helt sikre på om dette skal gøres, og I så fald hvordan det skal gøres. Her foretages istedet en manuel test.

# Refkelsion over arbejdsdeling

Efter udarbejdelse af designet mødtes vi og uddelegerede opgaver. 2 mand gik i gang med at implementere interfaces’ne samt deres implementeringer, og teste disse for sig. De sidste 2 gik i gang med at implementere funktionaliteten i ”hoved-klassen”, ATMclass.

***OBS:*** *Implementeringen af ATMclass krævede sparring, da dens funktionalitet er mere kompleks end for de øvrige klasser. Dette resulterede i at to mand sad ved en enkelt computer (Lasse og Daniel), hvilket er hvorfor Daniel ikke har committet så meget. Vi hæftede os desværre først ved at hele gruppen ideelt skal committe omtrent lige meget ved implementeringens ende. Hele gruppen kan dog stå inde for at alle har bidraget i en ens grad.*

Da holdet, som implementerede og testede de ”små” klasser var færdige, rykkede dette hold videre til opsætning af CI-job, så vi kunne holde øje med code coverage, og udarbejdelse af tests til de øvrige dele af systemet i takt med at disse blev implementeret.

Efter endt implementering og test lavedes et revideret klassediagram ud fra implementeringen samt sekvens-diagrammer til at beskrive nogle af hovedscenarierne. Dette er gjort for at skabe overblik over systemet med henblik på videreudvikling.

# Refleksion over CI

Vi har brugt git til at kunne versionsstyre, samt til at kunne arbejde flere mand på det samme projekt samtidig. På denne måde har vi alle kunnet arbejde samtidigt, hvilket selvfølgelig har øget vores effektivitet.

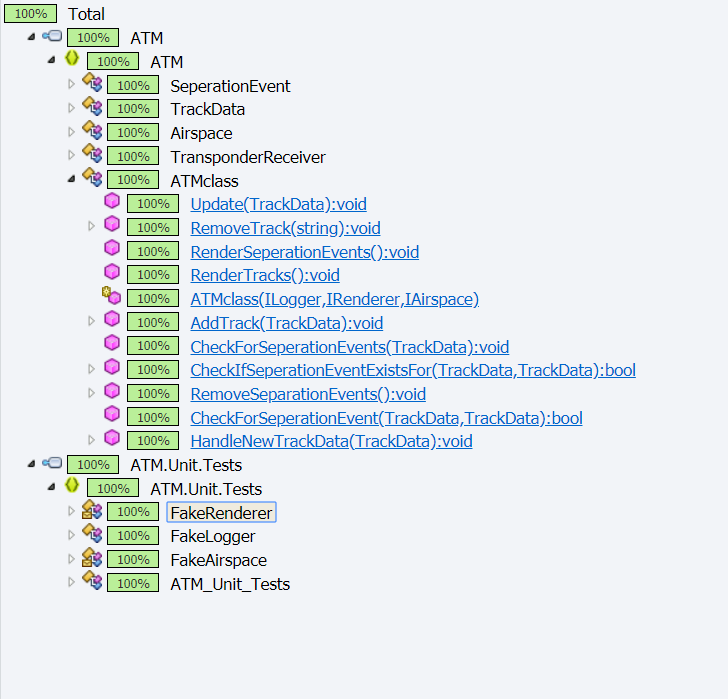
## Verfikation af tests

Vi har oprettet et Jenkins-job til at køre alle vores tests igennem, og rapportere hvor mange der er gået godt. På denne måde ses det tydeligt hvis noget pushet kode breaker noget. Dette har resulteret i at fejl hurtigt er blevet opdaget og rettet.

## Code Coverage

Ud over at verificere at den tilføjede kode fungerer har vi desuden brugt Jenkins til at oprette en Code Coverage report hver gang der er blevet uploadet ny kode, som går igennem alle tests. Denne Coverage Report giver et overblik over hvor meget af den skrevne kode der bliver berørt af tests. Vi har undladet at teste getters og setters, da disse blot er standard-implementeringerne leveret af .NET.

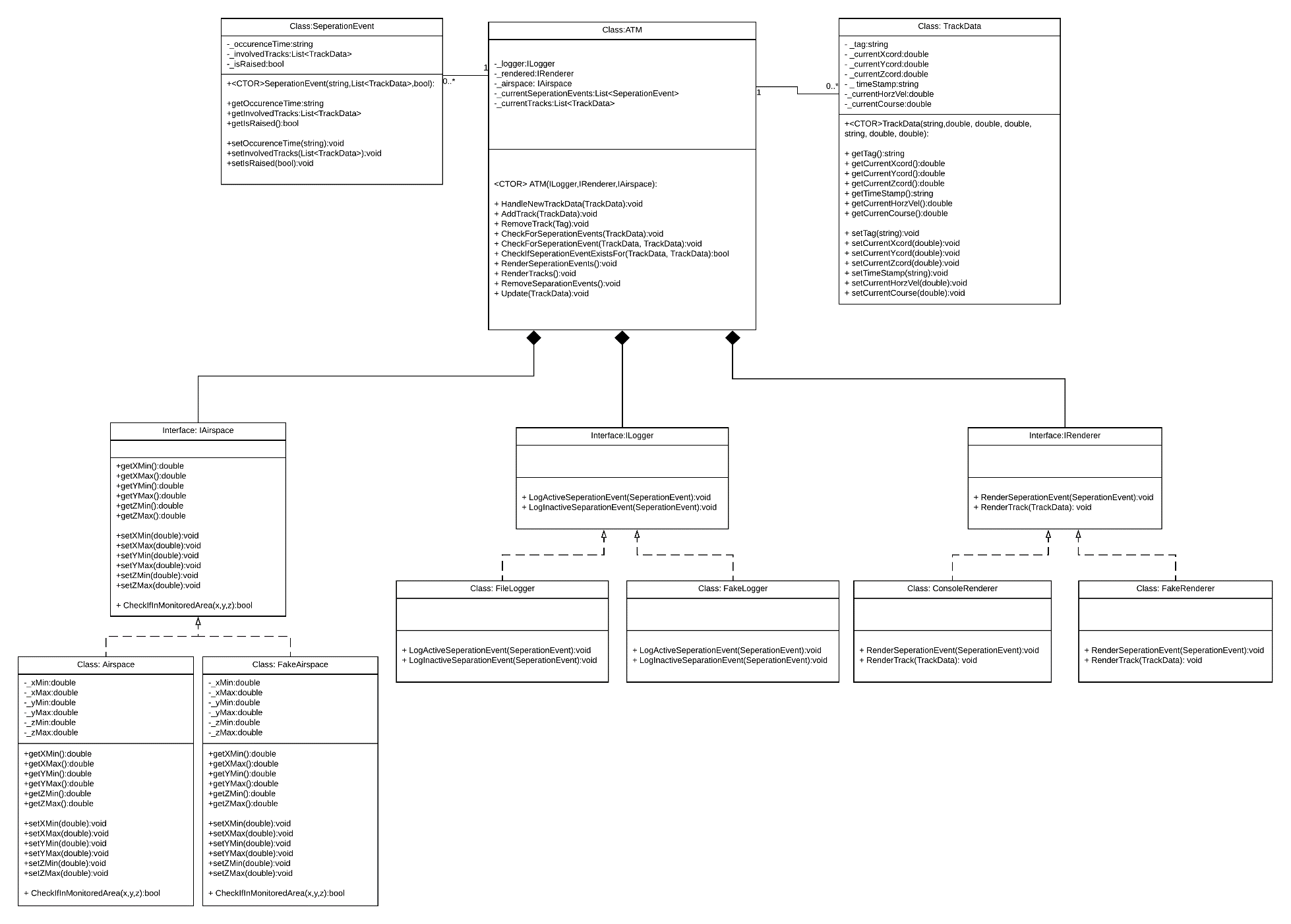
Derudover har vi, som nævnt tidligere, unladet at teste klasserne ConsoleRenderer og FileLogger. For at dette ikke resulterer i ”røde tal” i coverage reporten har vi lavet Coverage Filters i vores dotCoverCoverageConfig-fil, som tager højde for dette. Når der ses bort fra disse har vi opnået 100% Code Coverage.



Figur : Coverage Report

# Appendix

## Appendix 1 – Klassediagram udkast



## Appendix 2 – Sekvensdiagram udkast for handleNewTrackData

