$_{R}APPORT/Billeder/$

# SWT Hand In 3
# Gruppe 20

Aarhus University School of Engineering

## Air Traffic Monitoring

25-04-2018



| Studienummer | Navn | Email-adresse |
|---|---|---|
| 201607589 | Jakob Bonde Nielsen | au567214@post.au.dk |
| 201607110 | Kasper Juul Hermansen | au557919@post.au.dk |
| 201605114 | Stefanie Ruaya Nielson | au554093@post.au.dk |



Link til Git: `https://github.com/TeamTyve/ATM`

Link til Jenkins - UnitTest: `http://ci3.ase.au.dk:8080/job/TeamTyveATMUnitTest/`

Link til Jenkins - integrationstests:

Link til Jenkins - Coverage: `http://ci3.ase.au.dk:8080/job/TeamTyveATMCoverage/`

# Indholdsfortegnelse

# Design 1

## 1.1 Formål

Formålet med denne opgave er at lave et sytem der kan monitorere og præsentere flytraffik over et givent område. Der vil også blive kigget på om flyene i området flyver for tæt. Informationer om flight-tracks vil blive indhentet fra en TransponderReceiver, givet i strings. Dernæst objektifiseres disse strings, og objekterne præsenteres i en konsol. Hele systemet vil blive testet ved hjælp af både unit tests samt integrationstests.

## 1.2 Design

Link til Github: `https://github.com/TeamTyve/ATM`

Der er i opgaven givet en .dll fil der benyttes til at indhente flight-tracks og præsentere disse i strings. Der implementeres derfor en Track Objectification Service, som benyttes til at objektifisere de indhentede strings. Formålet med at få objektifiseret disse strings, er at man får gjort de indhentede strings meget mere overskuelige når de omdannes til objekter, samt at det bliver muligt at arbejde på disse objekter.
Objekterne er fly, som består af henholdsvis et tag (navnet på flyet), x- og y-koordianter, højde i luften, hastighed, retning i grader samt et timestamp.

Når flyene er blevet objektifiseret vil der blive checket for nogle forskellige ting. Først vil der blive checket for, om et fly er i et givent område - kaldet AirSpace. Hvis flyet er i AirSpace, vil flyets informationer blive udskrevet til konsollen. Dernæst vil der blive checket på om de fly, der er i det givne område, flyver for tæt på hinanden. Hvis dette skulle være tilfældet, vil der udskrives en advarsel på konsollen, omkring hvilke fly der flyver for tæt på hinanden, samt hvad tid advarslen fremkom.
**Klassediagram**
Nedenstående ses klassediagrammet for systemet
**Sekvensdiagram**
Nedenstående ses sekvensdiagrammet for systemet

## 1.3 Resultater

Link til Jenkins - Coverage: `http://ci3.ase.au.dk:8080/job/TeamTyveATMCoverage/`
Som det ses på nedenstående billeder er opgaven blevet løst som forventet. Flyene i det ønskede airspace bliver udskrevet, og der kommer en warning når flyene flyver for tæt på hinanden.

# Test 2

## 2.1 Unit tests

Link til Jenkins - Unit tests: `http://ci3.ase.au.dk:8080/job/TeamTyveATMUnitTest/`

Obs! Der er valgt ikke at teste ConsoleLogger.cs, da denne skriver direkte til konsollen, og det er ikke muligt at erstatte statiske metoder. Det vil derfor ikke være muligt at teste om Console.WriteLine() udskriver det korrekte.

## 2.2 Integrationstests

Link til Jenkins - Integrationstests:

Vi har i denne opgave valgt at benytte bottom up, som integrationstest strategi.Denne strategi er valgt grundet at der derved er færre komponenter der skal stubbes, samt at det har virket intuitivt at bruge denne strategi.
Generelt set går bottom up testing ud på at starte fra bunden i det tilhørende dependency tree. De nederste klasser, og derved de klasser der afhænger af mindst vil blive testet først. Hvorfra der hierakisk vil blive testet op igennem træet, indtil alle forbindelser imellem klasserne er blevet tests. Der vil altså blive testet fra submodulerne til hoved-modul(erne). En af fordelene ved bottom up strategien er at det kan eliminere behovet for anvendelse af stubbe i og med at der ikke er nogle højerestående dependencies at tage højde for. Dog er en af ulemperne ved strategien, at man får testet den overordnet funktionalitet sent i processen, og der derfor er risiko for at man skal lave noget af systemet om, senere end nødvendigt.

## 2.3 Diskussion

Til denne hand in var der ikke givet nogle specifikke retningslinjer for hvordan opgaven skulle løses. Dette betød også, at der var stor frihed til hvordan systemet skulle designes. Der blev valgt et design, der gjorde at Track Objectification Service (TOS) - klassen var meget afhængig af de andre klasse, som det også fremgår af det udarbejdede dependency tree. Ud fra dette, kan det også ses at det er et ret "horisontalt"system, i og med at TOS indhenter strings og objektifiserer disse. Når dette er gjort vil der arbejdes direkte på objekterne igennem de andre klasser, i stedet for at bruge en form for pipes and filters arkitektur, hvor der arbejdes i en vandret orden, og objekterne bearbejdes hen ad vejen i stedet for det hele på én gang.
Dette blev i starten valgt, grundet at flere ting arbejdede på én gang. Der blev blandt andet tjekket på om flyet var i AirSpace, men også om flyet fløj for tæt på andre fly. Set i bakspejlet har det formegentlig været mere testbart at benytte en pipes and filters arkitektur, for at undgå at TOS fik så mange dependencies.
Gruppen fik dog løst det på anden måde og fik et tilfredsstillende resultat.

## 2.4 Konklusion

Alt i alt er opgaven blevet løst som forventet. Der er, på trods af, at visse ting ikke er blevet testet, stadig en god code coverage.