OBLIGATORISK AFLEVERING 2
LADESKAB

Gruppe 14
25. marts 2020

Navn	**Studienummer**
Caroline Rhode Nissen	201806156
Ina Bargmann Meyer	201808400
Maria Louise Pedersen	201704927

URL til Jenkins:

- http://ci3.ase.au.dk:8080/job/SWTGroup14HandIn2/

URL til GitHUb:

- https://github.com/BestGroup14/HandIn2_Ladeskab

Indholdsfortegnelse

Indledning

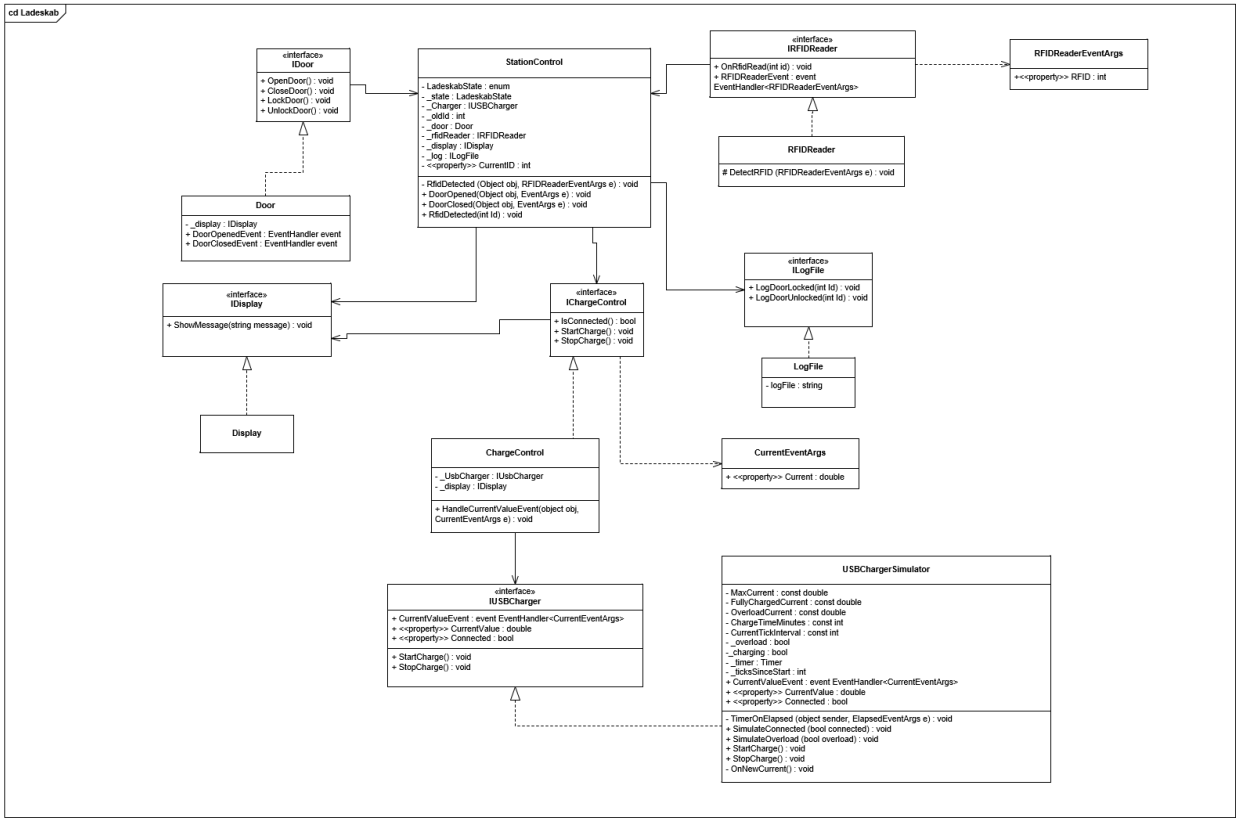
Denne journal vil først og fremmest indeholde beskrivelser af designet for ladeskabssystemet. Designet for ladeskabssystemet er beskrevet ud fra et klassediagram og et sekvensdiagram samt forklaringer, som vil understøtte beskrivelsen af det testbare design, opbygningen af løsningen og systemets opførsel. Derudover indeholder journalen refleksioner over det valgte design, herunder både fordele og ulemper ved dette. Sidst vil der være en beskrivelse samt en refleksion over, hvordan arbejdet er fordelt mellem gruppemedlemmerne, samt en refleksion over arbejdet med brug af et fælles repository og et continuous integration system.

Klasse- og sekvensdiagram samt refleksion over det valgte design

Se vedhæftede visio-fil Gruppe14_HandIn2_diagrammer for at se klassediagrammet tydeligere samt se sekvensdiagrammet.

Klassediagram

Nedenfor på figur 1 ses klassediagrammet for ladeskabet. Klassediagrammet kan også ses i vedhæftede visio-fil.



Figur 1. Klassediagram for ladeskabet

Interfaces

For at undgå at klasserne i systemet er koblet til hinanden, og dermed også er afhængige af hinanden, er der blevet gjort brug af interfaces. Ved at bruge interfaces fås der en meget lav kobling mellem klasserne i systemet, da klasserne ikke vil være afhængige af hinanden, men kun vil være afhængige af et konkret interface i stedet for en bestemt klasse. Dette gør systemet mere testbart, da man ikke længere er afhængig af konkrete klasser, men kun et interface. Havde en klasse haft flere afhængigheder, ville man ikke kunne teste klassen godt, da klassen, som der testes på, automatisk ville få dens afhængigheder med. Dette løses ved, at der lægges et interface ind imellem klassen og dens afhængigheder, for at få en lavere kobling. I klassediagrammet ses der at bl.a. at klassen Door implementerer klassen IDoor. Herved har klassen StationControl fået en lavere kobling, da den har mistet en af sine afhængigheder.

Desuden gør brugen af interfaces det også nemmere at udvide systemet, hvis der bliver behov for det. Dette er uddybet under Open-Closed principle.

Single Responsibilty Principle

Ved at bruge single responsibility princippet, sikres der at en klasse eller en metode kun har en opgave, og dermed kun en grund til at ændre adfærd. Dette betyder også, at klassen eller metoden kun skal have et ansvar. Hvis en klasse har mere end et ansvar, betyder det også, at der vil være flere grunde til at skulle ændre i klassen. Fordelen ved at bruge SRP er at man bl.a. får en bedre testbarhed. Dette gør man, da klasser og metoder er afkoblet fra hinanden.

Open-Closed Principle

I designet er der lagt fokus på Open-Closed principle. For at understøtte princippet er der implementeret interfaces og implementerende klasser. Designet er en fordel, da det sikrer, at det er muligt at implementere flere klasser, der indeholder de samme metoder, men som reagerer eller behandler data anderledes. I dette design anvender vi kun ét display, logfile, door osv., men hvis det senere blev relevant at ladeskabet eksempelvis skulle have flere displays, gør designet, at implementeringen af dette bliver lettere. En anden fordel er, at softwaren bliver mere testbar. Dette beskrives under afsnittet “Test”. En lille ulempe ved princippet er, at det tager lidt længere tid at oprette de enkelte interfaces og de implementerende klasser.

Events

I klassediagrammet ses det, at der er anvendt events der igangsætter og registrerer handlinger. Det er anvendt, da programmet anvender, at der igangsættes bestemte handlinger og at der handles på disse. Fordelen ved at anvende events er, at arbejdet reduceres ift. at der ikke skal implementeres nær så mange interfaces og tilhørende klasser som implementerer disse. Fordelen ved events er, at det er nemt at forbinde events til de metoder, der har forbundet sig til det enkelte event. Ulempen ved at anvende events er, at vi ikke tidligere har anvendt disse og det har derfor krævet mere at skulle sætte sig ind i det, samtidig med at designet udarbejdedes.

Sekvensdiagram

Det tilhørende sekvensdiagram er undladt i rapporten, da skriften bliver for utydelig pga. dets størrelse. Se den vedhæftede Visio-fil for at se diagrammet.

Sekvensdiagrammet er udarbejdet på baggrund af klasserne i ovenstående klassediagram. Diagrammet viser interaktionerne mellem de klasser, der implementeres i koden. Sekvenserne beskriver hvordan ladeskabet validerer RFID, udskriver beskeder samt tjekker ladeforhold, når en telefon er tilsluttet. Enkelte loops indgår i sekvensdiagrammet, hvor der sikres, at ladeskabet anvendes korrekt. Derudover ses de to aktører ”brugeren” og ”logfilen”. Brugeren interagerer med ladeskabet ved enten at åbne eller lukke ladeskabets dør eller ved at indlæse RFID tagget. Samtidig modtager brugeren respons fra ladeskabet, når der er udført interaktion med det. Logfilen indeholder oplysninger om, hvornår ladeskabet er blevet låst og låst op.

Test

Coverage

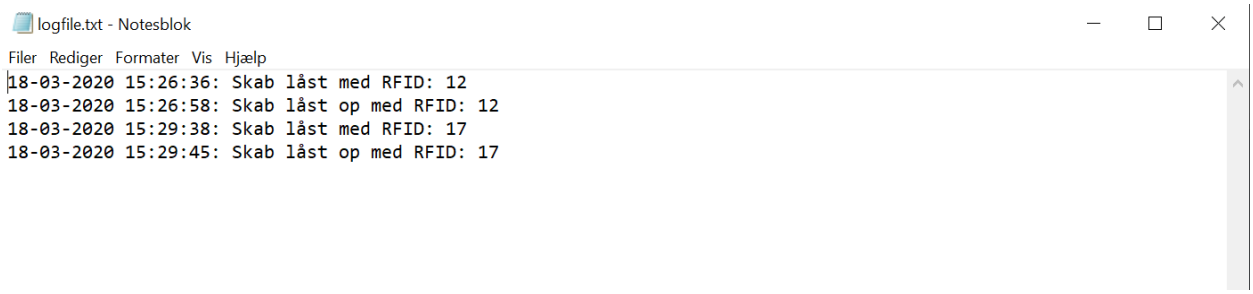
For at sikre kvaliteten af vores test, har vi benyttet coverage, som fortæller hvilke områder af ens kode, der er blevet testet og hvilke områder der ikke er blevet testet. Dette er en nem måde at overskueliggøre alle de områder af ens kode, som skal testes og er dermed en hjælp til, at man husker at teste alle områder. Dog er der nogle ulemper ved coverage, da den kun fortæller, om vi har testet det kode, vi har implementeret, men derimod ikke, om vi har testet nok. Coverage fortæller ikke noget om det, vi har udeladt.

Nsubstitute

I forbindelse med, at vi har implementeret interfaces, så det er nemmere at teste ved hjælp af fakes, har vi benyttet Nsubstitute til at gøre dette for os. Nsubstitute gør lige nøjagtigt det, vi gerne vil have fakes til at gøre automatisk, og kan kun oprettes fakes ud fra et interface. Vi har dermed ikke selv skulle oprette fakes, herunder stubs og mocks, men dette er blevet gjort for os ved hjælp af Nsubstitute.

Test af LogFile

Vi har i gruppen taget et aktivt valg om ikke at unit teste LogFile klassen og dens metoder LogDoorLocked(int Id) og LogDoorUnlocked(int Id) ved hjælp af test frameworket NUnit, men derimod valgt at lave en manuel test, hvor vi har kørt koden og indtastet RFID for både at låse et skab samt at låse skabet op igen med samme RFID. Første gang vi kørte koden og indtastede nummer forventede vi, at der ville blive oprettet en ny fil med navnet logfile.txt, hvori logging af, at skabet blev låst, men også blev låst op igen. Næste gang vi kørte koden forventede vi, at logging ville ske i samme fil, da vi har benyttet AppendText, som skriver til en ny fil, hvis der ikke er nogen fil i forvejen, men skriver til samme allerede eksisterende fil, hvis der er en fil med samme navn i forvejen på computeren. På billedet nedenfor på figur 2 ses det, at første gang, vi kørte programmet blev der skrevet til en fil, som blev oprettet ved navn logfile.txt, hvor der er benyttet RFID 12. Nogle minutter senere ses det, at der er blevet logget til filen med et RFID 17, hvor at der er blevet logget til samme fil som første gang, hvilket også var forventet.



Figur 2. Manuel test af LogFile klassen

I forbindelse med, at vi har valgt at udføre en manuel test af LogFile klassen er der dermed ingen coverage på denne klasse i Jenkins, og vi opnår dermed ikke fuld coverage for vores system.

Arbejdsfordeling i gruppen

I forbindelse med fordelingen af arbejdet imellem os, har vi i gruppen taget et aktivt valg om at arbejde sammen om alle opgaver i denne Hand In. Det vil sige, at vi har samarbejdet om alle opgaver, herunder både designet af koden, selve udformningen af koden samt testen af koden. Dette er gjort for at få en samlet og bedre forståelse for systemet, hvilket har givet det større overblik for alle gruppens medlemmer. Dette har også medført, at ingen af gruppens medlemmer har følt, at de er gået glip af noget, og der er ingen, som har fået hverken større eller mindre opgaver end de andre. Derudover vil vi

også mene, at vi alle har fået lige meget ud af arbejdet idet, vi har skiftes til at gennemføre arbejdet på hver vores computer, og dermed alle har haft fingrene i opgaven. Ovenstående vil vi mene har været en klar fordel for forståelsen af systemet samt læringen. Derudover har det også været muligt at snakke sammen om alle opgaver, hvilket også har fået alle gruppens medlemmer til at reflektere over samtlige beslutninger. Generelt føler vi ikke, at det har været en ulempe at sidde samlet og arbejde, men hvis man skal kigge på ulemperne ved at arbejde sammen om alle opgaver, vil det være, at man får undersøgt tingene en anelse mindre selv, hvis nogen af gruppens andre medlemmer vidste, hvad der skulle til at løse den enkelte opgave. Dog vil vi ikke mene, at dette har været et problem, da vi i gruppen er på samme niveau, og dermed har haft let og svært ved de samme opgaver. En ulempe ved vores fælles arbejde er dog, at vi har brugt mere tid på arbejdet end, hvis vi havde uddelegeret mere, og derved på den måde kunne spare noget tid på selve opgaven, men vi føler alle at udbyttet er givet godt ud i forhold til den tid, vi har brugt.

Refleksion over arbejdet med fælles repository og continuous integration system

Det at arbejde med et fælles repository har været en positiv oplevelse. Det har været overskueligt, og det har været med til at fremme processen, når alle gruppens medlemmer har kunne tilgå samme projekt, og se hele koden hele tiden samtidigt. Vi har kun oplevet at par merge fejl et par enkelte gange, men dette har været lige til at rette i og med, at man kan se de to versioner og sammenligne, og derefter vælge hvilken løsning, man ønsker at benytte. Derudover har vi været gode til at pushe ændringer og tilføjelser hurtigt afsted, så vi alle stort set næsten hele tiden har haft samme version af koden.

I forbindelse med arbejdet med et continuous integration system fik, vi dette op at køre en anelse sent, og vi har derfor været nødt til selv at gå ind og bygge på jenkins. Vi synes dog, at vi har været gode til at være inde og bygge forholdsvis ofte, hvilket er i forbindelse med ændringer. Vi har dog oplevet et par fejl undervis, hvilket har været med til at hjælpe os med vores mangler. Efter at have fået continuous integration op at køre har den bygget af sig selv.