AARHUS UNIVERSITET
INGENIØRHØJSKOLEN

EKSAMEN

Kursus:	I4SWT/ST4SWT – Software Test - Hjemmeeksamen
Eksamensdato:	2019-06-13 kl. 9:00
Varighed:	24 timer
Underviser:	Frank B. Jakobsen
Eksamenstermin:	Sommer 2019

Praktiske informationer:

Digital eksamen
Opgaven tilgås og afleveres gennem den digitale eksamensportal.

Opgavebesvarelsen skal afleveres i ZIP-format

Husk at uploade og aflevere i Digital eksamen. Du vil modtage en elektronisk afleveringskvittering, straks du har afleveret.

Husk at aflevere til tiden, da der ellers skal indsendes dispensationsansøgning.

Husk angivelse af navn og studienummer på alle sider, samt i dokumenttitel/filnavn.

Hjælpemidler:
Alle hjælpemidler må benyttes, herunder internettet som opslagsværktøj, men opgaven er en personlig opgave.

Indledning

Denne tekst definerer eksamensopgaven i faget I4SWT/ST4SWT, Software test, for indeværende eksamenstermin på diplomingeniørstudiet i Informations- og Kommunikationsteknologi hhv. Sundhedsteknologi på Ingeniørhøjskolen Aarhus Universitet.

Eksamensopgaven ligger til grund for den mundtlige eksamen i faget. Med udgangspunkt i opgavens besvarelse vil du blive eksamineret i fagets indhold.

Opgavens besvarelse (”afleveringen”) skal bestå af netop 2 dele:

- En *journal* i PDF-format, hvor du besvarer de spørgsmål der er givet i de enkelte delopgaver.

- En *implementering* i en Microsoft Visual Studio-solution, som indeholder implementeringen af det system og de tests, der efterspørges i opgaven.

Journalen og implementeringen skal samles og afleveres i netop 1 samlet ZIP-fil. Filen skal være navngivet efter følgende format: `i4swt-191-`*`studienummer`*`-`*`fuldeNavn`*`.zip`, hvor *`studienummer`* og *`fuldeNavn`* skal erstattes af studienummer hhv. fulde navn. Et eksempel på en korrekt navngivet fil er ”`i4swt-191-20111223-TroelsFedderJensen.zip`”.

Formålet med opgaven er at give dig mulighed for at demonstrere din kunnen og viden inden for fagets læringsmål. Der lægges derfor vægt på at opgavens løsning demonstrerer din erhvervede viden inden for fagets forskellige emner.

Opgaven er inddelt i et antal delopgaver, som bygger på hinanden. Det fremgår af hver enkelt delopgave hvad der skal afleveres i journalen hhv. implementeringen af opgaven. Al implementering skal finde sted i den Microsoft Visual Studio solution der afleveres.

Bemærk:

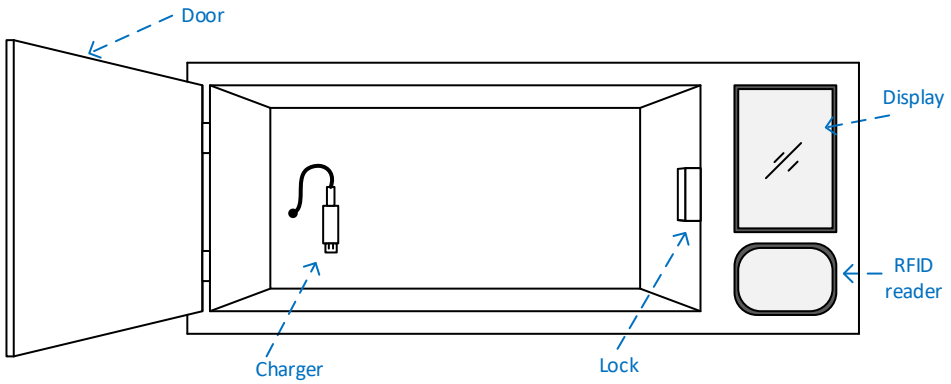
- Du må *ikke* anvende et offentligt tilgængeligt Git repository i forbindelse med løsningen af denne opgave. Det indebærer risikoen for at din opgavebesvarelse bliver synlig for andre deltagere i denne eksamen, hvilket ikke er tilladt.

- Du må *ikke* oprette et Jenkins job på den i undervisningen anvendte CI-server (eller nogen anden CI-server) i forbindelse med løsningen af denne opgave. Det indebærer risikoen for at din opgavebesvarelse bliver synlig for andre deltagere i denne eksamen, hvilket ikke er tilladt.

1 Ladeskab til mobiltelefon

Denne opgave omhandler design, implementering og test af software til et ladeskab til en mobiltelefon, som skal opstilles i omklædningsrummet i en svømmehal eller lignende. Formålet med ladeskabet er, at en bruger kan tilslutte sin mobiltelefon til opladning i ladeskabet, efterlade sin telefon her og senere afhente den igen. Som identifikation tænkes en RFID tag der hænger på badebåndet eller nøglen til et tøjskab.

En principskitse af ladeskabet er vist herunder:



Figur 1: Principskitse af ladeskabet

Ladeskabet tænkes brugt som følger (det antages at skabet ikke er i brug):

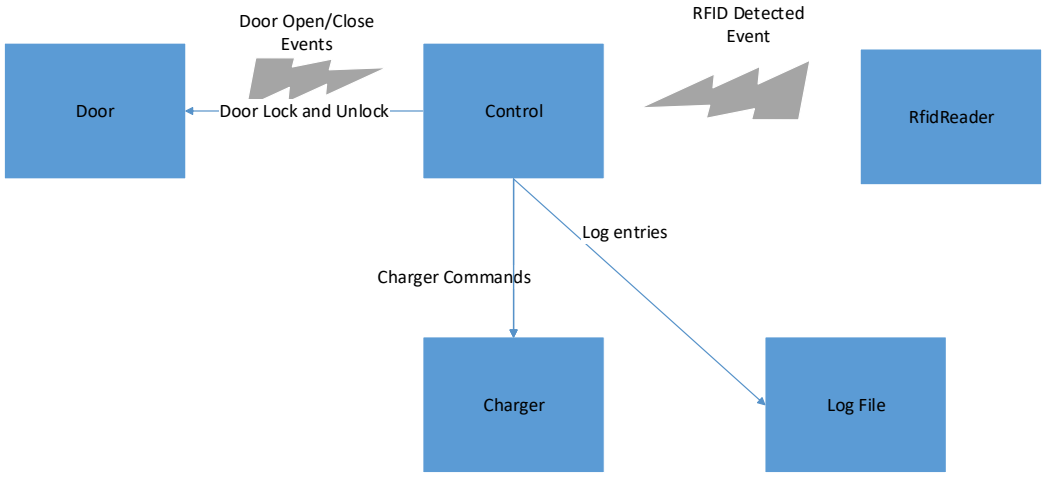
1. Brugeren åbner lågen på ladeskabet
2. Brugeren tilkobler sin mobiltelefon til ladekablet.
3. Brugeren lukker lågen på ladeskabet.
4. Brugeren holder sit RFID tag op til systemets RFID-læser.
5. Systemet aflæser RFID-tagget. Hvis ladekablet er forbundet, låser systemet lågen på ladeskabet, og låsningen logges. Skabet er nu *optaget*. Opladning påbegyndes.

Brugeren kan nu forlade ladeskabet og komme tilbage senere. Herefter fortsætter den tiltænkte brug som følger:

6. Brugeren kommer tilbage til ladeskabet.
7. Brugeren holder sit RFID tag op til systemets RFID-læser.
8. Systemet aflæser RFID-tagget. Hvis RFID er identisk med det, der blev brugt til at låse skabet med, stoppes opladning, ladeskabets låge låses op og oplåsningen logges.
9. Brugeren åbner ladeskabet, fjerner ladekablet fra sin telefon og tager telefonen ud af ladeskabet.
10. Brugeren lukker skabet. Skabet er nu *ledigt*.

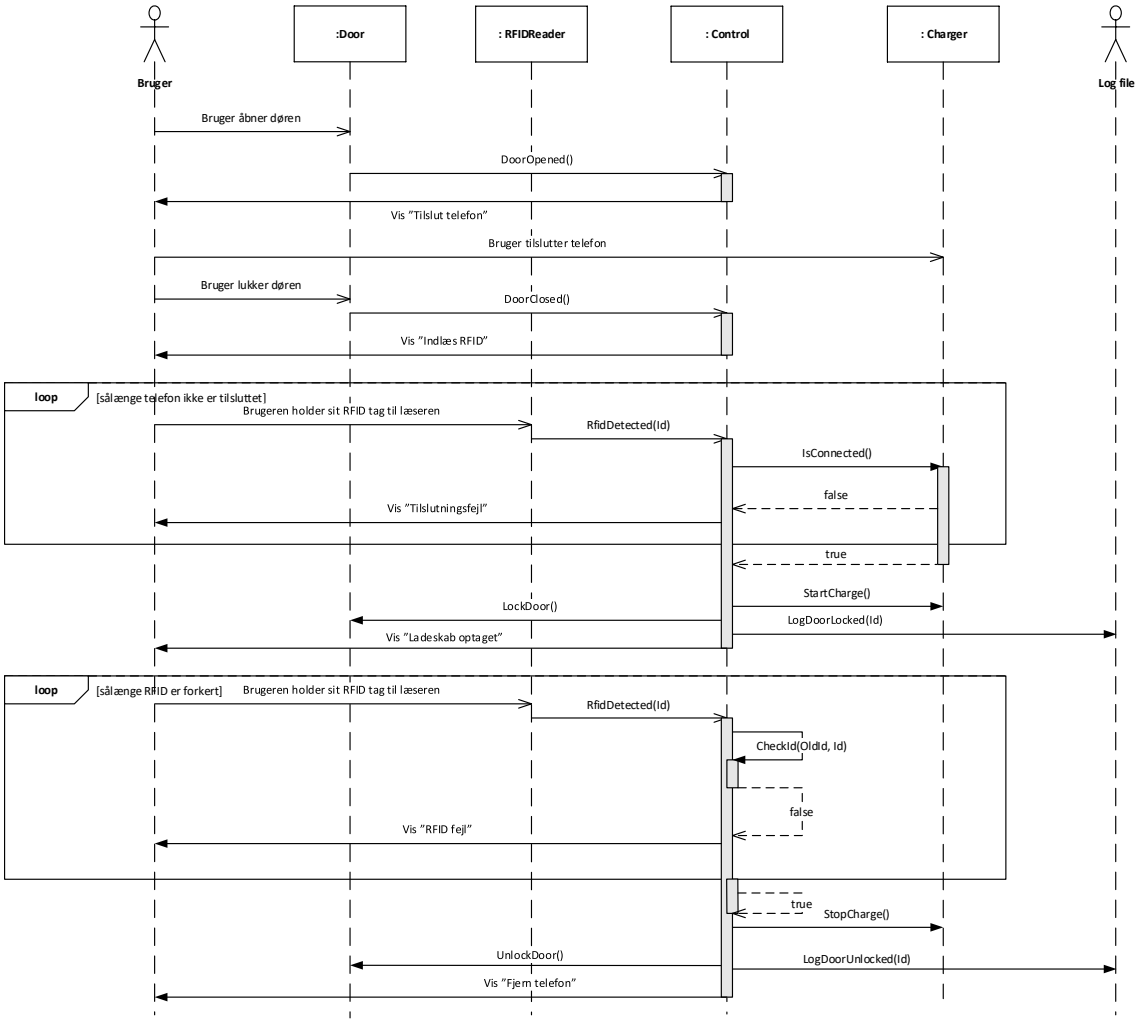
Når låsning/oplåsning finder sted, skal begivenheden logges til en fil inkl. timestamp (timer, minutter og sekunder), det aktuelle RFID samt en beskrivende tekst for begivenheden.

Et løseligt udkast til et design (*ikke* et UML klassediagram) er givet nedenfor:



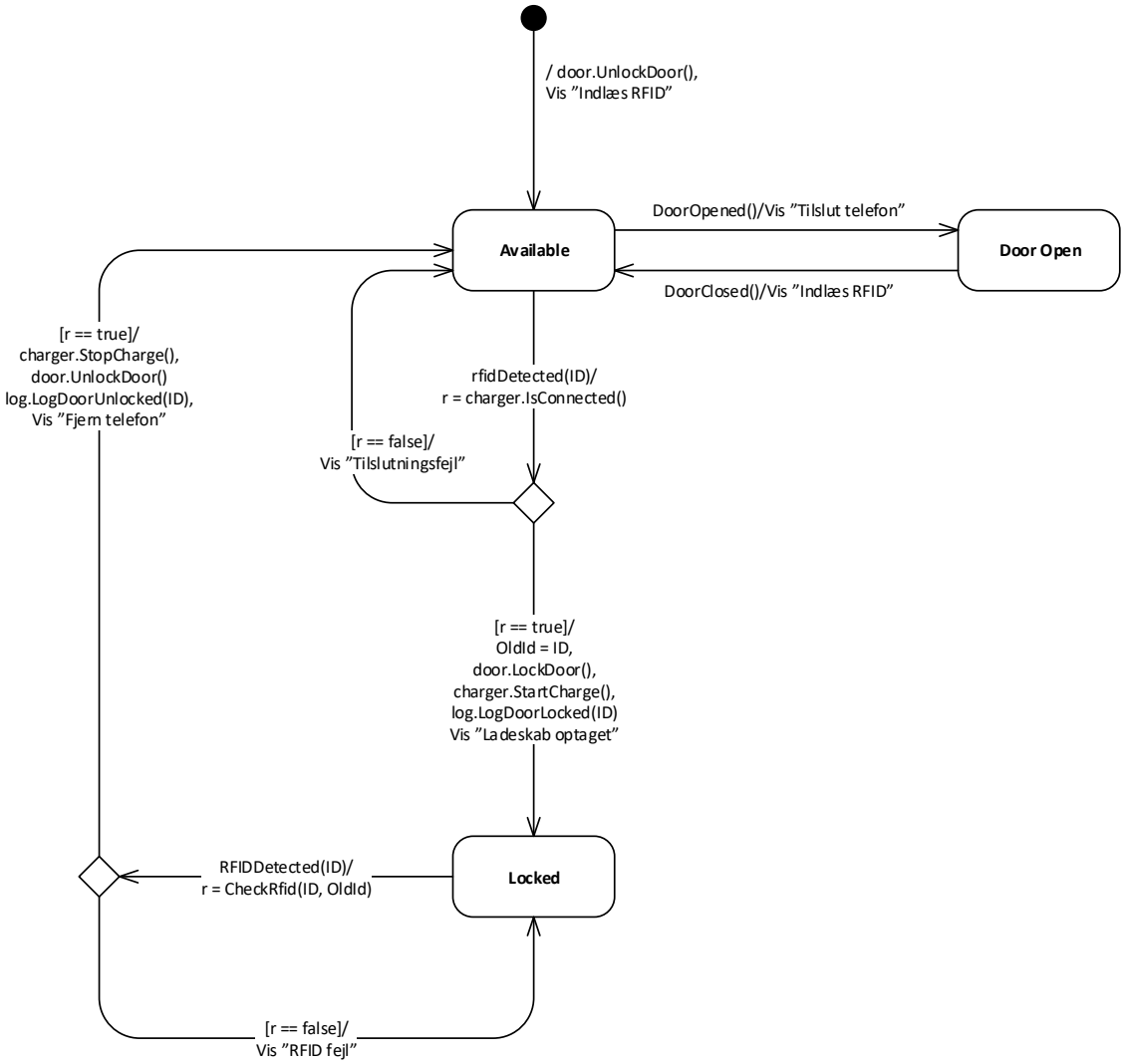
Figur 2: Designskitse

Klasserne `Door`, `Charger` og `RFIDReader` på Figur 2 repræsenterer komponenter af samme navne på Figur 1. `Log File` kan ikke ses, men er indbygget i systemet, og kan inspiceres med software, der ikke er en del af denne opgave. Klassen `Control` er controller-klassen for systemet. Klassernes interaktion er vist på Figur 3. Bemærk, at dele af kommunikationen er synkron (metodekald) mens andre er asynkron (events).



Figur 3: Sekvensdiagram for opladning af mobiltelefon

Klassen `Control`'s virkemåde kan beskrives vha. et UML tilstandsdiagram som følger:



Figur 4: Tilstandsmaskine for klassen Control

Et løseligt udkast til dele af implementeringen af klassen Control er givet i filen `Control.Handout.cs`.

På de næste sider følger delopgave 1 til 11.

1.1 Delopgave 1

Vis i din journal et testbart design (UML klassediagram) for softwaren til ladeskabet, med udgangspunkt i designudkastet i Figur 2. Redegør ligeledes for, hvad du mener der gør dit design testbart.

1.2 Delopgave 2

Implementér klassen `Control`, med udgangspunkt i sekvens- og tilstandsdiagrammerne Figur 3 og Figur 4 ovenfor og det udleverede, delvise forslag til source code.

1.3 Delopgave 3

Implementér de unit tests, du finder nødvendige for at teste klassen `Control`. Vis resultatet af de gennemførte unit tests i din journal.

1.4 Delopgave 4

Redegør i din journal for strukturen i din Visual Studio solution (vis f. eks. et screenshot af strukturen), og begrund dit valg af struktur.

1.5 Delopgave 5

Vis i din journal eksempler på, hvor du i dine unit tests har draget nytte af det testbare design, du specificerede i Delopgave 1.

1.6 Delopgave 6

Vis i din journal eksempler på anvendelsen af en *adfærdsbaseret* test, og redegør for forskellen på denne type test og en *tilstandsbaseret* test.

1.7 Delopgave 7

Vis i din journal eksempler fra din unit test, hvor du bruger et isolation framework. Redegør endvidere for fordele og ulemper ved brugen af et sådant framework.

1.8 Delopgave 8

Redegør i din journal for, hvordan du fandt frem til, at netop de unit tests du har implementeret er nødvendige og tilstrækkelige for at teste klassen `Control`.

1.9 Delopgave 9

Implementér og unit test de dele af dit design, der håndterer skrivning til en log-fil i forbindelse med læsning af RFID. Dokumentér i din journal, hvilke overvejelser dette giver anledning til.

1.10 Delopgave 10

Antag at systemets øvrige klasser er implementeret og unit testet. Vis i din journal et dependency-træ til brug for integrationstests af systemet. Redegør endvidere for, hvad der er særligt vigtigt at teste i en integrationstest.

1.11 Delopgave 11

En Continuous Integration-server (CI-server, f. eks. Jenkins) vil typisk blive brugt til at udføre jobs, som indeholder en række pipeline steps, dvs. forskellige aktiviteter knyttet til sikringen af god kvalitet i software. Et eksempel på et sådant pipeline step kunne være ”Kør unit tests”.

Redegør i din journal for, hvilke pipeline steps du vil konfigurere en CI-server til at udføre i testen af din software, og i hvilken rækkefølge du vil udføre disse steps.

Redegør ligeledes for, om de enkelte pipeline steps er forudsætning for andre pipeline steps eller ej.