Forår 2016

02324 – Videregående programmering

Gruppe nr.: 19

Afleveringsfrist: Lørdag den 27/02-2016 kl. 05:00

[PROJEKTOPGAVE – CDIO1]

Silas El-Azm Stryhn
s143599

Ramyar Hassani
s143242

Mikkel Hansen
s143603

Frank Thomsen
s124206

Martin Rødgaard
s143043

Denne rapport er afleveret via CampusNet

Denne rapport indeholder 17 sider inkl. denne side.

Timeregnskab

Timeregnskab					
	Silas	Martin	Mikkel	Frank	Ramyar
Analyse	1	2	3	6	2
Design	6	1	6	2	4
Implementering	18	20	40	15	15
Test	1	1	2	4	2
Dokumentation	10	7	7	4	6
I alt	**36**	**31**	**58**	**31**	**28**

Indholdsfortegnelse

Resumé

Rapporten indeholder et analyseafsnit, der kort beskriver vores tanker før projektet gik i gang samt en kravspecifikation for opgaveoplægget. Dernæst forefindes der et designafsnit, hvis formål er at give læseren et overblik over hvad systemet indeholder og hvordan det er bygget op mht. til klasser. Rapporten indeholder også et implementeringsafsnit, hvor der findes en kort forklaring om hvad der er brugt, og hvad der mangler. Afsluttende indeholder rapporten et testafsnit hvor de tests vi har lavet bliver gennemgået og forklaret.

Indledning

Denne rapport præsenterer en kort gennemgang af udviklingen bag et bruger-administrationsmodul med dertilhørende simpel brugergrænseflade. Vi er blevet bedt om at udvikle et bruger-administrationsmodul der kan benyttes til håndteringen af brugere/operatører, der skal have adgang til et afvejningssystem. Dette projekt er designet efter 3-lags modellen, hvor vi har forsøgt at efterkomme principperne bag denne. Vi har derfor udviklet systemet med et datalag, et funktionalitetslag, og til sidst et grænsefladelag. Rapportens basale formål er at give en grundlæggende forståelse for tankegangen igennem forløbet, hvilke overvejelser der er gjort og hvordan bruger-administrationsmodulet er stykket sammen. Rapporten gennemgår et kort analyseafsnit, der har til formål at vise tankerne før projektets start samt udarbejdelsen af diverse diagrammer, som skal give læseren et indblik i systemets opbygning.

Hovedafsnit

Analyse

Vi har i denne delopgave udarbejdet et lille analyseafsnit, der blot har til formål at vise vores tanker og overvejelser før vi gik i gang med projektet. Dertil er der lavet en kravspecifikation, som undergår en analyse af kravene til systemet.

Kravspecifikation

Vi har udarbejdet en kravspecifikation ud fra opgaveformuleringen. I denne kravspecifikation inddeler vi først vores krav i funktionelle krav og ikke funktionelle krav for derefter at bruge MoSCoW metoden på dem og kommentere vores valg.

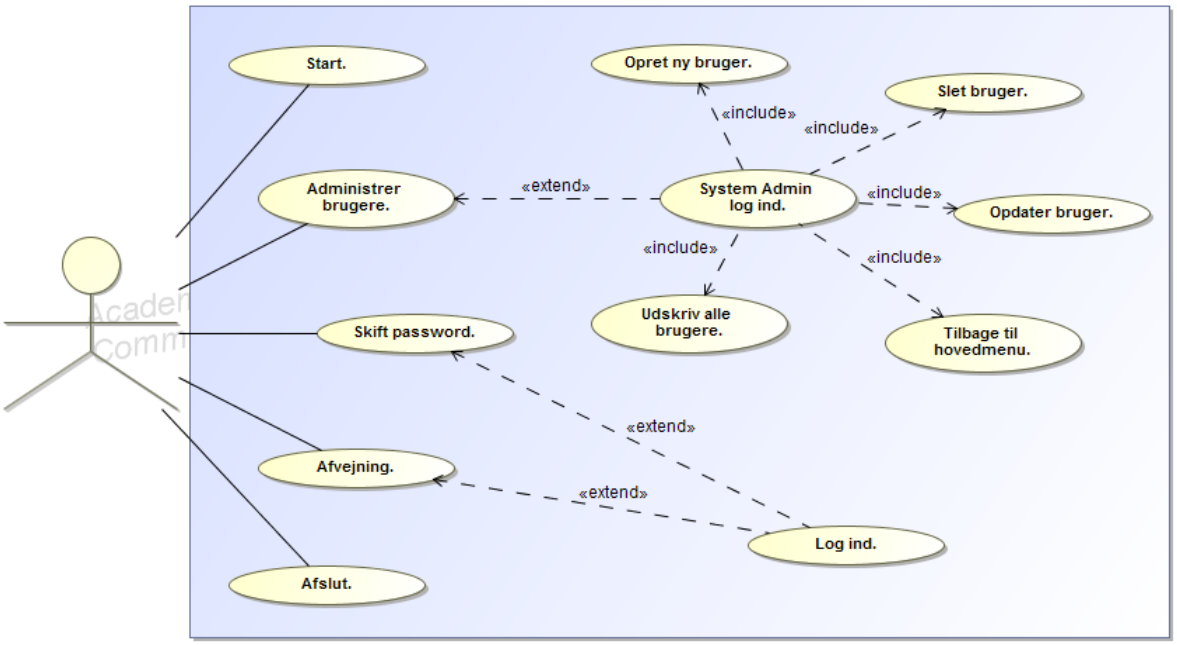
Programmet			
Programmet skal være designet efter principperne i 3-lags modellen.	Ikke funktionelt	Must have	Vi føler dette er et vigtigt krav da kunden specifikt nævner denne model.
Brugergrænsefladen skal være simpel.	Ikke funktionelt	Must have	Dette krav er vigtigt fordi kunden gerne vil have det simpelt. Simpel er dog ikke særlig godt defineret, men vi har arbejdet ud fra opgavens eksempel.
Hovedmenuen skal være en konsol-applikation.	Ikke funktionelt	Must have	Kunden nævner at hovedmenuen skal være en konsol applikation, dette synes vi er vigtigt at rette os efter og mener derfor at dette er et must have krav.
Hovedmenuen skal give mulighed for at starte en operatør administrations dialog.	Funktionelt	Must have	Da programmet er et "bruger-administratonsmodul" synes vi både dette er et funktionelt krav og et must have.
Hovedmenuen skal give mulighed for at brugere kan ændre deres passwords.	Ikke funktionelt	Should have	Dette krav er ikke nødvendigt for at programmet kan fungere, men vi antager at dette er noget kunden synes er vigtigt.
Hovedmenuen skal give mulighed for at brugere kan tilgå afvejnings applikationen.	Funktionelt	Must have	Vi mener at afvejnings applikationen er en vigtig del af programmet, derfor har vi sat dette krav til must have.
Administrator			

En Administrator skal kunne oprette en operatør.	Ikke funktionelt	Should have	Vi har antaget at kunden synes dette krav er vigtigt derfor har vi valgt at sætte dette krav til should have.
En administrator skal kunne læse en operatør.	Ikke funktionelt	Should have	Vi synes det er vigtigt at administratoren kan se hvilke brugere som findes derfor har vi sat dette krav til should have.
En administrator skal kunne opdatere en bruger.	Ikke funktionelt	Could have	Da det er relativ sjældent at folk ændrer navn eller andre oplysninger synes vi ikke dette krav er særlig vigtigt.
En administrator skal kunne slette operatører.	Ikke funktionelt	Should have	Ligesom vi antog at kunden synes det er vigtigt at kunne tilføje nye brugere antog vi også at han vil kunne fjerne brugere og hermed deres adgang til applikationen.
Operatører			
En operatør skal bestå af et entydigt userID (i form af et tal) mellem 11 og 99.	Ikke funktionelt	Must have	Vi antager at grunden til at kunden ønsker et userID er at de ønsker at kunne skilte mellem brugerne og dette synes vi er vigtigt.
En operatør skal bestå af et navn imellem 2 og 20 tegn.	Ikke funktionelt	Should have	Vi synes også det er vigtigt og mere brugervenligt operatørerne har et navn tilkoblet deres bruger derfor har vi valgt at dette er et should have krav.
En operatør skal bestå af et cpr. nummer.	Ikke funktionelt	Could have	Da vi allerede implementer et unikt userID at skilte mellem vores operatører på, synes vi ikke cpr nummeret er specielt vigtigt.
En operatør skal bestå af et password. Som overholder reglerne for passwords på DTU. Se https://password.dtu.dk/	Ikke funktionelt	Should have	Passwords er altid en god ting især hvis det kun er visse brugere som skal have adgang. Kodeordets sværhedesgrad er dog kun for at forstærke kodeordene.
En operatør skal tildeles et genereret password når brugeren oprettes.	Ikke funktionelt	Could have	Vi synes ikke det er særlig vigtigt om brugeren modtager et password eller laver sit eget når brugeren oprettes derfor synes vi at dette er et could have krav.
Afvejnings applikationen			
Afvejnings applikationen skal kun være tilgængeligt for administratorer og operatører som logger ind med password.	Ikke funktionelt	Must have	Da kunden ønsker et brugersystem må vi antage at det er særdeles vigtigt at det kun er disse brugere som har adgang. Det er dog ikke et funktionelt krav men vi har valgt at det er et must have krav.
Afvejnings applikationen skal kunne udregne nettovægten med formlen: Brutto=Netto+Tara	Funktionelt	Must have	Vi synes dette er et vigtigt krav da det som sådan er den eneste funktionalitet at programmet derfor har vi selvfølgelig valgt

(TARA == emballage) ved at brugeren indtastet en bruttovægt og en Tara vægt.			at sætte dette krav til must have.

Use Case diagram

Vores Use Case diagram giver et overblik over hvilke Use Cases vores system indeholder. Diagrammet viser os overordnet hvad systemet indeholder og hvilke muligheder aktøren har. I dette tilfælde kan aktøren tage to roller: som operatør, der får muligheden for at afveje ved login, eller system admin som får muligheden for at administrere brugere.



Use Case beskrivelser

Vi har udvalgt følgende Use Cases og beskrevet dem. Grundet tidspres, nåede vi desværre ikke flere.

Use Case Name:	Start
Use Case ID:	1
Actors:	Operatør
Description:	Brugeren starter systemet op.
Preconditions:	Systemet er i gang.
Postconditions:	Brugeren har valgt, Administrer brugere / Skift password / Afvejning eller Afslut
Main flow:	1. Administrer brugere. 2. Skift password. 3. Afvejning. 4. Afslut.
Alternative Flow:	1. Systemet lukkes ned

Use Case Name:	Administrer brugere, log ind
Use Case ID:	2
Actors:	Operatør
Description:	Log ind som administrator
Preconditions:	Kun administrator kan logge ind.
Postconditions:	Administrator har logget ind
Main flow:	1. Log ind

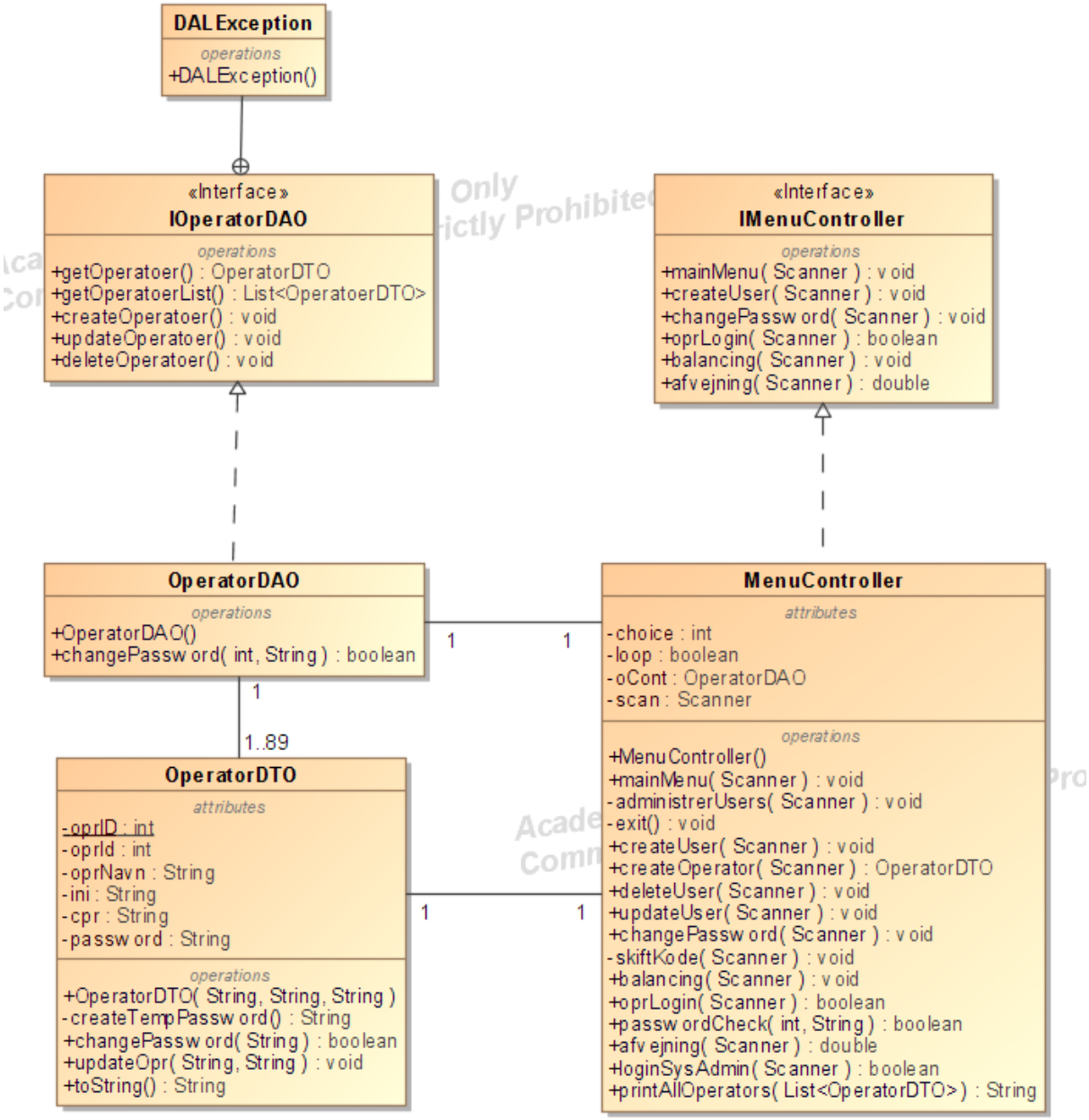
Use Case Name:	System Administrator
Use Case ID:	3
Actors:	Administrator
Description:	Administrator kan administrere brugerne
Preconditions:	logget ind som administrator
Postconditions:	Administrator har valgt, Opret ny bruger / Slet bruger / Opdater bruger / Udskriv alle brugere eller Tilbage til hovedmenu
Main flow:	1. Opret ny bruger. 2. Slet bruger. 3. Opdater bruger. 4. Udskriv alle brugere. 5. Tilbage til hovedmenu.
Alternative Flow:	1. Tilbage til hovedmenu.

Design

I design afsnittet vil man kunne se diagrammer, der er med til at give et overblik over hvad systemet indeholder af klasser og relationerne mellem disse. Først gennemgår vi klassediagrammet, og dernæst gennemgår vi design-sekvensdiagrammet, der er delt op i to forskellige diagrammer, som fokuserer på to af metoderne i vores system. Dette er gjort fordi vi mener, at det er disse to metoder som er de mest interessante.

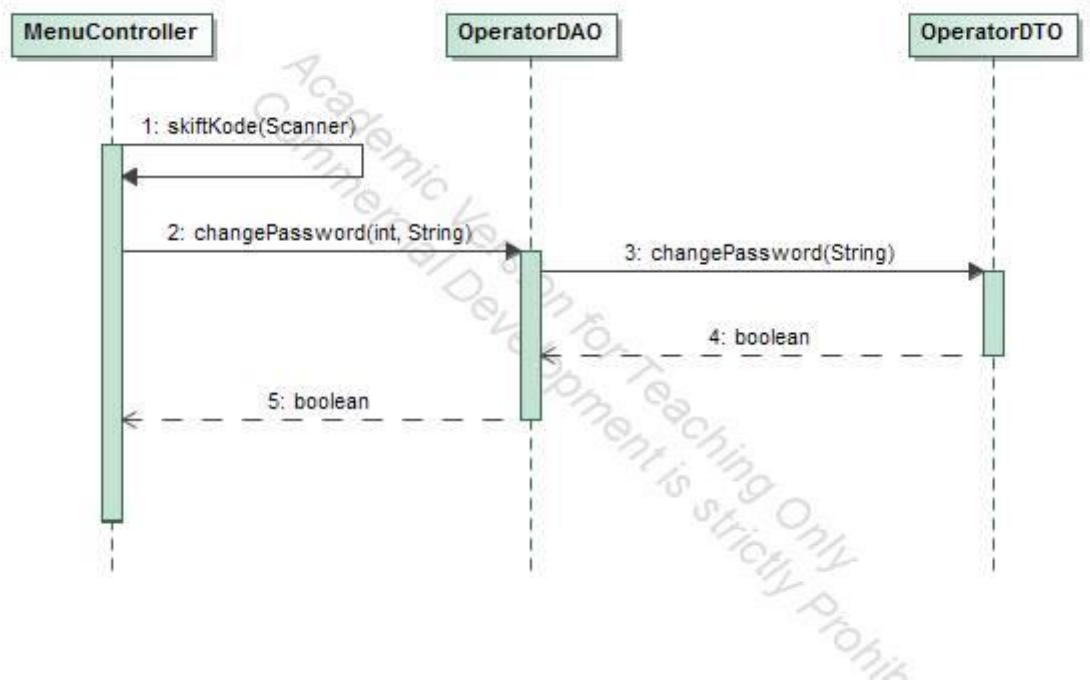
Design-klassediagram

Klassediagrammet viser os hvilke klasser systemet indeholder, og hvad de forskellige klasser indeholder af metoder og attributter. I denne delopgave er systemet lavet med to interfaces hvor deres tilhørende controllerklasser benytter deres metoder. Dertil har vi en dataklasse, som indeholder alt information om vores operatører.



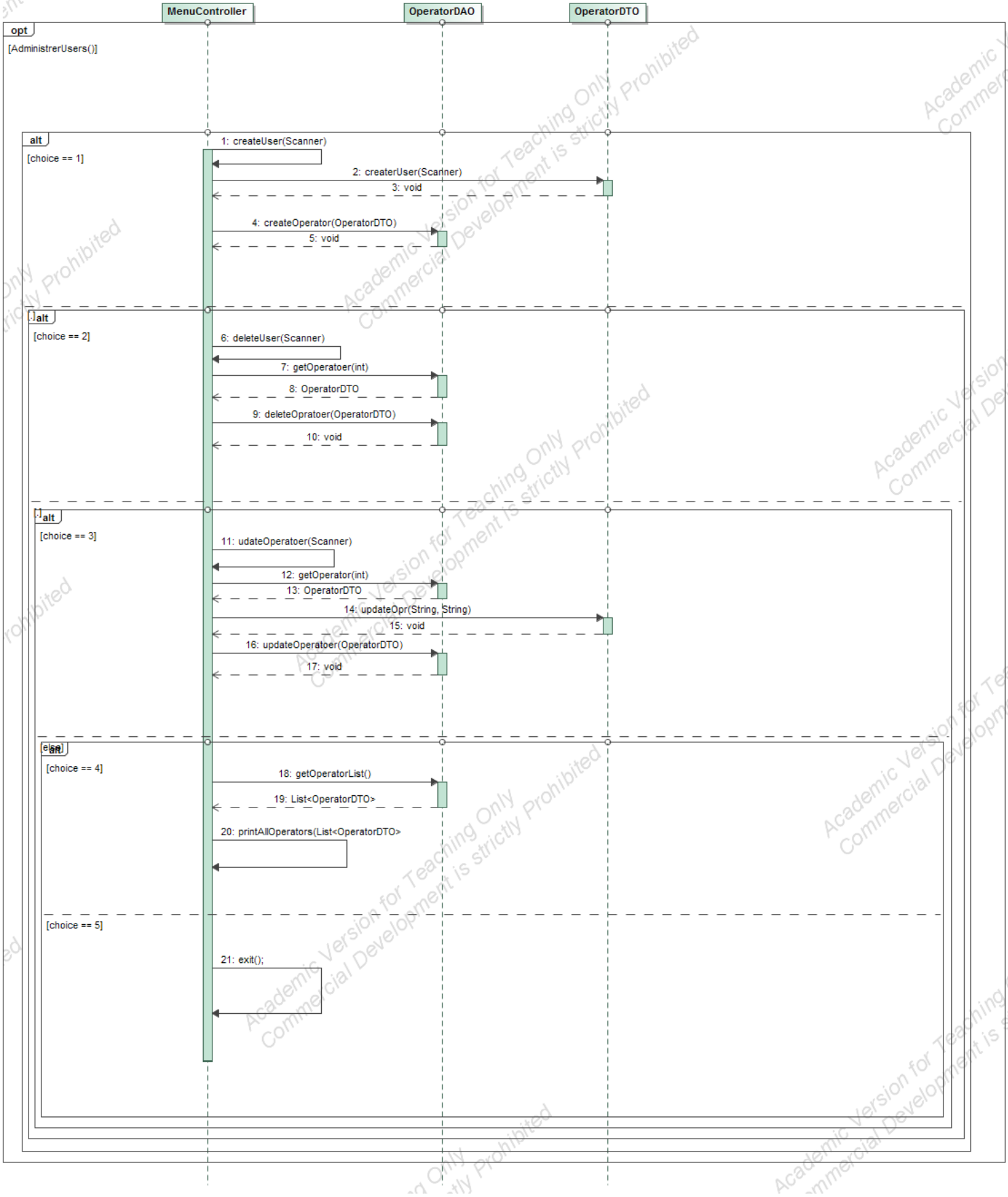
Design-sekvensdiagram - skiftkode()

På sekvensdiagrammet for at skifte kode ses det, at MenuController kalder *skiftKode(Scanner)* til sig selv, hvorefter der bliver kaldt *changePasswod(int, String)* til OperatorDAO. Den int er bruger nummeret for den bruger der skal have skiftet kode og String er den nye kode. Så kalder OperatorDAO *changePassword(String)* til OperatorDTO der så ændrer den gemte kode på brugeren. Begge metoder returnerer en boolean for at se hvor vidt det lykkedes at skifte koden.



Design-sekvensdiagram - administrerUsers()

Nedenstående diagram er delt op i flere if-else statements der sørger for at tage højde for hvad brugeren af systemet vælger. Hvis brugeren vælger "1" betyder det, at der blev valgt at oprette en ny bruger. Her kalder MenuController *createUser(Scanner)* til OperatorDTO som opretter brugeren og derefter kalder den *createOperator(OperatorDTO)* til OperatorDAO som sørger for, at den nye bruger bliver gemt i vores HashMap.
Hvis brugeren vælger "2" har man valgt at slette brugere. Her kalder MenuController *deleteUser(Scanner)* til sig selv hvorefter den kalder *getOperator(int)* til OperatorDAO og får en OperatorDTO retur. Til sidst kalder MenuController *deleteOperator(OperatorDTO)* til OperatorDAO der så sørger for, at få fjernet brugeren.
Hvis brugeren vælger "3" er udate blevet valgt. Her kalder MenuController *updateOperator(Scanner)* til sig selv efterfulgt af *getOperator(int)* til OperatorDAO. Her bliver en OperatorDTO så returneret. Når MenuControlleren har fundet den rigtige OperatorDTO kalder den *updateOpr(String, String) til OperatorDTO.* De to Strings er henholdsvis oprNavn og CPR-nummer. Her bliver der blot returneret en void. Dog er det værd at nævne, at brugerens password også bliver nulstillet til "Test1234". Bagefter kalder MenuController *updateOperator(OperatorDTO)* til OperatorDAO der så sørger for at opdatere vores HashMap.
Vælger brugeren "4" er det at printe alle operatørene ud. Her kalder MenuControlleren *getOperatorList()* til OperatorDAO, hvilket returnerer en List<OperatorDTO>. Derefter kalder MenuController *printAllOperators(List<OperatorDTO>)* der printer alle pladser i Listen ud via en forløkke. Vælger man "5" kører MenuControlleren *exit()* der bare sætter en boolean til false, så vores do-while løkke i *mainMenu(Scanner)* stopper med at køre.


MenuController
OperatorDAO
OperatorDTO
opt
[AdministrerUsers()]
alt
[choice == 1]
1: createUser(Scanner)
2: createrUser(Scanner)
3: void
4: createOperator(OperatorDTO)
5: void
alt
[choice == 2]
6: deleteUser(Scanner)
7: getOperatoer(int)
8: OperatorDTO
9: deleteOpratoer(OperatorDTO)
10: void
alt
[choice == 3]
11: udateOperatoer(Scanner)
12: getOperator(int)
13: OperatorDTO
14: updateOpr(String, String)
15: void
16: updateOperatoer(OperatorDTO)
17: void
alt
[choice == 4]
18: getOperatorList()
19: List<OperatorDTO>
20: printAllOperators(List<OperatorDTO>
[choice == 5]
21: exit();

Implementering

Se bilag for

Generelt

Til vores implementering har vi brugt Github, hvor vi har lavet et repository, så alle gruppemedlemmer har kunne arbejde på det samme projekt. Dette gør det lettere at kunne gå tilbage i kodningen, hvis der opstår en fejl, og det gør det lettere at holde styr på, hvad vi hver især har lavet på projektet. Vi har i vores kode prøvet at sætte det op, på den mest overskuelige måde, og har herefter kommenteret stort set alt i de forskellige klasser, så det er nemmere at forstå vores kode.

Manglende implementation

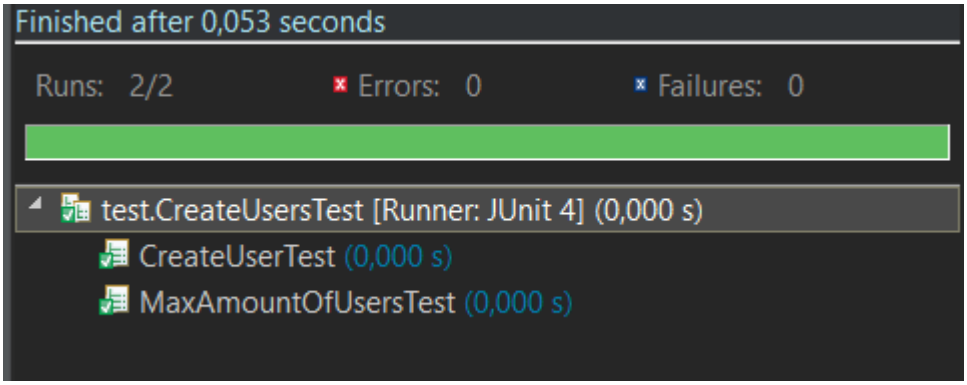
Vi mangler ikke at implementere noget i forhold til opgaveoplægget. Der er selvfølgelig ting vi kunne have implementeret anderledes for måske at få en "bedre" funktionalitet, men hvor vi alligevel valgte at holde os til det præcise opgave oplæg. Et eksempel på dette kunne være deleteOperator(OperatorDTO) hvor vi har lavet det så den bare sletter operatøren. Ulempen ved dette er, at når man først har slettet en operatør, vil man aldrig kunne bruge det bruger nummer den blev givet da den blev oprettet. En løsning til dette kunne være, at man "opdaterer" den bruger man gerne vil slette til at være null i alle parametre undtagen bruger nummer. Ideen med det er, at man så senere har muligheden for at benytte sig af updateOperator(OperatorDTO) til at ændre den tomme brugers værdi til at være en reel bruger igen. På den måde har man også mulighed for at lave flere brugere efter man har nået loftet på 89 ved at slette en bruger og oprette en ny.
En anden ting der kunne være blevet gjort bedre er, at vi mangler en enkelt sikkerhedsting i forhold til passwords. Vi fik desværre ikke specialtegn med, så derfor besluttede vi, at man i stedet skulle have et password der var minimum 6 tegn langt, indeholde minimum 1 stort tegn, 1 lille tegn og 1 tal. Det har vi valgt fordi det stadig lever op til DTU's regler for et sikkert password hvis man alligevel vælger ikke at benytte sig af specialtegn.

Test

Generelt

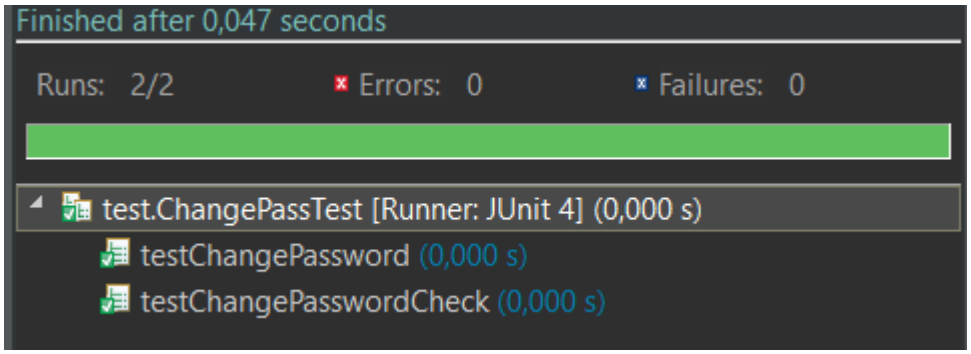
For at teste vores implementation har vi valgt at lave et par tests. Vi har valgt at teste de steder hvor vi har lavet flest restriktioner for brugeren. Det er gjort for at sikre os, at disse restriktioner virker optimalt. Her er der tale om når man opretter nye brugere og når man skal skifte password. Når man opretter brugere bliver der testet på hvor vidt vi har nået loftet for hvor mange bruger numre vi må tage i brug. I følge opgavebeskrivelsen skal bruger numrene være fra 10 - 99 (hvor nr 10 er SystemAdmin).
Hvad passwords angår var det foreslået, at man fulgte DTU's regler for en sikker kode. Det betyder altså, at man skal opfylde minimum 3 ud af de 4 følgende regler. Passwordet skal indeholde minimum 1 special tegn. Passwordet skal indeholde minimum 1 stort tegn. Passwordet skal indeholde minimum 1 lille tegn. Passwordet skal indeholde minimum 1 tal.
Derudover skal alle passwords minimum være 6 tegn langt.

JUnit tests




```
Du har desværre overskredet maximum antal brugere.
```

Her har vi vores test af CreateUser. Vi har valgt at lave to tests til det. Den første tester bare på hvor vidt en bruger bliver oprettet og gemt korrekt i vores HashMap. Den anden test ser på hvor vidt det er muligt at oprette flere brugere når man har nået bruger nummer 99. Begge tests lykkedes som forventet, og hvis man kigger i konsollen bliver der også skrevet en fejlmeddelelse når man prøver at oprette for mange brugere.



Her ses vores test af changePassword. Vi har lavet nogle restriktioner der gerne skulle sikre, at det nye password minimum er 6 tegn langt og indeholder minimum 1 lille tegn, 1 stort tegn og 1 tal. Her ses det, at begge tests lykkedes som forventet. Den første test prøver at ændre passwordet til noget der gerne burde gå igennem og den anden test prøver at ændre passwordet til "123456" hvilket ikke bør gå igennem. Her tester vi så på, om passwordet stadig er det gamle, hvilket det er.

Konklusion

Vi blev bedt om at udvikle et bruger-administrationsmodul, som skulle benyttes til håndteringen af brugere/operatører. Undervejs i udviklingen har vi i første fase fokuseret på at få et program, der først om fremmest virkede. Da vi havde et funktionsdygtigt program, begyndte vi at tilrette koden, så den var bedre delt op ved at mindske antallet af klasser, så systemet var mere overskueligt. Vi har selvfølgelig stødt på problemer undervejs, men ikke noget der har givet en større udfordring end at det kunne løses inden for kort tid. Som tidligere nævnt i rapporten, i implementations-afsnittet, er alle funktioner nævnt i opgaveoplægget blevet indført, dog er der visse funktioner der kunne have været lavet bedre, men dette er vi nødsaget til at lade ligge, grundet tidspres. Afsluttende kan vi konkludere, at opgaven ifølge opgaveoplægget er opfyldt med tilfredsstillende resultater. Vi har et funktionsdygtigt program, hvor alle funktioner mere eller mindre fungerer som de skal.

Bilag

Git

For at få fat i vores projekt, kan man importere det direkte til Eclipse vha. linket https://github.com/Chaaaaap/19_CDIO1 . Dette gøres ved at trykke File → Import → Git → Projects som Git → Clone URI → Indtaster linket i URI-feltet → Next → Finish. Når dette er gjort henter Eclipse selv hele projektet ned. Når først man har hentet projektet ned til Eclipse, skal man blot åbne klassen "main" og køre denne, for at starte spillet.
Man har dog også mulighed for at hente projektet ned som en .ZIP fil, der kan importeres i Eclipse som et eksisterende projekt. Dette gøres ved at trykke File → Import → General → Existing projects into workspace.

Systemkrav

Vores spil kan køres på alle maskiner der som minimum kører med Windows XP. Det eneste systemkrav der er, er at man skal have installeret JAVA 1.8, for at kunne bruge det library som vi har anvendt (jre 1.8.0_60). Man kan dog godt bruge en nyere udgave, så skal man blot ind og ændre i, hvilke libraries der er tilknyttet projektet. Det er ikke noget større problem, men hvis man ikke føler sig sikker på hvordan man tilføjer / fjerner libraries vil vi ikke anbefale det.