

Introduktion til programmering

Programmering og udvikling af små systemer samt databaser

HA.it 1. semester Eksamensprojekt

Case: ***Vi cykler til arbejde***


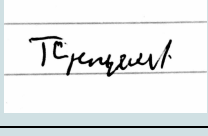
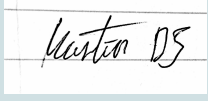
Underviser: Inge-Lise Salomon

Instructors: Michael Konnerup, Filip Andersen og Mathias Kahlen

Afleveres seneste 21 december 2017

Gruppenavn: XA12

Program: Kalorieforbrænding på cykel til arbejde

Studienummer	Navn	E-Mail	Tlf.	Underskrift*
119212	Malthe Wejnold Jørgensen	majo17ab@student.cbs.dk	+4528149305	
120372	Trym Ljungqvist	trlj17ab@student.cbs.dk	+4746946093	
120030	Martin Damgaard Jensen	maje17bi@student.cbs.dk	+4541279868	

*Med underskriften attesterer opgaveløserne at denne opgave er resultatet af en fælles indsats og at de almindelige regler for videnskabelig redelighed er overholdt. Herunder specielt, at materiale, der ikke udelukkende er lavet af opgaveløserne, er tydeligt angivet med kildehenvisninger.

Anslag og sider eksklusiv forside, indholdsfortegnelse, litteraturliste og bilag

Sider	34
Ord	9.339
Tegn (uden mellemrum)	46.043
Tegn (med mellemrum)	55.265
Afsnit	285
Linjer	921

Anslag og sider inklusiv forside, indholdsfortegnelse, litteraturliste og bilag

Sider	70
Ord	12.546
Tegn (uden mellemrum)	69.278
Tegn (med mellemrum)	87.632
Afsnit	1.201
Linjer	2.449

Ansvarsområder:

Rapportafsnit:	Malthe	Trym	Martin
1. Indledning	x		
1.1 Problemstilling	x		
1.2 Kravspecifikationer		x	
1.3 Metode			x
2. Proces			
2.1 Procesmodel	x		
2. 2 Procesplanlægning (Gantt-kort start)			x
3. Analyse & design		x	
3.1 Ide	x		
3.2 Objekt orienteret analyse			x
3.3 Analyse		x	
3.4 Design		x	
3.4.1 Use-cases	x		
3.4.2 Design klassediagram		x	
4. Hvad kan programmet			x
4.1 Hvad ville vi gerne have programmet til at kunne på forhånd	x		
4.2 Hvad kan programmet rent faktisk		x	
5. Brugervejledning			
5.1 Software krav			x

5.2 Hvordan bruger du programmet	x		
6. Implementering/ Programmering	x		
6.1 Main		x	
6.2 MainController			x
6.3 Bruger	x		
6.4 BrugerController		x	
6.5 Admin			x
6.6 AdminController	x		
6.7 CykelTur		x	
6.8 Data	x		
7. Test			
7.1 Formål og strategi for test	x		
7.2 Løbende test			x
7.3 Afsluttende test	x		
8. Konklusion			x
8.1 Gantt-kort (slut)	x		
8.2 Produkt		x	
8.3 Konklusion			x
9. Videre arbejde	x		
Klasser:			
Main		x	x
MainController	x	x	
Bruger	x	x	
BrugerController		x	x

Admin	x		x
AdminController	x		x
Data	x	x	
CykelTur		x	x

Indholdsfortegnelse

<i>1. Indledning</i>	8
1.1 Problemstilling	8
1.2 Kravspecifikationer til opgaven	9
1.3 Metode	9
<i>2. Proces</i>	11
2.1 Procesmodel	11
2.1 Procesplanlægning (Gantt-kort start)	12
<i>3. Analyse og design</i>	13
3.1 Idé	13
3.2 Objekt orienteret analyse	14
3.3 Analyse	14
3.4 Design	15
3.4.1 Use case	15
3.4.2 Design klassesdiagram	18
<i>4. Hvad kan programmet</i>	19
4.1 Hvad ville vi gerne have programmet til at kunne på forhånd	19
4.2 Hvad kan programmet rent faktisk	19
<i>5. Brugervejledning</i>	20
5.1 Software krav	20
5.2 Hvordan bruger du programmet	20
<i>6. Implementering/Programming</i>	23
6.1 Main	23
6.2 MainController	23
6.3 Bruger	25
6.4 BrugerController	26
6.5 Admin	29
6.6 AdminController	29
6.7 CykelTur	34
6.8 Data	34
<i>7. Test</i>	35
7.1 Formål og strategi for test	35

7.2 Løbende test	36
7.3 Afsluttende test	37
8. Konklusion	38
8.1 Gantt-kort (slut)	38
8.2 Produkt	39
8.3 Konklusion/perspektivering	40
9. Til videre arbejde	40
10. Litteraturliste	42
11. Bilag	44
11.1 Bilag A - Samarbejdskontrakt	44
11.2 Bilag B - Vandfaldsmodellen	45
11.3 Bilag C - Use case diagram	46
11.4 Bilag D - klassediagram	47
11.5 Bilag E - programkoden	48

1. Indledning

Vi cykler til arbejde er en dansk kampagne som har til formål at få flere danskere til at cykle til og fra arbejde, da det er nøglen til sundhed, renere miljø og mindre trængsel i trafikken.¹

Vi cykler til arbejde består allerede af 70.000-100.000 deltagere, men vi har som HA.it-studerende fået til opgave at lave en portal, som kan udvide brugeroplevelsen.

Vi har i XA12 besluttet os for at lave en kalorieberegner, da det linker sig til at få mere sundhed i Danmark. Kalorieforbrænding er en god måler på hvor meget sundhed cykling egentlig bidrager med, da det sætter tal på en utællelig størrelse, nemlig sundhed.

I dette program til beregning af kalorieforbrænding har man forskellige muligheder som afhænger af, om man er gæst, almindelig bruger eller admin bruger.

Programmet er kodet i JAVA ved hjælp af udvikler programmet IntelliJ IDEA.

1.1 Problemstilling

I denne rapport og tilhørende kode opretter vi et styresystem, der tillader Vi Cykler på Arbejde at lave en hurtig beregning af en deltagers kalorieforbrænding på én enkelt tur eller samlet for alle de cykelture brugeren måtte have.

For at kunne sætte sådan et program sammen kræver det en database, som indeholder oplysninger om en specifik bruger og den brugers tilhørende cykelture - samt giver mulighed for at en admin kan redigere i denne database ved at slette eller redigere oplysninger.

Da vi kun arbejder i RAM lageret vil alt det data som bliver indberettet i programmet blive slettet når det stoppes - dog har vi omtalt nogle udvidelser i afsnit 9. Til videre arbejde, som på sigt kunne være interessante at tilføje til programmet.

¹ <http://www.vcta.dk/OmVCTA>

1.2 Kravspecifikationer til opgaven

Kravene som stilles til opgaven er hovedsakelig funksjonene vi har lært som omhandler det å opprette, slette, redigere, vise alle tall og vise utvalgte tall. Vi ønsket av programmet vårt skulle gjøre det lett og logisk å navigere rundt i, noe vi har gjort ved og ikke lage for mange klasser. Vi har også brukt arv for å opprette en god oppbygging og struktur i vårt program.

Utover dette skal vårt program kunne skille mellom tre aktører: bruker, admin og gjest. Der det skal være muligt å lage en ny bruker, eller logge inn med en eksisterende bruker.

- Bruker skal ha adgang til sine egne og andre brukeres samlede tall, i tillegg til tallene fra sin siste sykkeltur. Brukeren skal også kunne nulstille sine egne tall, hvis brukeren ønsker å starte forfra med programmet.
- Administrator i programmet vil vi at skulle kunne de samme funksjoner som brukerne, i tillegg skal den kunne slette brukere, oprette nye brukere og redigere brukere. Når admin redigerer en bruker skal man kunne endre brukernavn, pinkode og nullstille en brukers tall.
- Gjesten i programmet har bare én funksjon og det er å se samlede kalorier brent for hver registrerte bruker. I tillegg skal det være muligt for en gjest å lage en ny bruker.

Når disse krav er oppfylt kan programmet vårt "vi cykle til arbeidet" anvendes til daglig bruk. Vi har et håp om at vårt program oppleves som brukervennlig og har de funksjoner som brukeren trenger. Vi ønsker videre at programmet er godt nok så folk i alle aldersgrupper ønsker og har en mulighet til å bruke det, spesielt studerende og barn som sykler til skolen.

1.3 Metode

I forbindelse med utviklingsarbeidet av systemet har vi arbeidet etter principperne i objektorientert programmering. Vi har søgt at programmere vores system således, at ansvarsområderne i de forskellige klasser er minimeret/begrænset. På denne måde kan vi genbruge dem i andre klasser.

Vi anvender klassernes attributter og metoder i andre klasser, ved at oprette op til flere forekomster af den pågældende klasse. Dette kaldes også at oprette et objekt af klassen, men vi anvender i vores projekt også nedarvning, da vi i vores system har flere klasser som arver fra dertilhørende superklasse. Dette vil dog blive gennemgået i større detalje under implementeringsafsnittene for de pågældende klasser.

En vigtigt del af objektorienteret programmering er endvidere brugen og formuleringen af modeller og diagrammer, som, på en overskuelig måde, viser hvordan de vigtigste dele af programmet skal se ud, når man begynder at kode². Dette skulle gerne være med til at gøre processen med at kode programmet lettere, da man så at sige har en skabelon for det som skal kodes. Man kan med andre ord bruge de forskellige diagrammer og modeller som en brugermanual for hvordan programmet skal samles.

Udarbejdelsen af modeller og diagrammer i dette projekt er lavet ud fra Unified Modeling Language (UML) principperne³. Der hører flere diagrammer til UML, og vi har i dette projekt arbejdet ud fra et par udvalgte, som i vores projekt gav mening.

Som det første har vi i vores projekt arbejdet med use case diagrammet. Dette diagram har vi valgt at bruge, da det giver et overordnet overblik over hvad programmet kan gøre. Vi kan i dette diagram, kort og enkelt beskrive de forskellige forbindelser i programmet overfor eksterne personer som ønsker at forstå programmet, eller som ikke har den store viden omkring programmering. Udover Usecase diagrammet har vi også vores klassediagrammer, som viser de forskellige klasser i vores program og relationerne mellem dem.

Vi har her gjort brug af to typer klassediagrammer nemlig design klassediagrammet og analyse klassediagrammet/domænemodellen, som tilsammen viser det overordnede billede af hvor mange klasser vi har, hvilke metoder og attributter klasserne har, og hvordan forholdene mellem klasserne er opbygget. Disse diagrammer er igen meget essentielle for kodningsfasen da de, ligesom use case diagrammet, hjælper os med at skabe et overordnet billede af hvordan kodningen af programmet skal gribes an.

Vi har i vores projekt, arbejdet ud fra vandfaldsmodellen. Dvs. at vi har brugt vandfaldsmodellen som fremgangsmåde for hvordan vi ønskede at vores projekt skulle skride fremad, og sat de relevante punkter fra vandfaldsmodellen ind i vores tidsplan.

² Litteraturliste 10.6

³ Litteraturliste 10.7

2. Proces

2.1 Procesmodel

Til udarbejdelse af denne opgave har vi som gruppe, som sagt, besluttet os for at bruge vandfaldsmodellen.

Valget er faldet på denne model, da vi har ønsket en stepvis fremgang i udarbejdelse af opgave og for at undgå forvirring af nye udviklere, som os selv, kan det være positivt at have en stram plan fra starten for at undgå løse ender i slutningen.

Vi som gruppe har gjort brug af interne deadlines, samt at kunne følge vores egen proces ved hjælp af et skema, som vi udarbejdede tidligt i rapportskrivningen⁴.

Vandfaldsmodellens grundlæggende principper går på, at man har nogle faser der skal gennemføres i en vis rækkefølge for at opnå det resultat man ønsker. Det første step skal gennemføres før der kan arbejdes på næste step og så videre⁵ - dette giver en noget langsommere proces, men giver os som gruppe mulighed for at udarbejde hvert step til det ambitionsniveau vi har sat for gruppen.

Vandfaldsmodellen har et lineært kendetegn modsat en procesmodel såsom SCRUM modellen, som er en mere agile model, hvor udviklingen af et program overlapper hinanden i både tids- og funktionalitet perspektiv. Denne model er blevet meget populær i nyere tid, da det giver mulighed for udviklere at udfordre hinandens ideer og kompetencer ved hjælp af prototyper - da vi ikke blot skal have et færdigt program, men også forhåbentlig skulle gøre hinanden og én selv som udviklere klogere og mere kompetente valgte vi vandfaldsmodellen. Vores kompetencer i forhold til udvikling af software er på samme niveau modsat dem som får mest udbytte af SCRUM modellen. Vi har derfor været nødt til at følges ad, da det let kan ske at nogen falder bagud.

Dog skal det siges at vi ikke nødvendigvis har været nødt til at ændre i de enkelte processer i vandfaldsmodellen løbende, når der har opstået udfordringer med programmet. Dette skete primært i implementeringen af koden og dette kan der læses mere om i afsnit 6.

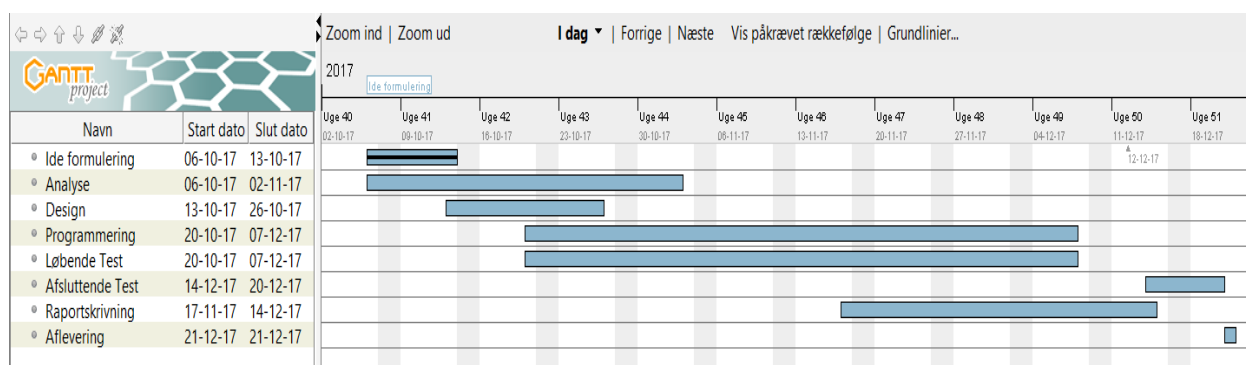
Implementering/Programmering og 7. Test.

⁴ Bilag B - Vandfaldsmodellen

⁵ Bilag B - Vandfaldsmodellen

Da vi modsat større organisationer ikke har nogen økonomiske udgifter kan det også give mening at bruge vandfaldsmodellen, da tid ikke koster penge for os. Vores eneste begrænsning er hvor meget tid vi har til vi skal aflevere.

2.1 Procesplanlægning (Gantt-kort start⁶)



Vi har i vores projektplan valgt at indlægge punkterne fra vores egen udarbejdede vandfaldsmodel, med de punkter og den fremgangsmåde vi havde forestillet os. Dette har vi valgt at gøre, da vi bruger vandfaldsmetoden som fremgangsmåden i vores projekt. Vi har ud fra disse punkter indlagt tidsintervaller for hvert punkt i forhold til, hvordan vi forestiller os at tidsforbruget kommer til at se ud.

Vores ide formulering har været forholdsvis kort, da vi i forvejen havde et projektoplæg som gav os rigeligt med gode ideer til hvordan vores projekt kunne se ud. Analysen er derimod lidt anerkendes. Vi har valgt at sætte analysen for, hvordan vi skal lave vores projekt til at være forholdsvis lang, da vi gennemløbende, mens vi udarbejder projektet, analyserer på om vi har truffet de rigtige beslutninger. Dette gør vi, da vi ønsker at kunne korrigere efterhånden som projektet skrider fremad, hvis vi på et senere tidspunkt skulle komme frem til, at vi har truffet en forkert beslutning. Vi mener derfor, at det vil være relevant at fortsætte analysen ind i programmeringsperioden, selvom designfasen reelt set er overstået. Af samme grund som ved analysen, har vi også valgt at sætte designfasen til at gå en smule ind i programmeringsfasen, så vi har planlagt for eventuelle udfordringer i vores implementering. Dette skulle gerne give os en buffer, hvis vi, når vi når til programmeringsfasen, opdager at vi har designfejl i vores designplan. Vi kigger naturligvis på det endelige resultat og udarbejder en projektplan for hvordan projektet reelt set gik, når projektet

⁶ Litteraturliste 10.11

nærmer sig enden, for at evaluere og se om det blev nødvendigt at korrigerer på vores planer og se om vi fulgte planen som vi forestillede os den i starten af projektet.

Selve programmeringsfasen er planlagt til at fylde størstedelen af projektet, og det er selvfølgelig her vi vil have vores hovedfokus, samtidig med at vi kontinuerligt vil teste systemet efterhånden som arbejdet skrider fremad. Vi planlægger også at starte rapportskrivningen, i forbindelse med at programmeringsfasen skrider mod enden, for at have arbejdet friskt i erindring når vi skal skrive rapporten. Med rapportskrivning menes udarbejdelse af rapporten og ikke dokumentaion, da det vil ske løbende i alle stadier.

Vi har endvidere i slutningen af projektet indlagt et par dage til at teste og finpudse systemet for, så vidt muligt, at holde en god kode skik, og sikre at systemet kører som planlagt inden aflevering.

3. Analyse og design

I denne fasen ser vi på hvilke krav vi stiller til vårt program. Her er det viktig at vi adresserer problemer vi kan møte og hva som eventuelt kan bli svært med å realisere vår ide. Vi har tenkt oss at vårt kaloriforburning program til sykkel skal være et program som viser hvor mye mange kalorier man forbrenner, på distansen man sykler. For å få dette til må man første lage en bruker, med brukernavn og passord. Her vil vi også ha mulighet til å kunne slette brukeren hvis man bestemmer seg for å slutte. Videre måtte vi finne ut av hvordan man regner ut hvor mye kalorier man bruker. Vi kikket på flere metoder for å få dette til, der mange utregninger ikke tok høyde for forskjell på menn og kvinner. Dette gir en meget unøyaktig beregning, vi valgte isteden å finne en mer nøyaktig metode som tok hensyn til både kvinner og menn. Vi brukte "En kvinde på 60-65 kg forbrænder 18 kcal pr. km. Tallet ligger på 20 kcal pr. cyklet km for en mand på 75-80 kg⁷." Derfra tok vi 18/60 for kvinner og 18/75 for å finne hvor mye du forbrenner pr. kg pr. km. Etter dette måtte vi gange dette med vekt og antal km, fordi brukerne har selvfølgelig forskjellig distanse til jobben. Derfor bruke denne formelen:

- For kvinner: $Vægt * 0,26 * \text{antal km}$
- For menn: $Vægt * 0,3 * \text{antal km}$

3.1 Idé

I denne fase kan metoden brainstorm bruges for at få alles idéer på bordet. Vi gennemgik diverse programmer vi kunne lave taget i betragtning af hvad vi kan og ikke kan. Vi valgte at lave et

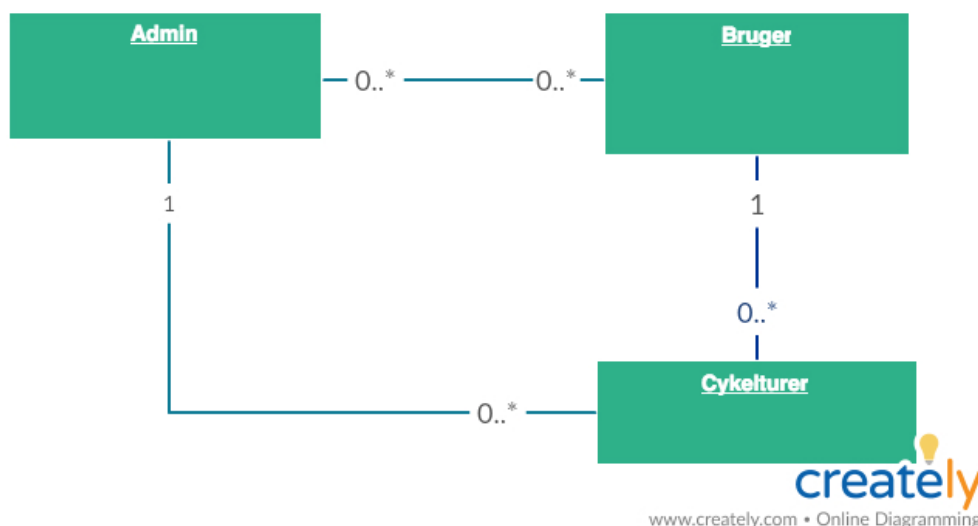
⁷ Litteraturliste 10.2

program som kan beregne en persons kalorieforbrænding på cykel. Dette er meget relevant i disse sundheds tider og det er samtidig noget som vi godt kan have med at gøre på vores niveau af programmering.

3.2 Objekt orienteret analyse

Objektorienteret analyse er brugt til at analysere og give en bedre forståelse og indsigt i hvad systemet kan, samtidig med at det giver os som gruppe mulighed for at opstille krav til systemet - efterfølgende designs hele systemet ved hjælp af objektorienteret design.

Vi har udarbejdet et analyse diagram:



3.3 Analyse

I analyse diagrammet vi har laget i 3.2 kan man se programmets konkrete objekter og deres direkte relasjoner mellom hverandre. Her kan man se at en admin kan ha 0 eller flere brugere. Selv om vi har tre prefabrikerede brukere, er det mulig å slette disse og dermed ikke ha noen brukere. Man kan også tilføye brukere, dermed kan antallet være høyere. Motsatt ser vi at en bruker også kan ha null eller flere admins. Dette er fordi man kan legge til flere admins (enten ved hard coding⁸

⁸ Litteraturliste 10.5

eller hvis en admin opretter en ny), men også slette admins, som betyr at vårt program faktisk kan ende opp med ingen administratorer. Videre har både admin og brukere mulighet til å ha 0 eller flere sykkelturner, men en sykkelturn er bare knyttet til en admin eller en bruker og har derfor tallet 1 til dem begge.

3.4 Design

Etter at vår analyse av systemet er ferdig kan man begynne å designe systemet. Vårt program har en objektorientert løsning, som betyr at vi har satt opp systemet ved å bygge opp systemet innenfra fra et IT-systemperspektiv. For å forklare dette har vi laget et design klassediagram som viser klassene samt attributtene og metodene og hvordan de interagerer med hverandre.

3.4.1 Use case

Programmets afgrænsning er beskrevet ved hjelp af use case-modeller, der skal beskrive hvad programmet kan, og hvilke aktører der kan hvad med systemet. Vi har valgt at have tre aktører nemlig gæst, bruger og admin.

Vi har oprettet tre prædefinerede brugere, som kan bruges til at teste programmet uden at skulle oprette en ny bruger hver gang.

Disse brugere kan logge ind på programmet. Nedenstående er en tabel med brugernes oplysninger:

Brugernavn	Pinkode	Køn	Bruger type
Inge-Lise	1111	kvinde	Admin
Michael	2222	mand	Almindelig
Filip	3333	mand	Almindelig

Mulighederne for de forskellige aktører har vi derefter opstillet i et use case skema, som beskriver hvilke funktioner programmet har og hvem der har adgang til disse funktioner.

Skemaet er opstillet således at de forskellige use cases er beskrevet i kolonne "Use case" og der er hertil et eventuelt kryds i kolonnen med aktører, hvis aktøren har adgang til use casen.

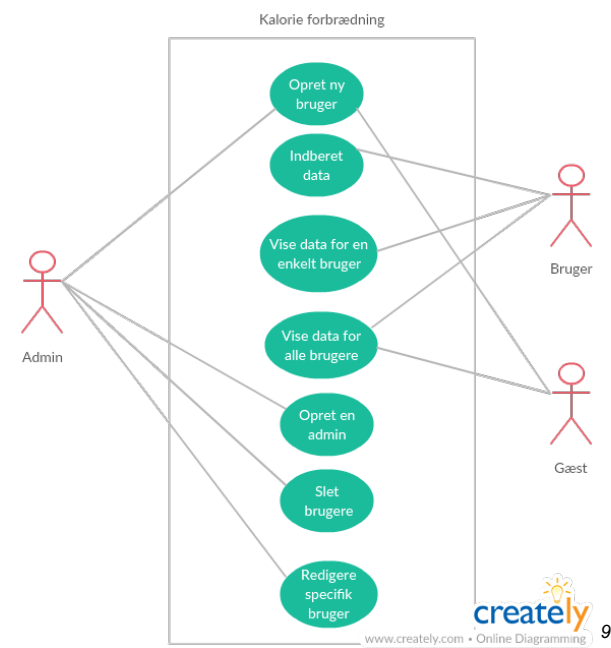
	Use case	Medlem	Admin	Gæst
Use case 1	Opret ny bruger		x	x
Use case 2	Indberet data	x	x	
Use case 3	Vise data for en enkelt bruger	x	x	
Use case 4	Vise data for alle brugere	x	x	x (begrænset)
Use case 5	Oprette en ny admin		x	
Use case 6	Slette bruger		x	
Use case 7	Redigere specifik bruger		x	
Use case 8	Nulstille personlig data (redigere)	x	x	

Vi har hertil opstillet et use case diagram, som er en visuel illustration af systemets elementer og funktioner, samt sammenhængen mellem disse og de forskellige aktører.

Use case diagrammet er sammensat med baggrund i ovenstående use case skema.

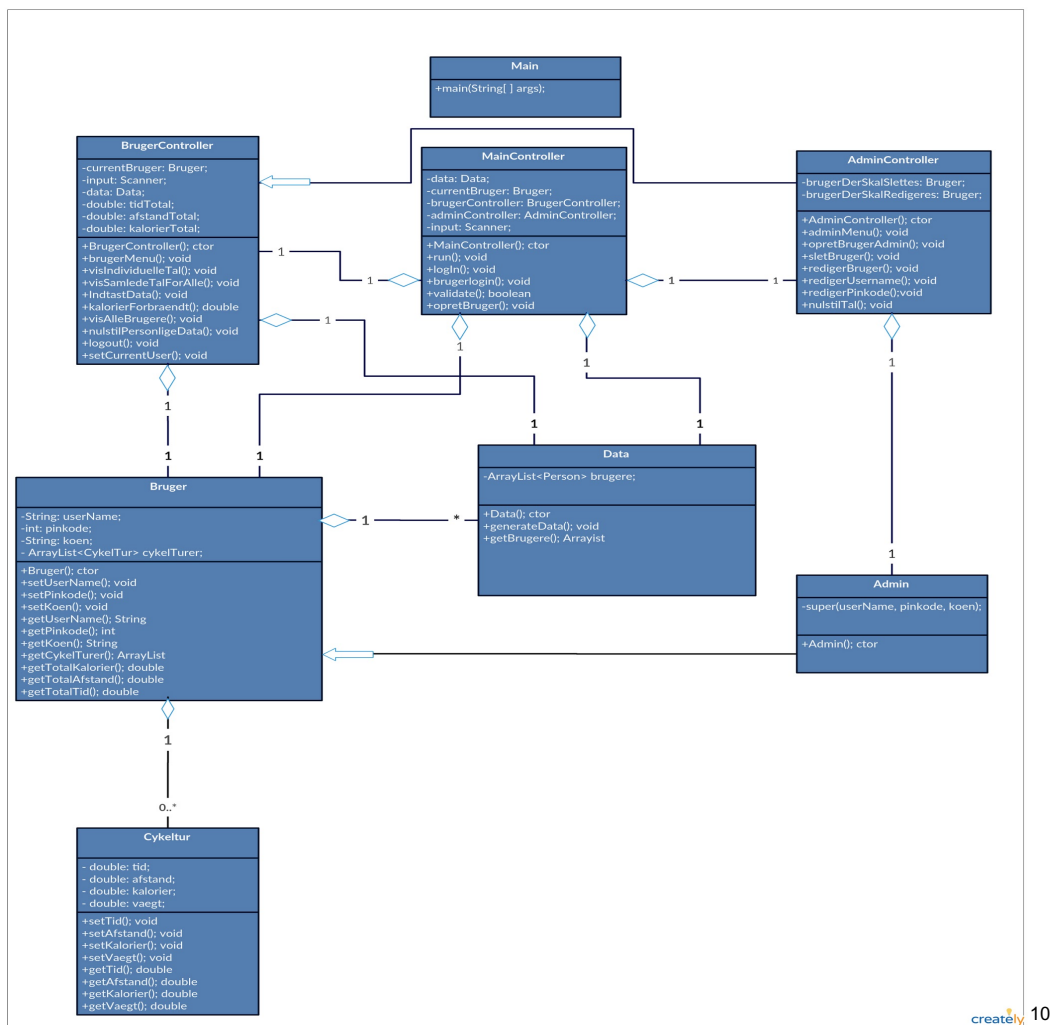
Da admin er nedarvet af bruger, så de streger som går fra bruger vil implicit også gå fra admin.

Nedarvning bliver forklaret uddybende i afsnit 6. Implementering/Programmering.



⁹ Bilag C - use case diagram

3.4.2 Design klassediagram



Dette design klassediagrammet viser alle våre klasser med deres tilhørende attributter og metoder, samt deres forhold til hverandre. Hele diagrammet viser strukturen i vårt program der klassene er listet øverst, deretter attributtene og nederst metodene. Diagrammet er statisk og viser hvilke klasser som samhandler med hverandre.

De forskjellige klasser interagerer forskjellige på hverandre. For eksempel ser man at "Admin" slekter på "Bruger", dette vises i programmet vårt ved at "Admin" har de samme attributter og metoder som "Bruger", mens en vanlig bruker ikke har samme rettigheter som en admin. Et annet eksempel er at man ser at en bruker kan ha null, eller flere cykelture. Mens en cykeltur bare kan tilhøre en bruker. Så ved en kikk på dette diagrammet kan enkelt få en oversikt over vårt program.

¹⁰ Bilag D - klassediagram

4. Hvad kan programmet

4.1 Hvad ville vi gerne have programmet til at kunne på forhånd

Vi som gruppe havde en forhåbning om at kunne lave et program som kunne få oplyst nogle informationer om en bestemt cykeltur og person og dertil beregne hvor mange kalorier personen forbrændte på den tilhørende cykeltur. Vi var udmærket klar over at kalorieforbrænding er meget svært at beregne, da det afhænger af ens aktivitetsniveau og ens forbrænding i hviletilstand. Vi ville rigtig gerne lave et program som også differentierede beregningen af kalorieforbrændingen om man var mand eller kvinde.

Det var ikke nemt at finde en formel for kalorieforbrænding for en person uden at vide præcis hvor meget de motionerer dagligt mm. Man kunne selvfølgelig lave et mere avanceret program som bad brugeren om mere information omkring dem, men da vi gerne vil have et program som er meget brugervenligt, også for mindre tekniske brugere, har vi valgt at gøre det så simpelt som muligt.

Desuden ville vi gerne have at programmet havde stor logik i forhold til administration af brugere dvs. oprette, slette, redigere og sammenligne brugere. Det meste af programmets logik og publiceringer ligger derfor i AdminController - se 6. Implementering/Programmering.

4.2 Hvad kan programmet rent faktisk

Som nevnt i avsnitt 2.1 har vi brukt waterfallmodellen¹¹ til vårt prosjekt, noe som har gjort at vi flere ganger har måttet gå tilbake et steg for å rette opp feil. Dette har gjort at programmet vårt ikke er helt som vi hadde forventet på forhånd. Vi har likevel fått til våre hovedpunkter som skrevet i 4.1. Vi fått til å lage en program der man kan logge inn, lage en ny profil og slette sin profil. Det kan også beregne kalorier ut i fra en sykkeltur og legge disse verdiene til en bestemt bruker. Dette gjør også at man kan legge sammen tallene fra sykkelturene og få totale tall fra alle sykkelturene overført til en bruker. I tillegg til dette kan man se hvem som har forbrønt mest og syklet lengst i både kilometer og minutter. Vår utregning av kalorier ble ikke så presis som vi kanskje hadde håpet på, men det fungerer greit ved at tallet er et realistisk tall uten at brukeren må skrive inn altfor mye informasjon om seg selv og sine treningsvaner. Med denne utregningen av kalorier kan man også se total kaloriforbrenning for hver enkelt av brukerne.

¹¹ Bilag B - vandfaldsmodellen

Vi er også veldig fornøyd med at vi både fikk til det å ha en admin som har mulighet til å både redigere og slette andre brukere. Adminen kan her redigere andres brukernavn, pinkode eller nullstille brukerens tall. Tillegg kan adminen også velge mellom det å lage en alminnelig bruker og en admin bruker. Dette var noe vi ikke hadde forventet at vi skulle klare å lage da vi skrev hva vi forventet at programmet skulle kunne(4.1). Man har også mulighet til å slette sin egen bruker. Faktisk så kan også adminen også slette seg selv, noe som kan gjøre at programmet kan stå uten admin - noe som selvfølgelig ikke skal skje. Den eneste redigeringen en alminnelig bruker kan er det å nullstille sine egne tall.

5. Brugervejledning

5.1 Software krav

For at kunne udføre programmet er der visse krav til hardware og software som er påkrævet. Det er påkrævet at du har en computer med et java development program som kan køre java kode. Vi anbefaler at der benyttes IntelliJ IDEA til dette formål, men andre programmer så som Eclipse kan også anvendes.

5.2 Hvordan bruger du programmet

Programmet starter ud med en login menu.



Her vælges ved at indtaste én af de fire valgmuligheder(0-3) efterfulgt af enter. Hvis man skal logge ind i første omgang vil du være nødsaget til at bruge en af de hardcoded brugere:

Brugernavn	Pinkode	Køn	Bruger type
Inge-Lise	1111	kvinde	Admin
Michael	2222	mand	Almindelig
Filip	3333	mand	Almindelig

Valg:

0

Programmet slutter.

3

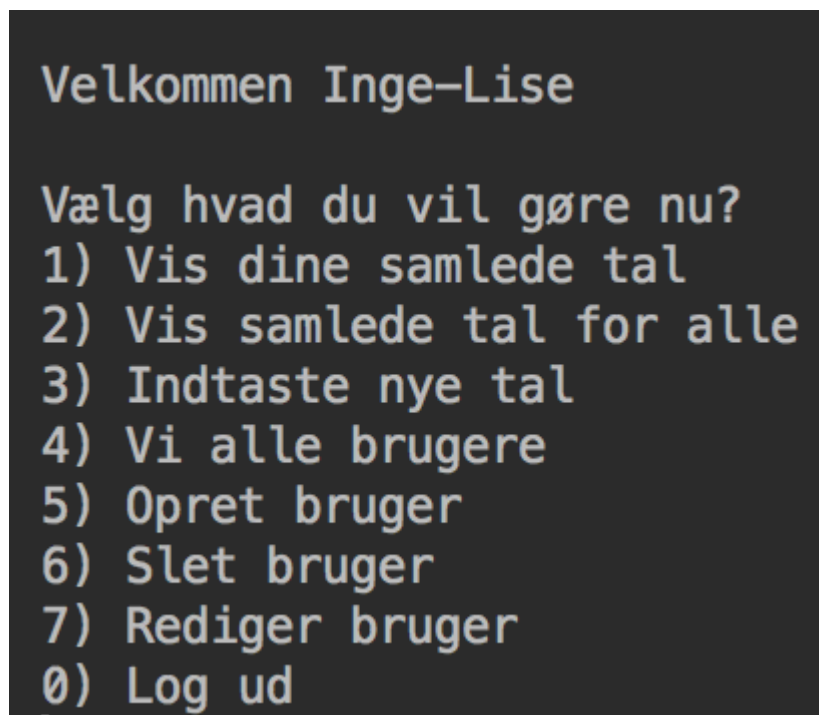
Du bliver præsenteret med alle brugere i programmet samt deres totalt kalorieforbrænding.

2

Du skal nu oprette en ny bruger. Det er ikke muligt at oprette en admin i denne funktion. Du vil blive bedt om at indtaste brugernavn, pinkode og køn. Der bliver tjekket om oplysningerne allerede findes og hvis de ikke gør bliver du oprettet i systemet - herefter vil du blive bedt om at logge ind.

1

Her vil du blive bedt om at indtaste et brugernavn og en pinkode - hvis disse ting passer med nogle af de brugere der er gemt i databasen vil man blive præsenteret med en menu, som for en alminde bruger kun indeholder mulighed 1,2,3, 4 og 0 mens at admin vil se således ud (Dog har almindelig bruger mulighed for at nulstille sine egne cykelture, mens admin skal vælge at redigere sig selv):



Her skal igen indtastes et tal mellem 0 og 7 efterfulgt af enter, så brugeren kan åbne op for funktionerne i programmet. Vælges 1, 2 eller 4 vil brugeren få vist noget statistik og menu vil blive præsenteret igen.

Indtastes 3 skal brugeren nu oplyse hvor langt, hvor hurtigt og ens vægt på seneste cykeltur. Dette vil blive gemt i databasen og kan senere præsenteres i mulighed 1,2 eller 4.

Gælder kun admin:

Indtastes 5 skal du som admin oplyse brugernavn og pinkode på den nye bruger du ønsker at oprette. Der bliver tjekket efter om brugernavnet allerede findes og hvis det gør kan det ønskede brugernavn ikke benyttes. I denne funktion er der som eneste sted mulighed for at oprette en til admin bruger, da du i denne funktion vil blive bedt om at vælge om den nye bruger skal være admin eller ej.

Indtastes 6 vil du blive præsenteret med en liste af alle brugere der er gemt i databasen og et tilhørende nummer. Herefter indtastes det tilhørende nummer udfra den bruger du ønsker at slette.

Indtastes 7 vil du igen blive præsenteret for en liste af alle brugere og et spørgsmål om hvem du vil redigere. Når du har valgt hvilken bruger du vil redigere har du mulighed for at ændre brugernavn, pinkode eller nulstille brugerens tal.

6. Implementering/Programming

6.1 Main

Main er vår første klasse og det er her vårt programmet begynner. Her har vi laget vår main, i tillegg til å opprette et objekt av vår MainController. Som sagt bruker vi arv til å strukturere vårt program og det er her det hele starter og holdes sammen. Vi har også laget en styling for å gi vårt program litt karakter. Denne stylingen er det første vårt program printer ut.

6.2 MainController

Maincontroller klassen i vores program er en vigtig klasse, da det er her brugere, administratorer og gæster bliver præsenteret for deres forskellige valgmuligheder. Man bliver i denne klasse spurgt om man ønsker at logge ind, oprette sig som bruger, se tallene for alle som er med i "vi cykler til arbejde", som er en gæste funktion, eller afbryde programmet. Disse muligheder bliver printet ved hjælp af en `System.out.println()`¹², som bruges til at fortælle brugeren hvad han eller hun nu skal foretage af valg. For at kunne undersøge hvilket valg brugeren foretager, har vi brugt en switch¹³, som alt efter hvilket valg brugeren foretager, kalder på en metode, som så fører brugeren videre.

Ønsker brugeren at logge ind, taster han eller hun 1 og kan derefter frit logge ind (forudsat at brugeren allerede er oprettet i systemet). Denne case i switchen kalder metoden `brugerLogin()`. Her bliver brugeren bedt om at indtaste brugernavn og pinkode. Passer det indtastede sammen med det som brugeren indtastede, da brugeren blev oprettet, logges brugeren ind. Skulle der dog være uoverensstemmelser i det som brugeren indtaster og det som brugeren indtastede ved oprettelse af brugeren, bliver man nægtet adgang. Dette skyldes at de indtastede værdier (brugernavn og pinkode) bliver taget med ned i vores `validate` metode. Denne metode er en Boolean, som tjekker efter om noget er sandt eller falskt. I vores system tjekker den som sagt efter om det brugernavn og den pinkode som bliver indtastet svarer til det brugernavn og den pinkode som blev indtastet da brugeren blev oprettet. Dette bliver undersøgt ved at sætte brugernavnene lig hinanden ved brug af `"equals"`¹⁴, og pinkoderne lig hinanden ved brug af `"=="`¹⁵. Der bruges i vores system `"equals"` og `"=="`, da vores brugernavn er String og pinkode er int, hvilket gør vi må bruge både `"equals"` og `"=="`, da `"equals"` ikke kan bruges på en int, og `"=="` ikke kan bruges på en

¹² Litteraturliste 10.1

¹³ Litteraturliste 10.1

¹⁴ Litteraturliste 10.3

¹⁵ Litteraturliste 10.1

String. Der bruges && for at sige til programmet det skal tjekke to parametre før det indenfor tuborgklammerne i if køres. Vi bruger en udvidelse af "equals", der hedder "equalsIgnoreCase", så programmet ignorere case sensitivity, men princippet er det samme.

Passer brugernavnene og pinkoderne så sammen, er vores Boolean true og man får lov til at fortsætte og brugeren der logger ind bliver sat til currentBruger. Passer de ikke sammen bliver man præsenteret for en fejlmeddelelse om at man har tastet forkert og må prøve igen.

Herunder ses koden for validate metoden.

```
112 public Boolean validate(String username, int pinkodeIndtast) {  
113     for (Bruger bruger : data.getBrugere()) {  
114         if (bruger.getUserName().equalsIgnoreCase(username) && bruger.getPinkode() == pinkodeIndtast) {  
115             currentBruger = bruger;  
116             System.out.println("\nVelkommen " + username + "\n");  
117             return true;  
118         }  
119     }  
120     return false;  
121 }
```

Når brugeren så er logget ind, går programmet tilbage til brugerLogin metoden og tjekker efter om den person som er logget ind er en almindelig bruger eller en administrator. Dette undersøges ved brug af "instanceof". Vi gør brug af denne, da vi benytter nedarvning i vores program, så programmet tjekker efter om den person som er logget ind er en "instanceof"¹⁶ admin. Instanceof tjekker om det objekt, der tjekkes er et objekt af det man beder efter eksempelvis admin.

Er dette sandt, kaldes adminMenu metoden som ligger i adminControlleren, og brugeren sendes videre til denne klasse, og får nye valgmuligheder. Er brugeren som er logget ind ikke admin, må brugeren være en almindelig bruger(da vi kun har 2 brugertyper), og derved kaldes brugerMenu metoden som ligger i personControlleren, og brugeren sendes så videre til denne klasse, og får nye valgmuligheder.

Brugerlogin metoden er endvidere indesluttet i en try-catch Exception¹⁷ for at fange eventuelle fejl input fra brugeren. Dette har vi valgt at tage med for at vise hvordan man ved hjælp af en try-catch kan fange input fejl, eks hvis brugeren vælger at indtaste en int, hvor man bliver bedt om en String, kan vi med en try-catch fange denne fejl og fortælle brugeren at der er fejl i input, og bede brugeren om at prøve igen. Ved at bruge en try-catch undgår vi også det uønskede scenarie, at programmet stopper hvis der skulle blive indtastet en forkert datatype. Vi har dog kun valgt at anvende try-catch på enkelte dele af programmet for at vise hvordan disse fungerer, eftersom vi i opgaven måtte gå ud fra at vores brugere opfører sig pænt.

¹⁶ Litteraturliste 10.3

¹⁷ Litteraturliste 10.1

Nedenfor er vores Exception vist:

```
72 public void brugerLogin() {  
73  
74  
75     try {  
76         System.out.println("Indtast Brugernavn");  
77         String username = input.next();  
78         System.out.println("Indtast Pinkode");  
79         int pinkodeIndtast = input.nextInt();  
80  
81         if (validate(username, pinkodeIndtast)) {  
82             brugerController.setCurrentUser(currentBruger);  
83             adminController.setCurrentUser(currentBruger);  
84  
85             // Her tjekkes om brugeren er admin. Hvis brugeren er admin bliver brugeren præsenteret med en avanceret menu  
86             if (currentBruger instanceof Admin){  
87  
88                 adminController.adminMenu();  
89  
90  
91             } else {  
92  
93                 brugerController.brugerMenu();  
94             }  
95  
96         } else {  
97             System.out.println("Forkert brugernavn eller pinkode! Prøv igen:");  
98         }  
99  
100     } catch (InputMismatchException e) {  
101  
102         System.out.println("Du skal taste et heltal i pinkode");  
103         input.next();  
104  
105     }
```

Den anden mulighed i vores mainController er at man kan oprette sig som bruger, hvilket man gør ved at taste 2, når man bliver præsenteret for login metoden. Gør man dette kaldes opretBruger metoden. I denne metode beder programmet brugeren om at indtaste et ønsket brugernavn. Igen har vi her gjort brug af en Boolean til at tjekke efter om det indtastede brugernavn er identisk med et brugernavn som allerede er i brug. Er dette tilfældet bliver brugeren bedt om at vælge et nyt brugernavn. Vi har valgt at gøre dette da brugernavne bør være unikke, så vi kan arbejde med data for den enkelte bruger, og derfor bør hvert brugernavn kun kunne forekomme 1 gang. Derefter bliver man bedt om at oprette sin nye pinkode, og indtaste den 2 gange, så vi kan tjekke efter at man har tastet rigtigt og kan huske sin kode, derefter bliver man bedt om at indtaste sit køn, hvilket er nødvendigt, da kalorieberegneren beregner afhængig af køn. Til sidst tilføjes den nye bruger så til ArrayListen¹⁸ med personer, og personen får tildelt en cykeltur.

6.3 Bruger

Denne klasse specificere vores Bruger. En bruger består af et brugernavn(username), pinkode, et køn samt en liste af cykelture. Bruger klassen er en superklasse og nedarver klassen Admin.

¹⁸ Litteraturliste 10.1

Denne klasse har ikke noget funktionalitet da det blot bruges som et objekt i vores kode. Vores Bruger klasse er fundamentet for næsten alt andet logik i vores system, da hele systemet bygger på at have en liste af brugere som har en tilhørende cykeltur.

Vi har getter og setter metoder for diverse attributter i klassen - getter og setter bruges til at få fat i en brugers oplysninger i en anden klasse (getter) eller ændre en attribut i anden klasse (setter)¹⁹.

Vi har nogle enkelte metoder der ser således ud i klassen:

```
43     public double getTotalKalorier() {  
44         double totalKalorier = 0;  
45         for(CykelTur cykelTur: cykelTure) {  
46             totalKalorier += cykelTur.getKalorier();  
47         }  
48         return totalKalorier;  
49     }
```

Disse get metoder bruges til at sammenlægge kalorier for alle ture for den enkelte bruger, da vi er interesserede i at kunne vise hvilken bruger der har forbrændt mest mm. Det er tilsvarende metode for afstand og tid.

6.4 BrugerController

Vår klasse BrugerController er en superklasse som inneholder alt det en bruger kan gøre i vårt program. Klassen nedarver også til adminController, der vår admin kan gøre det samme som en bruger, bare mer som blir forklart mer i klassen AdminController. Vi starter med å lage en Scanner, samt å definere våre attributter. Disse attributtene er protected som vil si at alle klassene i samme package har tilgang til dem. Dette er en viktig del av vårt program der vi har brukt arv til å strukturere det hele.

```
Data data;  
Scanner input = new Scanner(System.in);  
protected Bruger currentBruger;  
protected double tidTotal = 0;  
protected double afstandTotal = 0;  
protected double kalorierTotal = 0;
```

¹⁹ Litteraturliste 10.9

Videre kommer vi til vår brugermenu som alle almindelige brugere kommer til, når de har logget sig inn med sitt brugernavn og passord.

```
protected void brugerMenu() {  
    //Almindelig brugers menu  
  
    int i;  
    do {  
        try {  
            System.out.println("Vælg hvad du vil gøre nu?\n1) Vis dine samlede tal" +  
                               "\n2) Vis samlede tal for alle\n3) Indtaste nye tal\n4) Vi alle brugere \n5) Nulstil dine tal \n0) Log ud");  
  
            i = input.nextInt();  
  
            switch (i) {  
                case 1:  
                    visIndividuelleTal();  
                    break;  
                case 2:  
                    visSamledeTalForAlle();  
                    break;  
                case 3:  
                    indtastData();  
                    break;  
                case 4:  
                    visAlleBrugere();  
                    break;  
                case 5:  
                    nulstilPersonligeData();  
                    break;  
                case 0:  
                    logout();  
                    break;  
                default:  
                    System.out.println("Du har tastet forkert. Prøv igen!");  
            }  
        } catch (Exception e) {  
            System.out.println("Du har tastet noget forkert. Prøv igen");  
        }  
    } while (currentBruger != null);  
}
```

Som vi ser i skjermbildet over har vi brukt en do-løkke²⁰ til å lage vår menu, der vi bruker en switch valg funksjon til å definere hva som skjer når man trykker på det forskjellige case nummerene. I denne menuen har vi også lagt inn en try/catch funksjon. Først definerer vi vår try-blok, tryen gjør at vårt program ikke avbrytes hvis den oppdager en exception, den blir heller fanget opp af vår catch som forteller hva programmet skal gjøre hvis man ikke indtaste et af valgalternativene. Vi er stadig i loopen, noe som betyr at hvis en exception blir fanget opp af catchen vil det hele kjøre på nytt og man vil få en ny sjanse til å taste riktig tal i menuen.

- Case 1: Blir man sendt videre til en ny menu som viser brugerens tal fra siste tur i tillegg til de samlede tallene fra alle turene. Her bruker vi "getters" til å hente ut de spesifikke opplysninger fra arraylisten. Som man ser i skjermbildet ser man at funksjonen fungerer ved at den skiller på sist tur og sammenlagte turer.

²⁰ Litteraturliste 10.1

```
Alle dine ture lagt sammen  
Du har forbrændt 561,6 kalorier  
Du har kørt 24,0 km.  
Du har brugt 89,0 minutter  
Du vejer 90,0 kg.  
  
På din sidste tur  
Forbrændte du: 93,6 kalorier  
Kørte du: 4,0 kilometer  
Kørte du: 20,0 minutter  
Vejede du: 90,0 kg.
```

- Case 2: Taster du 2 vises samlede tal for alle brugerne og hvem som har forbrændt flest kalorier, hvem som har syklet lengst og hvem som har syklet i flest minutter. For å hente brukernes totale tall bruker vi get-metode til å hente brugerne fra Bruger-klassen, deretter henter vi alle brukernes sykkelturer for å kunne legge attributterne sammen. For å finne ut hvem der leder i de forskjellige kategorier setter vi bare den med størst totale verdier til å være vinner i de forskjellige kategorier. Her måtte vi sætte en "startverdi, så hvis ingen taster inn noen verdier

```
Bruger vinderKalorier = data.getBrugere().get(0);  
for (Bruger bruger : data.getBrugere()) {  
    if (vinderKalorier.getTotalKalorier() < bruger.getTotalKalorier()) {  
        vinderKalorier = bruger;  
    }  
}
```

- Case 3: Taster du 3 kan du indtaste nye tal til på din bruger. Det vil si at hver gang du sykler til arbeidet må man indtaste:
 - Hvor langt man syklet(km)
 - Hvor meget man vejer(kg)
 - Hvor lang tid man syklet(min)

Dette kommer frem ved en do-løkke, der vi også har lagt inn en try/catch i menyen for å forhindre at programmet krasjer. Videre skal disse verdiene og sykkelturen lagres på den aktuelle personen. Man kan senere se hvor langt og hvor lenge man har syklet, både på din siste tur og totalt. Dette gjør vi ved å legge til den nye sykkelturen til brukeren som syklet turen. Etter man har tastet inn dataen fra sykkelturen har vi laget metoden "kalorierForbraendt" som regner ut hvor meget man har forbrændt på sin sykkeltur. Her bruker vi vekt og km syklet samt enten 0,26 eller 0,3. Dette spørres om du er kvinne eller mann(fotnote), der vi har brukt en "if" til å skille disse.

- Case 4: Videre kan man velge å vise alle brukere. Her bruker vi get-metoder til å hente brukernavn og deres totale kaloriforbrenning. Vi har laget et stort mellomrom mellom disse to verdiene i vår System.out.print for å få det oppstilt på en som gir våre brugere en god oversikt.

- Case 5: Her vi gitt våre brukere en sjanse til å redigere sin bruker ved å nullstille sine egne tall. Dette gjøres ved å bruke den innebygde metoden "removeAll" som fjerner alle tillagte tall. Deretter resettes all data til den aktuelle brukeren ved å definere dem til 0 og legge dem til som nye verdier i tallene til den enkelte bruker. Denne funksjonen finnes for at man skal kunne starte dette kalori-programmet på nytt, noe vi kan tenke oss kan være en nyttig funksjon for våre brugere.

```
currentBruger.getCykelTure().removeAll(currentBruger.getCykelTure());  
System.out.println("Du har nullstillet dine tal\n");  
CykelTur nyC = new CykelTur( afstand: 0,  væegt: 0,  tid: 0,  kalorier: 0);  
currentBruger.getCykelTure().add(nyC);
```

- Case 0: I den siste casen i denne menyen skal man kunne logge ut av sin profil. Dette gjøres ved å skrive currentBruger=null, grunnen til dette er at så lenge currentBruger har en verdi så kjører programmet. Settes denne til null "hopper" den ut til den forrige meny der brukeren heller ikke har en verdi ved å bruke en do-løkke.

6.5 Admin

Vores admin klasse er en meget lille klasse, som vi dog er afhængige af, da vi arbejder med nedarvning i vores program. Selve klassen admin nedarver fra klassen person, da attributterne som gælder for en person også gælder for en admin. For at kunne tjekke efter om brugeren er "instanceof" admin eller om brugeren er en almindelig bruger er klassen nødvendig, men udover dette bruges den kun til at instantiere administratoren gennem en constructor.

6.6 AdminController

AdminController er den udvidet version af menuen, som den almindelige bruger har adgang til. AdminControlleren indeholder enormt meget logik der kan ændre ved listen af brugere som er i programmet.

AdminController er en nedarvet klasse af BrugerController²¹. Det at en klasse er nedarvet vil sige at der er adgang til de samme metoder og attributter, som i superklassen (BrugerController). Da vores admin ikke er forskellige på nogle parametre udover det faktum at være admin - dette bliver tjekket ved login er forskellen ved admin og almindelig bruger vist i AdminController.

²¹ Litteraturliste 10.4

Hvis brugeren er et objekt af Admin bliver der præsenteret med denne klasse, mere specifikt metoden adminMenu() i stedet for BrugerController/brugerMenu.

Brugeren bliver sat overfor nogle valg ved hjælp af System.out.print - som bruges til at kommunikere med brugeren. System.out.print bliver vist i konsollen og viser alle de steder vi som programmører har besluttet at brugeren skal tage stilling til noget, foretage en handling eller blive præsenteret med noget statistik.

Når vi præsenterer brugeren med et valg er det oftest for at vælge mellem diverse muligheder ved hjælp af indtastning på tastaturet. Dette gøres ved hjælp af en switch-valgfunktion²² - modsat en if-else²³ funktion giver switch mulighed for at man vælger mellem mange valgmuligheder som bruger. En switch fungere således at man opgiver en variable som switch reagere på. Dertil er der opstillet diverse 'cases', som siger noget om hvad der skal ske efter brugeren indtaster den gældende variable.

Et eksempel på switch i vores program kunne være (taget fra AdminController):

```
37         valg = input.nextInt();
38
39         switch (valg) {
40             case 1:
41                 visIndividuelleTal();
42                 break;
43             case 2:
44                 visSamledeTalForAlle();
45                 break;
46             case 3:
47                 indtastData();
48                 break;
49             case 4:
50                 visAlleBrugere();
51                 break;
52             case 5:
53                 opretBrugerAdmin();
54                 break;
55             case 6:
56                 sletBruger();
57                 break;
58             case 7:
59                 redigerBruger();
60                 break;
61             case 0:
62                 logout();
63                 break;
64             default:
65                 System.out.println("Du har valgt forkert. Prøv igen");
66
67         }
```

²² Litteraturliste 10.1

²³ Litteraturliste 10.1

Som det kan ses på skærbilledet så skal indtaste en Integer - et heltal, som bliver sat ind i variablen 'valg'. Vores switch kigger så efter hvad brugeren har indtastet for at finde ud hvilken case programmet skal fortsætte i.

Indtaster brugeren noget som ikke er indenfor 0-7 vil 'default' bliver kørt, som blot fortæller brugeren de har tastet forkert. Da vores switch er i en løkke vil brugeren blive bedt om at forsøge at indtaste noget indenfor intervallet 0-7.

Skulle brugeren finde på at indtastet noget andet end en Integer vil programmet melde en fejl. For at undgå dette kan man bruge en try-catch, hvilket bliver beskrevet i 6.4 BrugerController og MainController.

Case 1,2,3 4 og 0 er de samme som i BrugerControlleren og bliver forklaret i det afsnit.

I denne switch i case 5,6 og 7 har admin nogle funktioner som en almindelig bruger ikke har. Vælger brugeren at indtaste en af disse cases bliver kode linjen mellem casen og breaket kørt. I dette eksempel er det tre forskellige metoder der kan blive kørt.

opretBrugerAdmin er blevet berørt i vores use case afsnit, men rent teknisk er det egentlig den samme syntaks som i vores metode opretBruger i MainController. Den eneste forskel er at i denne metode bliver brugere spurgt om den nye bruger skal være admin eller alminde. Desuden vil nogle af de System.out.print, som brugeren bliver præsenteret med, være forskellige fra metoden opretBruger i MainController.

```
130         do {
131             switch (typeAfBruger) {
132                 case 1:
133                     System.out.printf("Du har nu oprettet %s som bruger\n", nytUserName);
134                     Bruger bruger = new Bruger(nytUserName, nyPinkode, nytKoen);
135                     data.getBrugere().add(bruger);
136                     CykelTur cykelTur = new CykelTur( afstand: 0,  vaegt: 0,  tid: 0,  kalorier: 0);
137                     bruger.getCykelTure().add(cykelTur);
138                     break;
139                 case 2:
140                     System.out.printf("Du har nu oprettet %s som admin", nytUserName);
141                     Admin admins = new Admin(nytUserName, nyPinkode, nytKoen);
142                     data.getBrugere().add(admins);
143                     CykelTur cykelTurAdmin = new CykelTur( afstand: 0,  vaegt: 0,  tid: 0,  kalorier: 0);
144                     admins.getCykelTure().add(cykelTurAdmin);
145                     break;
146                 default:
147                     System.out.println("Vælg venligst 1 eller 2");
148             }
149         } while (typeAfBruger != 1 && typeAfBruger != 2);
150     }
```

Vælger brugeren at oprette bruger som almindelig bruger bliver den nye bruger lagret i vores arrayList af brugere, samt tildeler den nye bruger en CykelTur der er nulstillet - linje 136-142.

Vælger man at oprette en ny admin sker det samme med undtagelse at den nye bruger bliver lagret, som et objekt af Admin dvs. når brugeren bliver tjekket ved login om de er InstanceOf Admin bliver de præsenteret med admin menuen frem for den almindelige menu - linje 148-154. Grunden til at dette kan lade sig gøre at Admin er nedarvet af Bruger, så i realiteten er det samme objekt blot med to forskellige navne og adgang til forskellige funktioner. Det der virkelig separeres i vores arv i programmet er BrugerController og AdminController.

sletBruger(); har det simple formål at itere igennem alle brugere og printe deres navn. Herefter skal admin vælge hvilken bruger der skal slettes. Når brugeren bliver valgt så bliver den valgte bruger slettet ved hjælp af:

```
191         brugerDerSkalSlettes = this.data.getBrugere().remove( index: indexAfBrugerDerSkalSlettes-1);
```

Grunden til vi siger indexAfBrugerDerSkalSlettes-1 er at når vi printer menuen for hvilken bruger der skal slettes så siger vi +1, så listen ikke starter i 0 - men da index i en arrayList starter i nul²⁴ og brugeren taster i princippet et tal for højt sætter vi det ned med én.

²⁴ Litteraturliste 10.8

Når en bruger bliver slettet vil deres data stadig blive brugt til at finde de samlede tal i VisSamledeTalForAlle i BrugerController, men dette er bevist, da brugerens indtastede data stadig er relevant for den samlede statistik.

redigerBruger(); gør ligesom sletBruger();, nemlig at printe alle brugere og man skal hertil vælge hvilken bruger der skal redigeres. Når dette er valgt bliver den bruger sat til brugerDerSkalRedigeres. Hertil har man tre muligheder for den bruger nemlig at ændre brugernavn, pinkode og nulstille brugeren.

Skal brugernavnet redigeres tjekker programmet om det nye indtastede brugernavn allerede findes i systemet.

```
265         if (bruger.getUserName().equals(nytUserName)) {  
266             eksisterendeBruger = true;  
267             System.out.println("Brugernavnet findes allerede. Prøv igen");  
268         }  
269     }  
270 }  
271 brugerDerSkalRedigeres.setUserName(nytUserName);
```

Gør det ikke det bliver det nye indtastede bruger sat til den valgte brugers nye brugernavn.

Princippet er det samme i ændring af pinkode, dog med den undtagelse at admin skal indtaste den nye pinkode to gange blot for at tjekke at der ikke blev tastet noget uønsket første gang.

Vælger admin at nulstille en brugers cykelture, så bliver alle cykelture fjernet fra brugeren (linje 329). I og med vi bruger den indbyggede funktion RemoveAll²⁵ bliver cykelturene ikke blot sat til 0, men de bliver fjernet helt. Derfor er vi nødt til at oprette en ny cykeltur til brugerDerSkalRedigeres hvor alle attributter er sat til 0 (linje 331) - herefter tilføjes den nulstillede cykeltur til arrayListen af cykelture til brugeren (linje 333). Se nedenfor:

```
329     brugerDerSkalRedigeres.getCykelTure().removeAll(brugerDerSkalRedigeres.getCykelTure());  
330  
331     CykelTur c = new CykelTur( afstand: 0,  vaegt: 0,  tid: 0,  kalorier: 0);  
332  
333     brugerDerSkalRedigeres.getCykelTure().add(c);
```

Ligesom i BrugerController bliver programmet afsluttet og sat tilbage til login menu, hvis der indtastet 0.

²⁵ Litteraturliste 10.3

6.7 CykelTur

Klassen CykelTur inneholder de forskellige dataene som en bruger får ved å sykle til arbeidet og intaste i programmet. Disse attributter er tid, vekt, avstand og kalorier. Denne klassen går igjennom hele programmet og er svært viktig ved at selve programmet er bygget nemlig for sykkelturer til arbeidet. Det brukes også get-metoder for å hente de forskjellige verdiene til attributtene nevnt tidligere for å danne en sykkeltur.

6.8 Data

Vores data klasse består primært af at hardcoded nogle prædefinerede brugere og deres tilhørende cykelture. Disse prædefinerede brugere er blevet berørt tidligere i rapporten.

Vores data klasse består desuden af vores ArrayList af brugere. En ArrayList er en samlet række af data objekter. I dette tilfælde består vores ArrayList af Bruger objekter. Dvs vi kan lagre vores personer i en liste af brugere. Det smarte ved en ArrayList fremfor en almindelig Array er at man kan have flere forskellige datatyper i den samme liste, modsat en Array skal det defineres hvilken datatype der bliver gemt i Array'et - dette kunne være en række af tal (ex. Integer eller double) eller ord (ex. String)²⁶

En ArrayList bliver oprettet på følgende måde:

```
13      ArrayList<Bruger> brugere = new ArrayList<>();
```

Dog er det vigtigt at importere et java udvidelses program der indeholder ArrayList. Dette gøres ved at skrive import.java.util.ArrayList; før påbegyndelse af klassen hvor ArrayListen skal oprettes.

Vi opretter vores hardcoded brugere på følgende måde:

```
24      Admin admin1 = new Admin( userName: "Inge-Lise", pinkode: 1111, koen: "kvinde");  
25      Bruger bruger1 = new Bruger( userName: "Michael", pinkode: 2222, koen: "mand");  
26      Bruger bruger2 = new Bruger( userName: "Filip", pinkode: 3333, koen: "mand");
```

Samme princip som når vi opretter en ny ArrayList, dog skal der ikke importeres noget udvidelsesprogram for at gøre dette, dog kan det lade sig gøre fordi vi har to klasser der hedder Bruger og Admin. Når vi opretter en ny hardcoded bruger vil programmet melde fejl, hvis der ikke indsættes noget i de attributter som klassen har. IntelliJ er så smart at det oplyser hvilken attribut man sætter en værdi på - som det kan se på ovenstående screenshot.

²⁶ Litteraturliste 10.1

Da et Admin objekt er nedarvet fra Bruger, så er det muligt at lagre dem i samme liste da objektet i realiteten er det samme - i hvert fald i vores system. Vi tilføjer vores hardcoded brugere til vores liste af Brugere på følgende måde:

```
28 brugere.add(admin1);  
29 brugere.add(bruger1);  
30 brugere.add(bruger2);
```

Denne måde at tilføje objekter til ArrayListen bliver på samme måde brugt når vi tillader brugeren at oprette sig som ny eller det bliver gjort i AdminController.

Brugere har som sagt en tilhørende ArrayList af cykelture, som bliver oprettet i Bruger klassen og er blevet berørt i afsnittet omhandlende Bruger klassen.

Når vi opretter en ny CykelTur bliver det gjort på samme måde som Bruger. Nemlig ved at skrive: `CykelTur c1 = new CykelTur(afstand, vaegt, tid, kalorier);` Vi har i de hardcoded cykelture valgt at afstand, vaegt, tid og kalorier er sat til nul - da vi ønsker at brugeren selv indtaster dette.

Navnet c1 er blot et variable navn og af programmøren kaldes hvad man ønsker. Vi har bare kaldt det c1, c2 osv for at gøre det simpelt.

Det sidste som bliver gjort i Data klassen er at lagre den nye ArrayList af cykelture til en specifik Bruger eller Admin. Dette gøres så man kan finde én specifik brugers ture eller endnu mere præcist en specifik tur brugeren har haft

```
38 admin1.getCykelTure().add(c1);  
39 person1.getCykelTure().add(c2);  
40 person2.getCykelTure().add(c3);
```

Ved de hardcoded brugere får de blot tilføjet de tre 'tomme' cykelture som vi netop har oprettet. Disse ture er ikke som sådan tomme attributterne i cykelturen er bare sat lig nul.

7. Test

7.1 Formål og strategi for test

Test er meget brugbart når man udvikler et program, da det hjælper udviklerne finde fejl og mangler ved programmet. Der er mange forskellige måder at teste ens program.

Som udvikler er man nødt til at foretage løbende test af programmet, da vi, specielt som nye udviklere, oplever mange fejl i både syntaks og opbygning af programmet. Disse løbende test er essentielle for programmets udvikling, da de kan bygge sig op og blive meget krævende at håndtere længere henne i processen, hvis de ikke bearbejdes så snart de opdages. Disse test kan også være med til at finde små fejl i vores printninger i konsollen, som brugeren bliver præsenteret for.

Til sidst er det nødvendigt at lave afsluttende test, som kan variere meget i forhold til hvad man gerne vil finde af fejl. Vores program kan godt stoppes af brugeren ved at indtaste noget andet end der bliver bedt om, da vi kun har benyttet Try-Catches få steder i programmet da vi som udviklere forventer at brugeren opføre sig ordentligt og det samtidig ikke er et krav at have Try-Catches i alle valgmuligheder i programmet.

Da vores program ikke er større end det er, så har vi ikke været nødt til kun at test enkelte elementer af programmet, men det har været muligt at teste alle funktioner og deres output i programmet. Dog skal der skelnes mellem fejl som stopper programmet uden det er meningen eller blot fejl som er påtvunget af brugeren.

Vi har forsøgt at udrydde alle former for fejl der stopper programmet - hvordan vi har gjort dette bliver beskrevet i de to følgende afsnit.

7.2 Løbende test

I løbet af projektperioden er vi kontinuerligt stødt på fejl. I takt med at vi langsomt er blevet bedre og bedre til programmering er det dog lykkedes os at finde fejlene hurtigere og løse dem effektivt.

Vi har løbende testet programmet, og det blev hurtigt klart for os, at den bedste måde løbende at teste om programmet virker og gør som vi vil have det til, er ved at printe mellemresultater som vi kalder det. Dermed mener vi at vi undervejs har gjort stort brug af de forskellige udskrivnings metoder eks `System.out.print()` for at se om det vi har programmeret også gør det vi vil have det til at gøre. Eksempelvis har vi i arbejdet med vores 2 `ArrayList`er gjort meget brug af denne funktion for at se om de brugere vi tilføjede til vores personer `Arrayliste` nu også blev tilføjet korrekt. Denne metode til løbende at teste vores program, har vist sig, i hvert fald for os, at være den bedste måde løbende at teste programmet, hvilket er grundlaget for at vi har brugt den så meget som vi har.

Et andet eksempel på et problem som gik os på var, at vi programmet crashede når vi indtastede en forkert datatype i forhold til det som eks. en pinkode var defineret som. Da vi brugte en Sting som brugernavn, hvilket også godtager tal, men en int som pinkode, hvilket ikke godtager bogstaver, fik vi flere gange programmet til at gå ned. For at komme udenom denne fejl, valgte vi at oprette en Exception enkelte steder i koden for at komme udenom denne fejl (Se afsnit 6.2 MainController). Dette gav brugeren mulighed for at indtaste noget forkert uden at programmet gik ned. Brugeren ville blot modtage en fejlmeddelelse og blive bedt om at prøve igen. Vi kunne godt have anvendt Exceptions alle steder i programmet med brugerinput for at gøre koden skudsikker for datatype fejl, men grundet anden prioritering valgte vi kun at gøre det få steder i koden, eks som i brugerLogin metoden i MainControlleren for at vise hvordan det virker.

7.3 Afsluttende test

Som nævnt i afsnittet Formål og strategi for test har vi forsøgt at køre alle funktioner af programmet igennem for at se at det fungerer som det skal, dette kaldes en White box test²⁷. Man kontrollerer sin white box test ved at indtaste data og kontrollere outputtet - dette kaldes en interaktiv testkørsel.

Derudover findes der mange forskellige former for test såsom valideringstest, unittest og systemtest²⁸. Men da vores program ikke tager flere timer at gå igennem har vi primært gjort brug af White box test både løbende og afsluttende i testforløbet.

Derudover findes der mere alternative test former såsom q-test eller monkey-test²⁹. Q-test går ud på at man taster q hver gang det er muligt - dette vil stoppe vores program i flere funktioner, da der som sagt ikke er try-catch metoder på alle valg menuer. Monkey-test fungerer i realiteten på samme måde, men går ud på at man bare taster vilkårligt på alle taster - dette vil ligesom q-test få vores program til at stoppe, men hvis det var et krav kunne dette undgås ved at implementere try-catches alle de funktioner en bruger skal tage et valg.

Som bruger eller kunde af dette produkt ville man udføre en såkaldt accepttest³⁰. Denne test beskriver om programmet kan det man ønsker det kan. Da vi selv har opsat use cases for dette program, ville en eventuel kunde eller bruger køre disse use cases igennem for at kunne kontrollere, at de forskellige funktionaliteter fungerer optimalt.

I vores program ville man i accepttesten primært kigge efter om programmet indberetter data korrekt, viser korrekt statistik og håndtering af brugere – opretning, sletning og redigering.

²⁷ Litteraturliste 10.1

²⁸ Litteraturliste 10.10

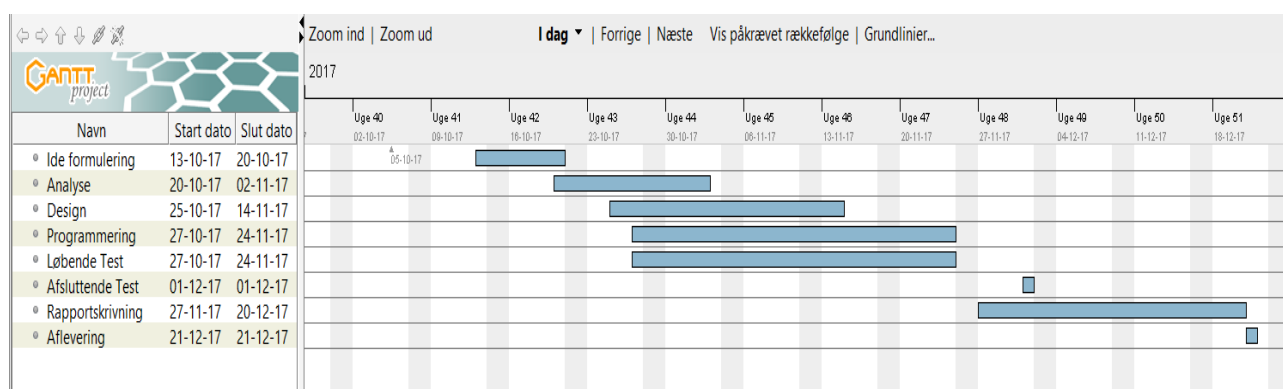
²⁹ Litteraturliste 10.10

³⁰ Litteraturliste 10.10

Vi har selv forsøgt at lave en accepttest på vores use cases og efter disse test kan vi konkludere at programmet er i stand til det, som er beskrevet i use casene. Dog skal det understreges, at det kan være en udfordring at lave en accepttest på ens eget program, da en eventuel kunde måske ønsker andre funktionaliteter end det som vi har udviklet.

8. Konklusion

8.1 Gantt-kort (slut)



Vi er nu færdige med projektet og udarbejder derfor et nyt Gantt-kort³¹ for projektførløbet, som det kom til at se ud. Dette sammenligner vi med Gantt-kortet vi lavede i starten af projektet, for at se hvordan det egentlige forløb kom til at gå i forhold til hvordan vi planlagde det.

Som det fremgik af vores første Gantt-kort, så brugte vi den første tid på at formulere en ide for hvordan vores projekt skulle se ud, og hvilken type program vi ønskede at lave, eftersom vi ikke ønskede at lave et program ud fra de foreslåede ideer. Selve perioden for ide formuleringen kom til at gå som planlagt, med den undtagelse af at vi kom i gang en uge senere end vi planlagde. Dernæst gik vi i gang med at analysere på, hvordan vi skulle lave systemet. Her indgik udarbejdelsen af forskellige diagrammer og modeller. Denne fase kom også til at starte senere end forventet, men endte samtidig også med at være kortere, da vi i denne fase arbejdede mere intensivt, som følge af at vi kom senere i gang end forventet.

Designfasen blev dog som følge af forsinkelsen, også en smule længere end forventet, og kom endvidere til at gå et godt stykke ind i programmeringsfasen. Dette skabte dog ikke de store problemer, da vi fra starten af planlagde for muligheden for at gå tilbage i designfasen og modificere designet af systemet. Det viste sig at være en god ide at planlægge designfasen ind i

³¹ Litteraturliste 10.11

selve programmeringsfasen, da vi undervejs blev nødt til at lave små ændringer efterhånden som arbejdet skred frem.

Programmeringsfasen blev, som følge af at vi kom senere i gang, også forsinket en uge. Selve tiden det tog at programmere, blev dog forkortet betydeligt i forhold til hvad der var planlagt, som følge af intensivt arbejde med klare retningslinjer. Endvidere testede vi kontinuerligt under hele programmeringsfasen som vi fra starten havde planlagt. For den afsluttende test havde vi planlagt et par dage, men det viste sig at være unødvendigt. Det viste sig at være nok med én dag til at vi alle tre kunne teste programmet.

Rapportskrivningen af programmeringsfasen blev først påbegyndt efter koden var færdig, da vi besluttede at dette ville give os mulighed for at gøre koden færdig før vi begyndte at dokumentere denne fase. Vi har gjort brug af dokumentering, rapportskrivning, undervejs i hele forløbet specielt i planlægningsstadiet. Dog ventede vi som sagt med at dokumentere selve koden til efter den var færdig.

Vi havde heller ikke taget højde for retning og gennemlæsning i vores første Gantt-kort, men dette er med i det afsluttende - dette er markeret i kolonnen "rapportskrivning", det vil dog ikke sige at det kun er i denne periode vi har dokumenteret vores projekt.

I enkelte tilfælde har vi gået tilbage i eksempelvis designfasen for at modificere en smule og dette har gjort at vi har redigeret i vores dokumentation undervejs.

Havde vi fulgt vores oprindelige plan ville dette føre til at vi skulle lave massive ændringer i rapporten undervejs, hvilket vi ikke var interesserede i. Vi lavede i gruppen dog en aftale om at vi skulle være færdige med at skrive rapporten d. 7 december, da vi grundet eksamensforberedelse ville have brug for et break mellem den 8 og 13 december. Vi endte med at overholde vores deadline, og holde vores break, hvorefter vi finpudsede vores rapport i de sidste dage af projektperioden.

8.2 Produkt

Produktet vi har udviklet er rent teknisk et program som inneholder medlemmer som skal sykle til arbejdet for nemlig det å sette mer fokus på både mosjon og miljø. Man oppretter en bruker med en pinkode, der man må svare på om man er mann eller kvinne. Når man har laget en bruker kan alle faktisk se at du er medlem samt hvor mange kalorier du har forbrent til sammen. Fra dette kan man godt lage flere undersøkelser og statistikker som går på hvor mye som har blitt forbrent, og hvor aktive medlemmene er. Hvis man lanserer en reklame eller en annen type markedsføring kan dette føre til økt popularitet. For å finne ut av hva som fungerer kan man markedsføre forskjellig i

forskjellige deler av landet for å se hva som gir best resultat for aktiviteten i programmet. Eventuelt kan man se hvor i landet aktiviteten er størst for å finne ut av produktets potensial.

8.3 Konklusion/perspektivering

Kigger man på verden omkring os, så har helsebranchen hæderkronede dage, med stigende interesse i helseprodukter og fitness. Det er en branche som i den seneste tid har oplevet stigende efterspørgsel, og der er i samfundet generelt kommet et større fokus på at man skal være sund, og passe på hvad man putter i sin krop.

Som nævnt tidligere i rapporten, så fungerer vores system som en kalorieberegner. Systemet kunne eks. bruges i mange sammenhænge til at give folk mulighed for at beregne hvor meget de cykler, men samtidig vise hvor sundt det var at cykle til arbejde.

Ikke kun i sammenhængen med at man tilmelder sig "vi cykler til arbejde", men også for folk som generelt bare ønsker at vide hvor meget de cykler på et givent tidspunkt. Det kunne eksempelvis være at man i et fitnesscenter havde et spinninghold, som tilsvarende kunne få glæde af vores system til at holde øje med hvem der giver den mest gas på cyklen i fitnesscentret. Dette kunne være med til at skabe stigende konkurrence blandt de seriøse spinnere og på den måde give folk et incitament til at nå deres mål hurtigere, hvis de kan se præcis hvor langt de er kommet, og hvor meget de mangler for at opnå deres mål.

9. Til videre arbejde

Da vi endnu ikke har lært at arbejde med databaser og lagring af data uden for RAM-lageret begrænser det programmet. Når vi lærer dette kan man i fremtiden arbejde videre med projektet ved at indsætte en funktioner, som kunne lagre de cykelture som en bruger indtaster i systemet. Dette ville samtidig give mulighed for at programmet kunne beregne kalorier mm. i forhold til tid såsom hvor mange kalorier forbrænder en person månedligt eller årligt.

Desuden kunne programmet udvides, så der ikke blot var én gruppe af brugere, men man kunne oprette grupper og tildele brugere i den enkelte gruppe - disse grupper kunne eksempelvis være en enkelt arbejdsplads der ønskede at konkurrere med hinanden eller helt ned til enkelte afdelinger - en gruppe inde i en gruppe. Dette kunne vi godt have tilføjet med den viden vi har, men da vi har ønsket et brugervenligt system, så har vi begrænset det til, at der er logisk administration af

brugere og det er simpelt at indtaste data og bearbejde den data - da vores primære ønske var brugervenlighed.

10. Litteraturliste

10.1 - Lærebog

Intro to Java Programming, Brief Version, Global Edition, Udgave 10
Daniel Liang

10.2 - Beregner kalorier for mænd og kvinder <https://www.cyklistforbundet.dk/Alt-om-cykling/Cykling/Bliv-en-bedre-cyklist/Saa-meget-forbraender-du-i-Maj>

Mand:

$Vægt * 0,26 * \text{antal km}$

Kvinde:

$Vægt * 0,3 * \text{antal km}$

10.3 - Supplerende læring v. Stackoverflow

<https://stackoverflow.com/>

10.4 - 'Arv' slides

Inge-Lise Salomon

10.5 - Definition på hard coding

https://en.wikipedia.org/wiki/Hard_coding

10.6 - Brug af UML

<http://edn.embarcadero.com/article/31863>

10.7 - 'Noget om systemudvikling' slides

Inge-Lise Salomon

10.8 - 'Array og ArrayList' slides

Inge-Lise Salomon

10.9 - 'Klasser og objekter' slides

Inge-Lise Salomon

10.10 - 'Noget om test' slides

Inge-Lise Salomon

10.11 – Gantt-kort

<https://da.wikipedia.org/wiki/Gantt-skema>

11. Bilag

11.1 Bilag A - Samarbejdskontrakt

Samarbejdskontrakt for X12:

Medlemmer:

Martin Jensen,
Malthe Jørgensen,
Trym Ljungqvist

Paragraffer §:

§1: Man kommer til tiden, når gruppemøder er aftalt.

§1.1: Forsinkelse meldes hurtigst muligt.

§1.2: Afbud gives hurtigst muligt, senest 1 time før, valid forklaring kræves.

§2: Effektivt arbejde når gruppen er samlet.

§3: Det forventes at alle bidrager til det fælles arbejde, og møder forberedt til gruppemøder.

§3.1: Gruppearbejde på studiet prioriteres over ex. Fritidsarbejde og fritidsaktiviteter.

§4: Deadlines skal overholdes.

§5: Konstruktiv kritik tages imod, og reflekteres over.

§5.1: Kritik tages aldrig personligt, og kritik rettes ikke personligt mod nogen.

§6: Vi hjælper hinanden.

§7: Ved personlige udfordringer. Ex. Dødsfald eller familiære problemer, meldes det til gruppen.

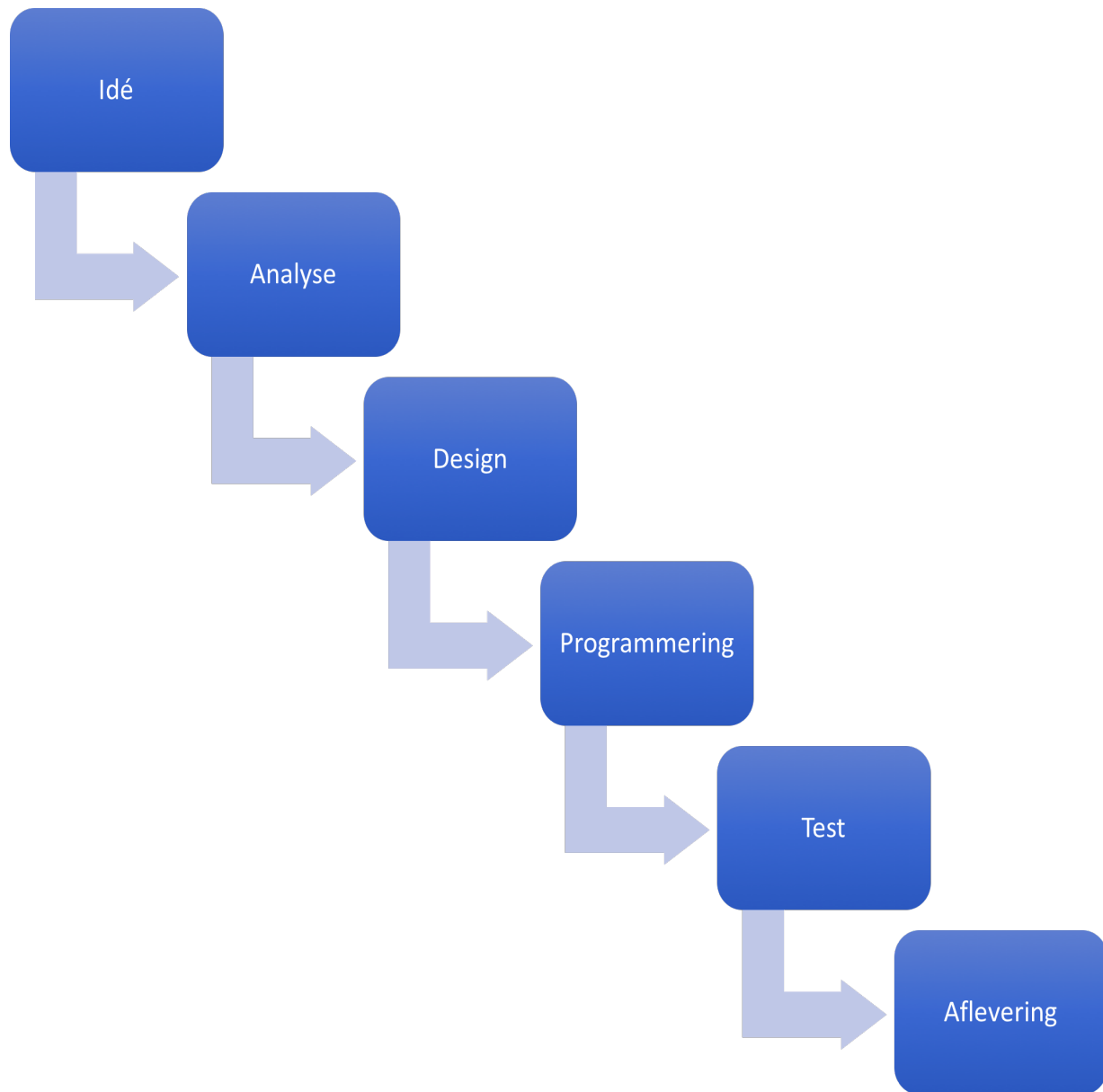
Ved overtrædelse af paragraffer:

§7: Ved første overtrædelse af paragraffer gives kaffe/sodavand til resten af gruppen.

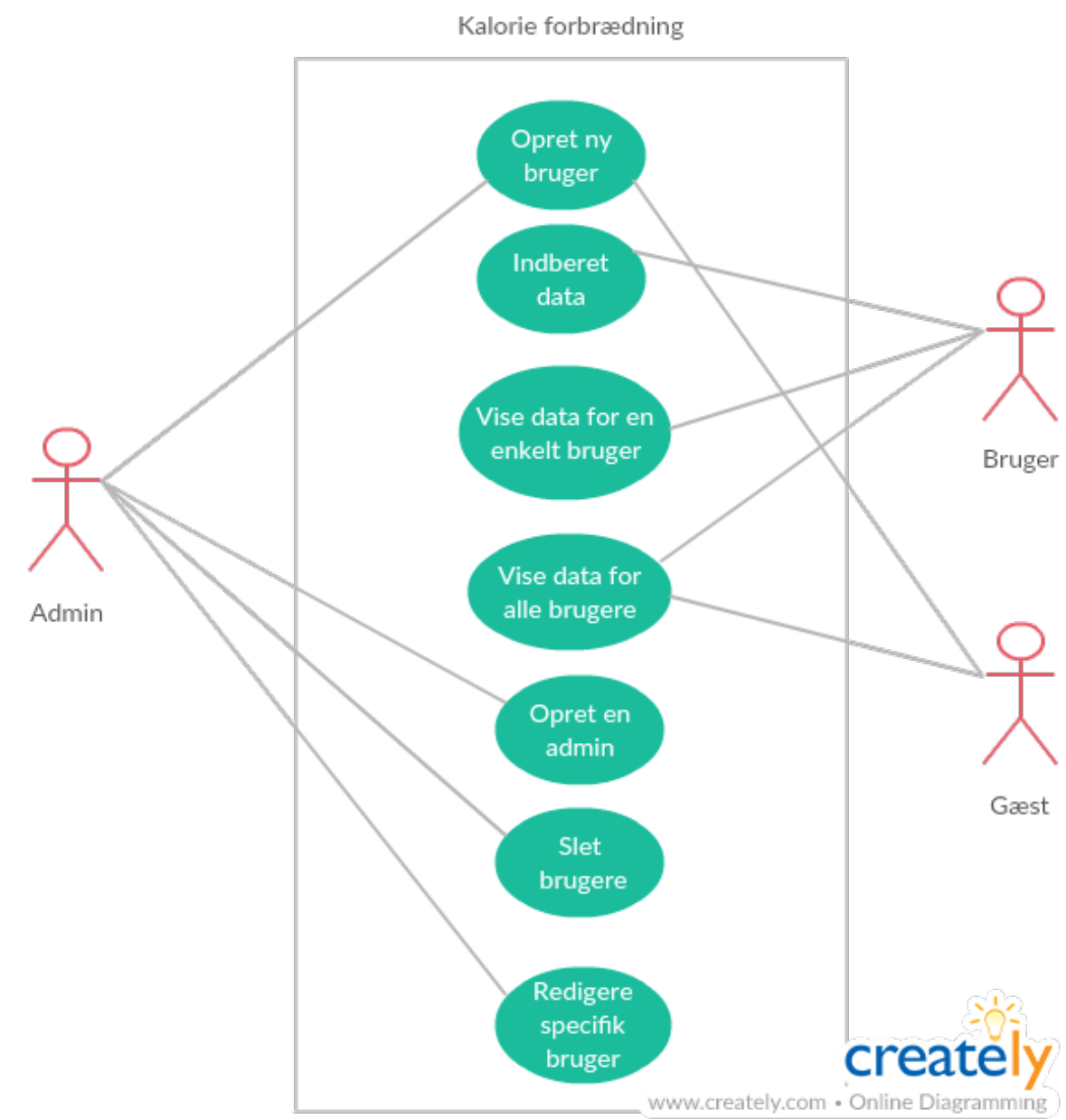
§7.1: Ved anden overtrædelse af paragraffer, mundtlig advarsel.

§7.2: Ved tredje overtrædelse af paragraffer, det pågældende medlems status i gruppen tages til diskussion.

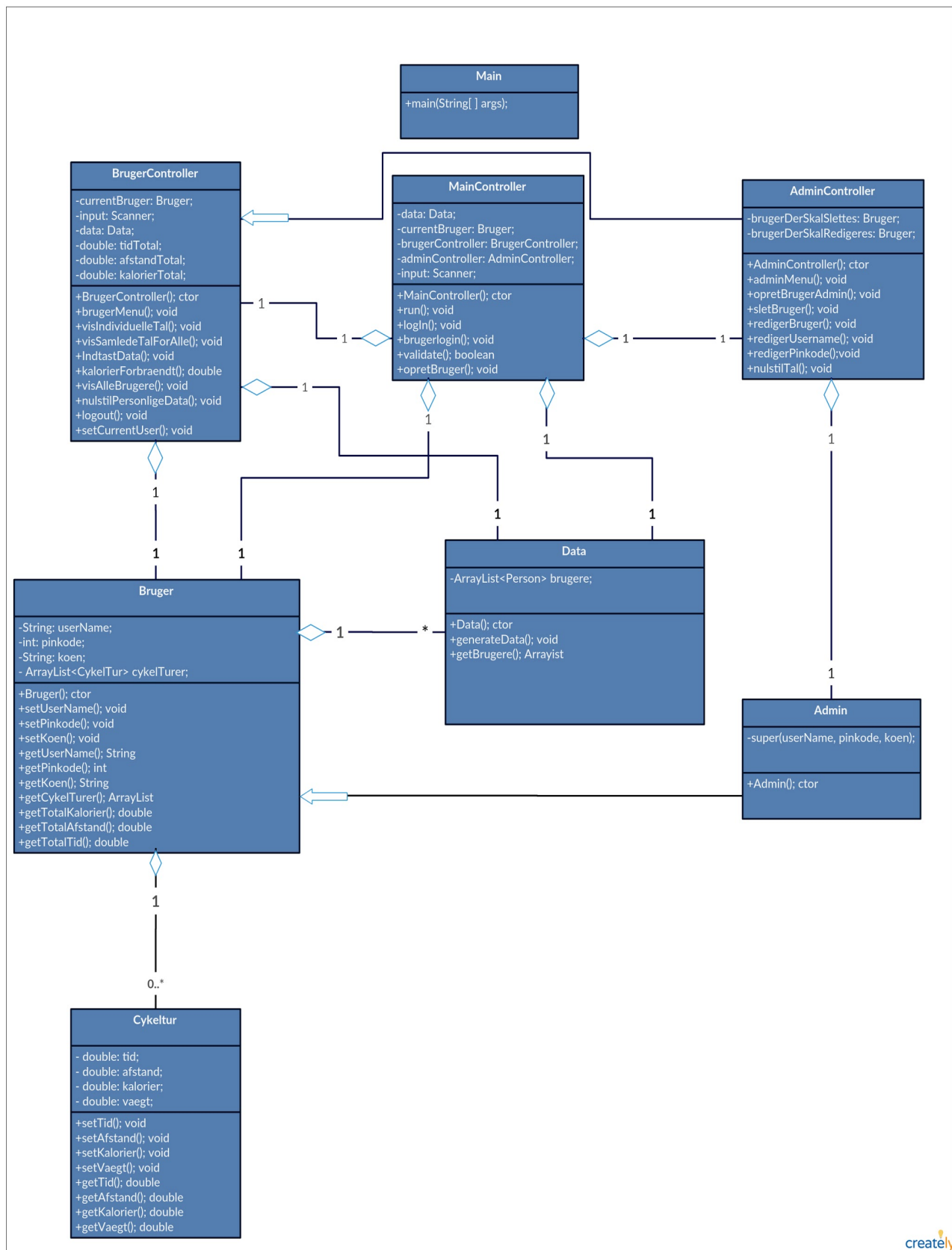
11.2 Bilag B - Vandfaldsmodellen



11.3 Bilag C - Use case diagram



11.4 Bilag D - klassediagram



11.5 Bilag E - programkoden

```
package controllers;
```

```
import data.Data;  
import models.Admin;  
import models.CykelTur;  
import models.Bruger;
```

```
import java.util.InputMismatchException;  
import java.util.Scanner;
```

```
// Denne klasse er udarbejdet af Malthe og Trym
```

```
public class MainController {
```

```
    private Data data;  
    private Bruger currentBruger;  
    private BrugerController brugerController;  
    private AdminController adminController;  
    private Scanner input = new Scanner(System.in);
```

```
    public MainController() {  
        data = new Data();  
        brugerController = new BrugerController(data);  
        adminController = new AdminController(data);  
    }
```

```
    public void run() {  
        login();  
    }
```

```
// Her logger brugeren ind ved at indtaste sit brugernavn og pinkode
```

```
    private void login() {  
        int valg;  
  
        do {  
  
            System.out.println("Vælg:");  
            System.out.println("1) Login");  
            System.out.println("2) Opret dig som bruger");  
            System.out.println("3) Se alle brugere - kræver ikke login");  
            System.out.println("0) Afbryd program");
```



```
    valg = input.nextInt();

    switch (valg) {

        case 1:
            brugerLogin();
            break;
        case 2:
            opretBruger();
            break;
        case 3:
            brugerController.visAlleBrugere();
            break;
        case 0:
            System.out.println("Tak for i dag");
            break;
        default:
            System.out.println("Du har valgt forkert");
    }
} while (valg != 0);

}

/* Programmet checker ved hjælp af validate metoden om det stemmer
overens hvad brugeren har indtastet og om brugeren findes i databasen */

public void brugerLogin() {

    try {
        System.out.println("Indtast Brugernavn");
        String username = input.next();
        System.out.println("Indtast Pinkode");
        int pinkodeIndtast = input.nextInt();

        if (validate(username, pinkodeIndtast)) {
            brugerController.setCurrentUser(currentBruger);
            adminController.setCurrentUser(currentBruger);

            // Her tjekkes om brugeren er admin. Hvis brugeren er admin bliver brugeren
            præsenteret med en avanceret menu
            if (currentBruger instanceof Admin){

                adminController.adminMenu();
            }
        }
    } catch (Exception e) {
        System.out.println("Fejl: " + e.getMessage());
    }
}
```

```
        } else {
            brugerController.brugerMenu();
        }

    } else {
        System.out.println("Forkert brugernavn eller pinkode! Prøv igen:");
    }

} catch (InputMismatchException e) {

    System.out.println("Du skal taste et heltal i pinkode");
    input.next();

}
}

//Validering af om brugeren eksistere

public Boolean validate(String username, int pinkodeIndtast) {
    for (Bruger bruger : data.getBrugere()) {
        if (bruger.getUserName().equalsIgnoreCase(username) && bruger.getPinkode() ==
pinkodeIndtast) {
            currentBruger = bruger;
            System.out.println("\nVelkommen " + username + "\n");
            return true;
        }
    }
    return false;
}

/* Denne metode bruges til at oprette en ny bruger.
   Denne bruger vil som default være almindelig. Kun en admin bruger kan oprette en ny
   admin */

public void opretBruger() {

    String nytUserName;
    int nyPinkode;
    int nyPinkodeTjek;
    String nytKoen="";
    int koenValg;
    boolean eksisterendeBruger;

    do {
        eksisterendeBruger = false;
```

```
System.out.println("Hvad skal dit brugernavn være?");

nytUserName = input.next();

for (Bruger bruger : data.getBrugere()) {

    if (bruger.getUserName().equals(nytUserName)) {
        eksisterendeBruger = true;
        System.out.println("Brugernavnet findes allerede. Prøv igen");
    }
}

} while (eksisterendeBruger);

do {
    System.out.println("Hvad skal din pinkode være?");

    nyPinkode = input.nextInt();

    System.out.println("Indtast din pinkode igen");

    nyPinkodeTjek = input.nextInt();

    if (nyPinkode == nyPinkodeTjek) {

        do{
            System.out.println("Hvad køn er du? \n1) Mand\n2) Kvinde");

            koenValg = input.nextInt();

            if(koenValg==1){
                nytKoen="mand";
            }

            else if(koenValg==2){
                nytKoen="kvinde";
            }

            else {
                System.out.println("Du har valgt forkert. Tast venligst 1 for mand eller 2 for
kvinde");
            }

        }while(koenValg!=1 && koenValg!=2);

        System.out.println("Du har nu oprettet dig som bruger");
    }
}
```

```
//Den nye bruger oprettes og tilføjes til arraylisten af brugere
Bruger bruger = new Bruger(nytUserName, nyPinkode, nytKoen);

data.getBrugere().add(bruger);

//Den nye bruger får en cykeltur der er sat til nul og denne cykeltur bliver tilføjet til
brugeren
CykelTur cykelTur = new CykelTur(0, 0, 0, 0);

bruger.getCykelTure().add(cykelTur);

} else {
    System.out.println("Dine pinkoder matcher ikke");
}
} while (nyPinkode != nyPinkodeTjek);
}
}
```

```
package controllers;
```

```
import data.Data;
import models.CykelTur;
import models.Bruger;
```

```
import java.util.ArrayList;
import java.util.InputMismatchException;
import java.util.Scanner;
```

```
//Denne klasse er udarbejdet af Malthe og Martin
```

```
public class BrugerController {
```

```
    Data data;
    Scanner input = new Scanner(System.in);
    protected Bruger currentBruger;
    protected double tidTotal = 0;
    protected double afstandTotal = 0;
    protected double kalorierTotal = 0;
```

```
public BrugerController(Data data) {
    this.data = data;
}

protected void brugerMenu() {

    //Almindelig brugers menu

    int i;

    do {
        try {
            System.out.println("Vælg hvad du vil gøre nu?\n1) Vis dine samlede tal" +
                "\n2) Vis samlede tal for alle\n3) Indtaste nye tal\n4) Vi alle brugere \n5) Nulstil
dine tal \n0) Log ud");

            i = input.nextInt();

            switch (i) {

                case 1:
                    visIndividuelleTal();
                    break;
                case 2:
                    visSamledeTalForAlle();
                    break;
                case 3:
                    indtastData();
                    break;
                case 4:
                    visAlleBrugere();
                    break;
                case 5:
                    nulstilPersonligeData();
                    break;
                case 0:
                    logout();
                    break;
                default:
                    System.out.println("Du har tastet forkert. Prøv igen!");
            }
        } catch (Exception e) {
            System.out.println("Du har tastet noget forkert. Prøv igen");
        }
    } while (currentBruger != null);
}
```

```
}
```

```
//Her vises currentBrugers tal både samlet og på den sidste tur
```

```
public void visIndividuelleTal() {
```

```
    ArrayList<CykelTur> cykelTure = currentBruger.getCykelTure();  
    double kalorier = currentBruger.getTotalKalorier();  
    double afstand = currentBruger.getTotalAfstand();  
    double tid = currentBruger.getTotalTid();  
    double vaegt = cykelTure.get(cykelTure.size() - 1).getVaegt();
```

```
    System.out.println("\nAlle dine ture lagt sammen");  
    System.out.printf("Du har forbrændt %.1f kalorier\n", kalorier);  
    System.out.printf("Du har kørt %.1f km.\n", afstand);  
    System.out.printf("Du har brugt %.1f minutter\n", tid);  
    System.out.printf("Du vejer %.1f kg.", vaegt);
```

```
    System.out.println("\nPå din sidste tur");
```

```
    System.out.printf("Forbrændte du: %.1f kalorier\n", cykelTure.get(cykelTure.size() -  
1).getKalorier());
```

```
    System.out.printf("Kørte du: %.1f kilometer\n", cykelTure.get(cykelTure.size() -  
1).getAfstand());
```

```
    System.out.printf("Kørte du: %.1f minutter\n", cykelTure.get(cykelTure.size() -  
1).getTid());
```

```
    System.out.printf("Vejede du: %.1f kg.\n", cykelTure.get(cykelTure.size() -  
1).getVaegt());
```

```
    System.out.println("\n");
```

```
}
```

```
/* Her vises samlede tal for alle brugere samt hvem der fører i forhold til mest forbrændt  
kalorier,
```

```
længste distance og længste tid på cyklen*/
```

```
public void visSamledeTalForAlle() {
```

```
//Iterere igennem alle brugere i dataen
```

```
for (Bruger bruger : data.getBrugere()) {
```

```
//Iterere igennem alle brugeres cykelture for at kunne ligge attributterne sammen
```

```
for (CykelTur cykelTur : bruger.getCykelTure()) {
```

```
    tidTotal += cykelTur.getTid();  
    kalorierTotal += cykelTur.getKalorier();  
    afstandTotal += cykelTur.getAfstand();
```

```
    }
```

```
}
```

// Den bruger som har forbrændt flest kalorier findes

Bruger vinderKalorier = **data**.getBrugere().get(0);

for (Bruger bruger : **data**.getBrugere()) {

if (vinderKalorier.getTotalKalorier() < bruger.getTotalKalorier()) {
 vinderKalorier = bruger;

 }
}

// Den bruger som har kørt i længst tid findes

Bruger vinderTid = **data**.getBrugere().get(0);

for (Bruger bruger : **data**.getBrugere()) {

if (vinderTid.getTotalTid() < bruger.getTotalTid()) {
 vinderTid = bruger;

 }
}

// Den bruger som har kørt den længste distance findes

Bruger vinderAfstand = **data**.getBrugere().get(0);

for (Bruger bruger : **data**.getBrugere()) {

if (vinderAfstand.getTotalAfstand() < bruger.getTotalAfstand()) {
 vinderAfstand = bruger;

 }
}

System.**out**.printf("Der er totalt blevet kørt i %.1f minutter\n", tidTotal);

System.**out**.printf("Der er totalt blevet kørt %.1f km.\n", afstandTotal);

System.**out**.printf("Der er totalt blevet forbrændt %.1f kalorier\n", kalorierTotal);

System.**out**.println(vinderKalorier.getUserName() + " har forbrændt mest med " +
vinderKalorier.getTotalKalorier() + " kalorier");

System.**out**.println(vinderTid.getUserName() + " har cyklet i længst tid " +
vinderTid.getTotalTid() + " minutter");

System.**out**.println(vinderAfstand.getUserName() + " har cyklet længst " +
vinderAfstand.getTotalAfstand() + " kilometer");

System.**out**.println("\n");

}

```
public void indtastData() {

    double afstand = 0;
    double vaegt = 0;
    double tid = 0;

    /*
    Brugere skal nu indtaste følgende:
    Afstand
    Vægt
    Tid
    */

    do {
        try {
            System.out.println("Hvor langt cykler du til arbejde?(km)");
            afstand = input.nextDouble();

            System.out.println("Hvor meget vejer du?(kg)");
            vaegt = input.nextDouble();

            System.out.println("Hvor lang tid tog det?(minutter)");
            tid = input.nextDouble();

        } catch (InputMismatchException e) {
            System.out.println("Du skal indtaste tal");
            System.out.println("Har du indtastet decimal tal så brug '.' og ikke punktum");
            input.next();
        }
    } while (afstand < 0);

    double kalorierForbraendtPaaTuren = kalorierForbraendt(afstand, vaegt, tid);
    CykelTur cykelTur = new CykelTur(afstand, vaegt, tid, kalorierForbraendtPaaTuren);
    currentBruger.getCykelTure().add(cykelTur);
}

//Beregner hvor mange kalorier der blev forbrændt på den sidste tur og printer det
public double kalorierForbraendt(double afstand, double vaegt, double tid) {

    double talDerBeregnerKalorieForbraening = 0;
    double kalorier;
    if (currentBruger.getKoen().equals("mand")) {
        talDerBeregnerKalorieForbraening = 0.26;
    }
}
```



```
    } else if (currentBruger.getKoen().equals("kvinde")) {  
        talDerBeregnerKalorieForbraening = 0.3;  
    }  
  
    kalorier = afstand * vaegt * talDerBeregnerKalorieForbraening;  
    System.out.printf("\nDu har forbrændt %.1f kalorier på vej til arbejde\n", kalorier);  
    System.out.println("\n");  
  
    return kalorier;  
}  
  
//Viser alle brugeres navn og deres totale kalorieforbrændning  
public void visAlleBrugere() {  
  
    System.out.println("Brugernavn      Total kalorieforbrændning");  
  
    for (int i = 0; i < data.getBrugere().size(); i++) {  
  
        System.out.printf("%-10s \t %,10.2f\n", data.getBrugere().get(i).getUserName(),  
data.getBrugere().get(i).getTotalKalorier());  
  
    }  
  
    System.out.print("\n");  
  
}  
  
//Brugeren nulstiller sine tal til 0  
private void nulstilPersonligeData() {  
  
    currentBruger.getCykelTure().removeAll(currentBruger.getCykelTure());  
  
    System.out.println("Du har nulstillet dine tal\n");  
  
    CykelTur nyC = new CykelTur(0, 0, 0, 0);  
  
    currentBruger.getCykelTure().add(nyC);  
  
}  
  
//Denne metode bruges til at logge ud  
public void logout() {  
  
    currentBruger = null;  
}
```

```
public void setCurrentUser(Bruger currentUser) { this.currentBruger = currentUser; }  
}
```

```
package controllers;
```

```
import data.Data;  
import models.Admin;  
import models.CykelTur;  
import models.Bruger;
```

```
//Denne klasse er udarbejdet af Martin og Trym
```

```
public class AdminController extends BruggerController {
```

```
    Brugger brugerDerSkalRedigeres;  
    Brugger brugerDerSkalSlettes;
```

```
// Da klassen er nedarvning af BruggerController kan de samme metoder og attributter  
bruges
```

```
    public AdminController(Data data) {  
        super(data);  
    }
```

```
//Den avancerede admin menu
```

```
    public void adminMenu() {
```

```
        int valg;
```

```
        do {
```

```
            System.out.println("Vælg hvad du vil gøre nu?");  
            System.out.println("1) Vis dine samlede tal");  
            System.out.println("2) Vis samlede tal for alle");  
            System.out.println("3) Indtaste nye tal");  
            System.out.println("4) Vis alle brugere");  
            System.out.println("5) Opret bruger");  
            System.out.println("6) Slet bruger");  
            System.out.println("7) Rediger bruger");  
            System.out.println("0) Log ud");
```

```
            valg = input.nextInt();
```

```
switch (valg) {
    case 1:
        visIndividuelleTal();
        break;
    case 2:
        visSamledeTalForAlle();
        break;
    case 3:
        indtastData();
        break;
    case 4:
        visAlleBrugere();
        break;
    case 5:
        opretBrugerAdmin();
        break;
    case 6:
        sletBruger();
        break;
    case 7:
        redigerBruger();
        break;
    case 0:
        logout();
        break;
    default:
        System.out.println("Du har valgt forkert. Prøv igen");
}

} while (currentBruger != null);

}

// Her kan en admin oprette en ny bruger. Admin kan også oprette en ny admin
public void opretBrugerAdmin() {

    String nytUserName;
    int nyPinkode = 0;
    int nyPinkodeTjek;
    String koen = null;
    boolean eksisterendeBruger;
    int typeAfBruger;
    int valg;
```

```
do {
    eksisterendeBruger = false;

    System.out.println("Hvad skal den nye brugers brugernavn være?");

    nytUserName = input.next();

    for (Bruger bruger : data.getBrugere()) {

        if (bruger.getUserName().equalsIgnoreCase(nytUserName)) {
            eksisterendeBruger = true;
            System.out.println("Denne bruger findes allerede. Prøv igen");
        }

    }

} while (eksisterendeBruger);

do {
    try {
        System.out.println("Hvad skal den nye brugers pinkode være?");

        nyPinkode = input.nextInt();

    } catch (Exception e) {
        System.out.println("Du skal indtaste et heltal. Prøv igen.");
        input.next();
    }

    System.out.println("Indtast pinkoden igen");

    nyPinkodeTjek = input.nextInt();

    if (nyPinkode == nyPinkodeTjek) {

        do{
            System.out.println("Hvad køn er brugeren? \n1) Mand\n2) Kvinde");

            valg = input.nextInt();

            if(valg==1){
                koen="mand";
            }

            else if(valg==2){
```

```
        koen="kvinde";
    }

    else {
        System.out.println("Tast venligst 1 for mand eller 2 for kvinde");
    }

}while(valg!=1 && valg!=2);

System.out.println("Vælg hvad du vil");
System.out.println("1) Oprette en almindelig bruger");
System.out.println("2) Oprette en admin bruger");

typeAfBruger = input.nextInt();

do {
    switch (typeAfBruger) {

        case 1:
            System.out.printf("Du har nu oprettet %s som bruger\n", nytUserName);

            Bruger bruger = new Bruger(nytUserName, nyPinkode, koen);

            data.getBrugere().add(bruger);

            CykelTur cykelTur = new CykelTur(0, 0, 0, 0);

            bruger.getCykelTure().add(cykelTur);
            break;

        case 2:
            System.out.printf("Du har nu oprettet %s som admin\n", nytUserName);

            Admin admins = new Admin(nytUserName, nyPinkode, koen);

            data.getBrugere().add(admins);

            CykelTur cykelTurAdmin = new CykelTur(0, 0, 0, 0);

            admins.getCykelTure().add(cykelTurAdmin);
            break;

        default:
            System.out.println("Vælg venligst 1 eller 2");
    }
} while (typeAfBruger != 1 && typeAfBruger != 2);
```

```
    } else {  
        System.out.println("Pinkoderne matcher ikke");  
    }  
} while (nyPinkode != nyPinkodeTjek);  
}  
  
//Admin kan slette en bruger  
private void sletBruger() {  
  
    boolean tjek;  
    int i;  
    int indexAfBrugerDerSkalSlettes;  
  
    do {  
        System.out.println("Hvilken bruger vil du gerne slette");  
  
        for (i = 0; i < data.getBrugere().size(); i++) {  
  
            System.out.print(i+1 + " ") + data.getBrugere().get(i).getUserName() + "\n");  
  
        }  
        System.out.println("0) Afslut");  
  
        indexAfBrugerDerSkalSlettes = input.nextInt();  
  
        if(indexAfBrugerDerSkalSlettes==0){  
            return;  
        }  
  
        if(indexAfBrugerDerSkalSlettes>data.getBrugere().size() ||  
indexAfBrugerDerSkalSlettes<0){  
  
            System.out.println("Du har valgt forkert");  
            tjek = true;  
        }  
  
        else{  
            brugerDerSkalSlettes =  
this.data.getBrugere().remove(indexAfBrugerDerSkalSlettes-1);  
            System.out.println("Du har slettet brugeren\n");  
            tjek = true;  
        }  
    }  
}
```

```
    }while (tjek != true);

}

//Her kan admin redigere en specifik bruger
public void redigerBruger() {

    boolean tjek = false;
    int i;
    int indexAfBrugerDerSkalRedigeres;

    do {
        System.out.println("Hvilken bruger vil du gerne redigere");

        for (i = 0; i < data.getBrugere().size(); i++){

            System.out.print(i+1 + " " + data.getBrugere().get(i).getUserName() + "\n");

        }
        System.out.println("0) Afslut");

        indexAfBrugerDerSkalRedigeres = input.nextInt();

        if(indexAfBrugerDerSkalRedigeres==0){
            return;
        }

        if(indexAfBrugerDerSkalRedigeres>data.getBrugere().size() ||
indexAfBrugerDerSkalRedigeres<0){

            System.out.println("Du har valgt forkert");
            tjek = true;
        }

        else{
            brugerDerSkalRedigeres =
this.data.getBrugere().get(indexAfBrugerDerSkalRedigeres-1);

            System.out.println("Du har valgt at redigere " +
brugerDerSkalRedigeres.getUserName() + "\n");

            System.out.println("Hvad vil du redigere");
            System.out.println("1) Rediger brugernavn");
            System.out.println("2) Rediger pinkode");
        }
    }
}
```

```
System.out.println("3) Nulstil tal");
System.out.println("0) Afslut redigering");

switch (input.nextInt()){

    case 1:
        redigerUsername();
        break;
    case 2:
        redigerPinkode();
        break;
    case 3:
        nulstilTal();
        break;
    case 0:
        break;
    default:
        System.out.println("Du har valgt forkert. Prøv igen");

}

} while (tjek);
}

//Admin kan redigere brugerens username
private void redigerUsername() {

    boolean eksisterendeBruger;
    String nytUserName;

    do {
        eksisterendeBruger = false;

        System.out.println("Hvad skal det nye brugernavn være?");

        nytUserName = input.next();

        for (Bruger bruger : data.getBrugere()) {

            if (bruger.getUserName().equals(nytUserName)) {
                eksisterendeBruger = true;
                System.out.println("Brugernavnet findes allerede. Prøv igen");
            }

        }

    }

}
```



```
        brugerDerSkalRedigeres.setUserName(nytUserName);

    } while (eksisterendeBruger);

    System.out.println("Brugernavnet er nu ændret\n");

}

//Admin kan redigere brugerens pinkode
private void redigerPinkode() {

    int pinkodeGammel;
    int nyPinkode;
    int nyPinkodeTjek;
    boolean tjek = false;

    System.out.println("Indtast den gamle kode");

    pinkodeGammel = input.nextInt();

    do {

        if (pinkodeGammel == brugerDerSkalRedigeres.getPinkode()) {

            System.out.println("Hvad skal den nye pinkode være?");

            nyPinkode = input.nextInt();

            System.out.println("Indtast den nye pinkode igen");

            nyPinkodeTjek = input.nextInt();

            if (nyPinkode == nyPinkodeTjek) {
                tjek = true;
                brugerDerSkalRedigeres.setPinkode(nyPinkode);
            }

            else {
                System.out.println("Pinkoderne matcher ikke");
            }

        }

        else{
            tjek = true;
            System.out.println("Du har ikke tastet den gamle kode korrekt");
        }
    }
}
```

```
    }  
    System.out.println("Pinkoden er blevet ændret\n");  
  
    }while(tjek != true);  
  
}  
  
//Admin kan nulstille den specifikke brugers tal  
private void nulstilTal() {  
  
    brugerDerSkalRedigeres.getCykelTure().removeAll(brugerDerSkalRedigeres.getCykelTure())  
    ;  
  
    CykelTur c = new CykelTur(0, 0, 0, 0);  
  
    brugerDerSkalRedigeres.getCykelTure().add(c);  
  
    System.out.println("Du har nulstillet brugerens tal\n");  
  
    }  
  
}  
  


---

  
package models;  
  
import java.util.ArrayList;  
  
//Denne klasse er udarbejdet af Malthe og Trym  
  
public class Bruger {  
  
    private String userName;  
    private int pinkode;  
    private String koen;  
    private ArrayList<CykelTur> cykelTure = new ArrayList<>();  
  
    public Bruger(String userName, int pinkode, String koen) {  
        this.userName=userName;  
        this.pinkode=pinkode;  
        this.koen=koen;  
    }  
  
//Dette er set metoder
```

```
public void setUsername(String userName) { this.userName = userName; }
```

```
public void setPinkode(int pinkode) { this.pinkode = pinkode; }
```

```
public void setKoen(String koen) { this.koen = koen; }
```

```
public void setCykelTure() { this.cykelTure = cykelTure; }
```

```
//Dette er get metoder
```

```
public String getUsername() { return userName; }
```

```
public int getPinkode() { return pinkode; }
```

```
public String getKoen() { return koen; }
```

```
public ArrayList<CykelTur> getCykelTure() { return cykelTure; }
```

```
//Dette bruges til at sammenlægge brugerens kalorier for alle ture
```

```
public double getTotalKalorier() {  
    double totalKalorier = 0;  
    for(CykelTur cykelTur: cykelTure) {  
        totalKalorier += cykelTur.getKalorier();  
    }  
    return totalKalorier;  
}
```

```
//Dette bruges til at sammenlægge brugerens afstand for alle ture
```

```
public double getTotalAfstand(){  
    double totalAfstand = 0;  
  
    for (CykelTur cykelTur : cykelTure) {  
        totalAfstand += cykelTur.getAfstand();  
    }  
  
    return totalAfstand;  
}
```

```
//Dette bruges til at sammenlægge brugerens tid på cyklen for alle ture
```

```
public double getTotalTid(){  
    double totalTid = 0;  
  
    for (CykelTur cykelTur : cykelTure) {  
        totalTid += cykelTur.getTid();  
    }  
}
```

```
    }  
  
    return totalTid;  
}  
  
}
```

```
package models;
```

```
//Denne klasse er udarbejdet af Malthe og Martin
```

```
//Denne klasse er nedarvet af bruger. Det eneste klassen bruges til er at tjekke hvad en  
bruger er i login
```

```
public class Admin extends Bruger {  
  
    public Admin(String userName, int pinkode, String koen) {  
        super(userName, pinkode, koen);  
    }  
  
}
```

```
package models;
```

```
//Denne klasse er udarbejdet af Trym og Martin
```

```
public class CykelTur {  
  
    private double tid;  
    private double afstand;  
    private double kalorier;  
    private double vaegt;  
  
    public CykelTur(double afstand, double vaegt, double tid, double kalorier){  
  
        this.tid= tid;  
        this.afstand= afstand;  
        this.kalorier=kalorier;  
        this.vaegt= vaegt;  
    }  
  
}
```

```
//Dette er getter for cykelturen
```

```
public double getTid() { return tid; }

public double getAfstand() {return afstand; }

public double getKalorier() { return kalorier; }

public double getVaegt() { return vaegt; }
}
```

```
package data;
```

```
import models.Admin;
import models.CykelTur;
import models.Bruger;
```

```
import java.util.ArrayList;
```

```
//Denne klasse er udarbejdet af Malthe og Trym
```

```
public class Data {
```

```
    ArrayList<Bruger> brugere = new ArrayList<>();
```

```
    public Data (){
        generateData();
    }
```

```
    public void generateData() {
```

```
//Dette er nogle prædefinerede brugere. Inge-Lise er admin
```

```
    Admin admin1 = new Admin("Inge-Lise", 1111, "kvinde");
```

```
    Bruger bruger1 = new Bruger("Michael", 2222, "mand");
```

```
    Bruger bruger2 = new Bruger("Filip", 3333, "mand");
```

```
    brugere.add(admin1);
```

```
    brugere.add(bruger1);
```

```
    brugere.add(bruger2);
```

```
//Der oprettes prædefinerede cykelture. Hertil tilføjes de prædefinerede cykelture til en bruger
```

```
    CykelTur c1 = new CykelTur(0, 0, 0, 0);
```

```
    CykelTur c2 = new CykelTur(0,0,0,0 );
```

```
    CykelTur c3 = new CykelTur(0, 0,0, 0 );
```

```
        admin1.getCykelTure().add(c1);  
        bruger1.getCykelTure().add(c2);  
        bruger2.getCykelTure().add(c3);  
    }  
  
    public ArrayList<Bruger> getBrugere() { return brugere; }  
  
}
```