Svømmeklubben Delfinen
Prøve-eksamen Projekt

Gruppe Pizza - Kasper Lydeking, Mads Frederiksen, Martin Larsen og Kasper Vodschou

Indholdsfortegnelse

Opgaveformulering

Svømmeklubben Delfinen er en mindre klub, der er i vækst. Klubbens ledelse ønsker derfor at få udviklet et administrativt system til at styre medlemsoplysninger, kontingenter og svømmeresultater.

1. Det er klubbens formand, der tager sig af nye medlemmer.

2. Ved indmeldelse i klubben registreres diverse stamoplysninger om personen herunder alder.

3. Desuden registreres oplysninger om personens ønskede aktivitetsform, det vil sige aktivt eller passivt medlemskab, junior eller senior svømmer, motionist eller konkurrencesvømmer.

4. Klubbens kasserer tager sig af alt vedrørende kontingentbetaling.

5. Kontingentets størrelse er betinget af flere forhold. For aktive medlemmer er kontingentet for ungdoms svømmere (under 18 år) 1000 årligt, for seniorsvømmere (18 år og over) 1600 kr. årligt. For medlemmer over 60 år gives der 25 % rabat af senior taksten. For passivt medlemskab er taksten 500 kr. årligt.

6. Kassereren har ønsket, at systemet kan vise en oversigt over medlemmer, der er i restance.

7. Konkurrencesvømmerne har tilknyttet en træner. Konkurrencesvømmerne er inddelt i 2 hold efter alder. Ungdomsholdet er for svømmere under 18 år. Seniorholdet er for svømmere på 18 og over.

8. Hver konkurrencesvømmer er desuden registreret i forhold til hvilke svømmediscipliner, han er aktiv i.
   Inden for hver svømmedisciplin registreres den enkelte svømmers bedste træningsresultat og dato løbende.

9. For de svømmere, der har deltaget i konkurrencer, registreres stævne, placering og tid.

10. Det er på baggrund af de enkelte svømmers resultater, at træneren udtager svømmere til deltagelse i konkurrencer. Træneren ønsker derfor en oversigt, der kan vise klubbens top 5 svømmere inden for hver svømmedisciplin.

Intro

Vi har fået som opgave at udvikle et program til en svømmeklub (Delfinen), hvor programmet skal kunne oprette medlemmer, se hvem der er i restance, indtaste træningstider og konkurrencetider, hvor man skal kunne se de 5 hurtigste svømmere i klubben.

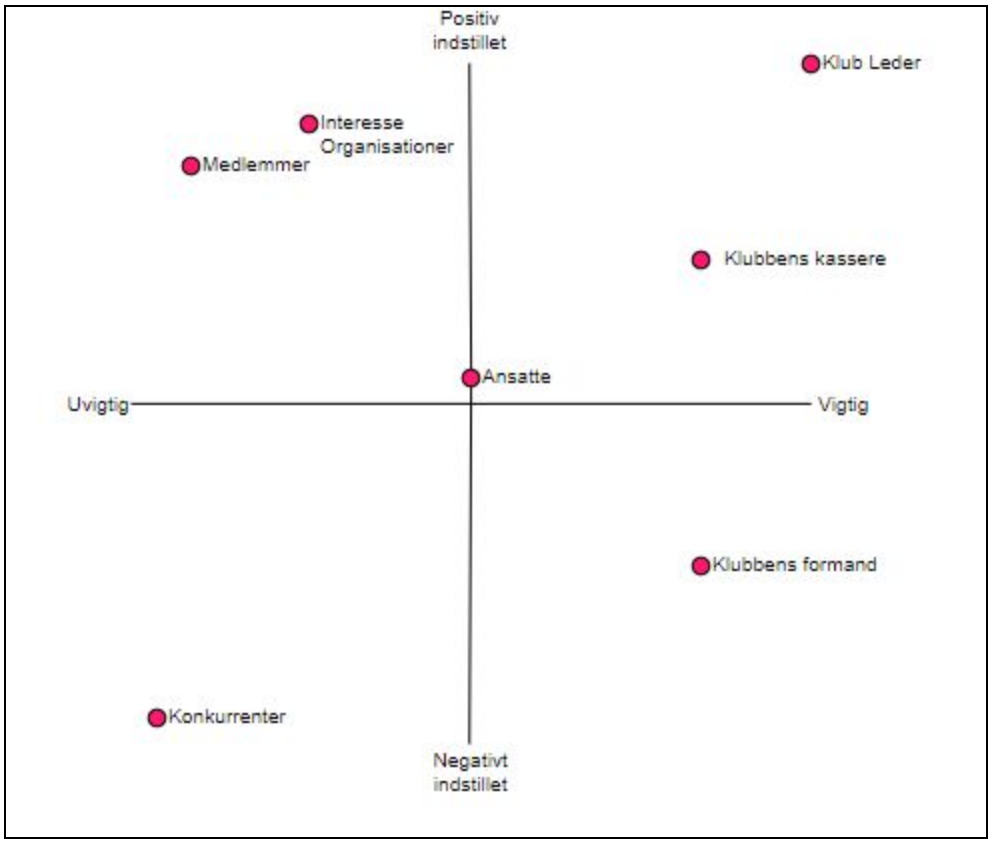
Som tidsplan har vi lavet et Gantt-chart.

Vi anskuet problemet og startet med at få et overblik over problemets indhold, ved først at bruge UML, i form af use cases, domæne modeller, klasse modeller og sekvensdiagrammer. Der er brugt draw.io og Visual Paradigm til at hjælpe med at illustrere diverse modeller og diagrammer, hvor vi støttende har brugt bogen Larman, C. (2002) *Applying UML and patterns.*

Til at løse problemet, har vi valgt at bruge Java til at udvikle programmet, hvor vi har trukket primært fra egen erfaring og lærebogen, Reges, S. & Stepp, M. (2010) *Building Java Programs, A back to Basics Approach 2nd. Edition.* Desuden har vi også brugt stackoverflows hjemmeside, brugt diverse andre hjemmesider.

Herudover har vi lavet en interessent analyse af programmet til svømmeklubben, lavet en SWOT analyse af vores egen gruppe (Vi har antaget vi er en lille virksomhed for opgavens skyld) og lavet en risikoanalyse samt handlingsplan på risikomomenter. Der er igen primært trukket fra egen erfaring, men der er støttet med Storm-Henningsen, P., Skriver, H. & Staunstrup, E. (2017) *Organisation.*

Interessentanalyse



Klubbens leder er positivt indstillet da det er ham der har bestilt programmet, samt er han meget vigtig da det i bund og grund er ham der kan lukke for projektet.

Klub formandens position er baseret på at han er vigtig for oprettelsen af nye medlemmer, men er negativt indstillet da han skal lære et nyt program at kende.

Klubbens kasserer er ligeledes vigtig da han skal indkassere pengene og have oversigt over medlemmer der er i restance. Bedømt til at være positivt indstillet da han selv er kommet med input til systemkrav, under antagelse af at klubben ikke før har haft et elektronisk system til at hjælpe.

Klub medlem er baseret på at de ikke kommer til at have en indflydelse på systemet, men kan være positivt indstillet da der kommer bedre management på tider til træning og stævner.

Ansatte (især trænere er der taget i betragtning) skal også bruge systemet for at holde styr på tider i forhold til træning og stævner. Bedømt til at være lidt mere positiv da det bliver måske nemmere at holde styr på tider, men kan argumenteres for at være negativ da de også skal lære et nyt system.

Konkurrenter er bedømt til at være negativt indstillet, da det er en svømmeklub der er i vækst og der kan være bekymring for overtag af kunder mm. Bedømt til slet ikke at være vigtig for udviklingen af systemet, men kan i skræk scenarier, være sabotage og lignende.

Interesseorganisationer er positivt indstillet, da det kan hjælpe dem med organisering. uvigtig da de ikke har indflydelse på programmet.

Informering:

Vi ønsker en god informationsstrøm med **klubbens leder**, da det er ham der har bestilt vores service, og har en interesse for at han er tilfreds, samt hvis der er ændringer. Tilmed at give status på vores progression.

Ønsker en god dialog med **klubbens formand**, da han er en af brugerne af systemet. Dialogen skal også bruges til at systemkrav er klarlagt og da han kan være negativ anlagt, kan det være at han skal "varmes lidt op" for systemet.

Klubkasserer er endnu en af brugerne, og da han allerede lader til at være positiv anlagt, er der mere fokus på progression, end dialog - men ønsker stadig dialog i tilfælde af ændringer eller tilføjelser.

Ansatte er de sidste brugere til systemet, og det tilstræbes at have dialog med dem også, da de skal indtaste forskellige inputs. Information dertil er også tilstræbt.

Klubbens medlemmer er ikke et direkte mål i forhold til udviklingen af systemet og det regnes med at medlemmerne får information gennem ansatte eller klubben selv.

Konkurrenter er ikke en faktor som vi har problemer med.

Interesseorganisationer er også informeret, da de kan have en interesse i systemet, da det kan tilhjælpe organisatoriske opgaver.

SWOT Analyse

SWOT Analysen tager udgangspunkt i at vi er en lille organisation.

SWOT Gruppe Pizza	Positiv	Negativ
Intern	**Strength:**	**Weakness:**
	· Lille og fokuseret · Erfaring m. management system · Lokalisation	· Lav kapital · Lav ressource · Langsom til genrejsning efter store fejl · Laver et system af gangen
Ekstern:	**Opportunity:**	**Threat:**
	· Lave systemer til andre klubber · Sælge koden · Mere branding	· Større firmaer · Markedssituationen · Systemnedbrud (eksterne årsager)

Styrker

Lille og fokuseret, da vi er en lille virksomhed, regnes det med at vi kun kan varetage en opgave af gangen, hvilket gør at alle vores ressourcer bliver brugt på det projekt.

Erfaring med management systemer, da det er det eneste vi har lavet - se Car Wash

Lokalisation vi er tæt på svømmeklubben og kan hurtigt og nemt yde service og mødes med svømmeklubben.

Svagheder

Lav kapital, da vi igen er en lille virksomhed, som ikke kan påtage sig større opgaver og flere opgaver på samme tid.

Få ressourcer, da vi er en lille virksomhed, har vi heller ikke så mange ansatte, og har ikke ressourcer vi kan hente fra andre projekter hvis nødvendigt.

Svært ved at komme sig efter fatale fejl, grundet den lave kapital og lille virksomhed, vi er meget afhængig af at vores projekter går godt.

Grundet ressourcer kan vi kun arbejde på en opgave af gangen.

Muligheder

Virksomhedens brand kan blive bedre ved at levere et solidt program.

Vi kan muligvis sælge koden, eller videre udvikle til andre systemer eller kunder.

Som i forlængelse af branding, kan vi få muligheder for at sælge enten til klubbens konkurrenter eller andre sportsklubber.

Trusler

Større firmaer med flere ressourcer har større konkurrence egenskaber end vi har.

Markedssituation kan gøre at der ikke er så store behov for små projekter og små virksomheder.

Systemnedbrud i form af strøm, oversvømning eller andre naturkatastrofer kan, afhængelig af skaden, forsinke os drastisk.

Risikoanalyse

Risikoskema version: 1.0

Risiko moment	Sandsynlighed	Konsekvens	Produkt	Præventiv	Ansvarlig
Medlem forlader gruppen	1	5	5	Alle kender opgaverne til at erstatte medlemmet. God arbejdsklima	Alle
Sygdom	3	5	15	2 deler arbejdsopgaverne	Opgave ansvarlige
Fejl Estimeringer	5	3	15	Tidsplan/Handlingsplan	Alle
Krav ændringer	1	10	10	Løbende kontakt med kunden	Alle
Implementering Problemer	5	8	40	Prototype/beta test	Alle
Utilgængelig kode	1	10	10	Mange backups	Backup ansvarlig
Mangelfuld test	3	10	30	Lav testansvarlig	Testansvarlig

Risikoskema version: 1.01

Risiko moment	Sandsynlighed	Konsekvens	Produkt	Præventiv	Ansvarlig
Medlem forlader gruppen	1	5	5	Alle kender opgaverne til at erstatte medlemmet. God arbejdsklima	Alle
Sygdom	3	3	9	2 deler arbejdsopgaverne	Opgave ansvarlige
Fejl Estimeringer	3	3	9	Tidsplan/Handlingsplan	Alle
Krav ændringer	1	10	10	Løbende kontakt med kunden	Alle
Implementering Problemer	2	8	16	Prototype/beta test	Alle
Utilgængelig kode	1	10	10	Mange backups	Backup ansvarlig
Mangelfuld test	2	10	20	Lav testansvarlig	Testansvarlig

Eftersom vi har fokus på de problemer, falder de fleste problemer enten i konsekvens eller sandsynlighed.

Et medlem der forlader gruppen forventes stadig at have samme konsekvens selv når vi har en handleplan og er så godt forberedt som vi kan være for det.

Sygdom forventes at have en mindre konsekvens, da vi har gennem risikoskemaet, kommet frem til en løsning, samt opmærksomheden på dette problem.

Fejlestimeringer er mindre sandsynligt, da vi har et øje på dette og følger det mere nøje.

Kravændringer forventer vi ikke sker noget ved, da det er en opgave udleveret til en masse gruppe og det ikke bare er lige til at skifte opgavekrav på et uddannelsessted.

Implementeringsproblemer forventes at have en mindre sandsynlighed for at ské, da vi tager forbehold for dette, efter opmærksomheden på dette efter første risikoskema.

Utilgængelig kode ser vi med lignende sandsynlighed og konsekvens, da cloud servere er ekstremt pålidelige, og selv med de præventive tiltag som vi har taget, har dette problem stadig samme konsekvens.

Mangelfuld testing ser vi som havende en mindre sandsynlighed efter første risikoskema, da der nu er mere opmærksomhed på dette.

Handlingsplan

Medlem forlader gruppen		
Initiativ	**Tidsplan**	**Ansvarlig**
Det tilstræbes at gruppen hurtigst muligt aftaler hvem overtager de forladte arbejdsopgaver.	Hurtigst muligt	Hele gruppen
Sikring af arbejdsopgaven er mulig.	Efter arbejdsopgaven er blevet uddelt	Opgave modtager og gruppen

Sygdom		
Initiativ	**Tidsplan**	**Ansvarlig**
Melding til resten af gruppen om hvorvidt det kommer til at have nogen følge for opgaven.	Hurtigst muligt	Sygeramte
Syge-makker overtager opgaven	Efter syge ramte har meldt sig syg og videregivet sin opgave (Hvis nødvendigt)	Syge ramte og makkeren
Opfølgning af syge ramtes status	Afhængig af situationen, typisk når det vurderes at syge ramte ikke opfylder sin arbejdsopgave	Syge ramte og gruppen

Fejlestimeringer		
Initiativ	**Tidsplan**	**Ansvarlig**
Omrokering af ressourcer efter skredet tidsplan, potentiel ekstra arbejde	Hurtigst muligt	Hele gruppen
Opfølgning af status på diverse arbejdsopgaver	Dagligt efter om-rokering	Hele gruppen

Krav ændringer		
Initiativ	**Tidsplan**	**Ansvarlig**
Forhandling med kunden omkring forlængelse til levering af produkt og forhandling om betaling for produktet	Hurtigst muligt	Hele gruppen
Gen-opdeling af arbejdsopgaver, potentiel om-rokering af ressourcer	Efter aftale med kunden	Hele gruppen
Kontinuerlig samtale med kunden om potentiel andre kravændringer og opfølgning på de implantationer vi kommer med	Efter aftale med kunden	Hele gruppen

Implementering Problemer		
Initiativ	**Tidsplan**	**Ansvarlig**
Identificere fejl, potentiel om-rokering af ressourcer hvis det er et ressourceproblem	Hurtigst muligt	Hele gruppen
Kontakt til support firma, eller rådgivningsfirma	Afhængig af problemet (Hurtigst muligt)	Hele gruppen
Opfølgning af problemet	Efter afløst problem	Hele gruppen

Utilgængelig Kode		
Initiativ	**Tidsplan**	**Ansvarlig**
Opdag årsagen til fejlen	Hurtigst muligt	Hele gruppen
Tag kontakt til backup ansvarlig	Efter identificeret problem	Backup Ansvarlige og hele gruppen
I tilfælde af internet problemer, undersøg andre computere (Og utilgængelig backup fra backup ansvarlige)	Efter identificeret problem	Hele gruppen
I tilfælde af serverproblemer, forsøg at finde koden på den computer der sidst redigeret koden (Og utilgængelig backup fra backup ansvarlige)	Efter identificeret problem	Hele gruppen
I tilfælde af andre problemer, græd og be´ en bøn (Og utilgængelig backup fra backup ansvarlige)	Hopefully never	Hele gruppen

Mangelfuld testning		
Initiativ	**Tidsplan**	**Ansvarlig**
Møde og uddelegering af testopgaver	Hurtigst muligt	Testansvarlige
Opfølgning af test	Efter uddelegering af testopgaver	Testansvarlige

Navneord / Udsagnsord liste

For at vi kunne udarbejde vores domæne model lavede vi vores liste med både navneord såvel som udsagnsord ud fra systemkravene, som er beskrevet i opgaveformuleringen. Vi brugte disse til at danne vores domænemodel og use-case diagram.

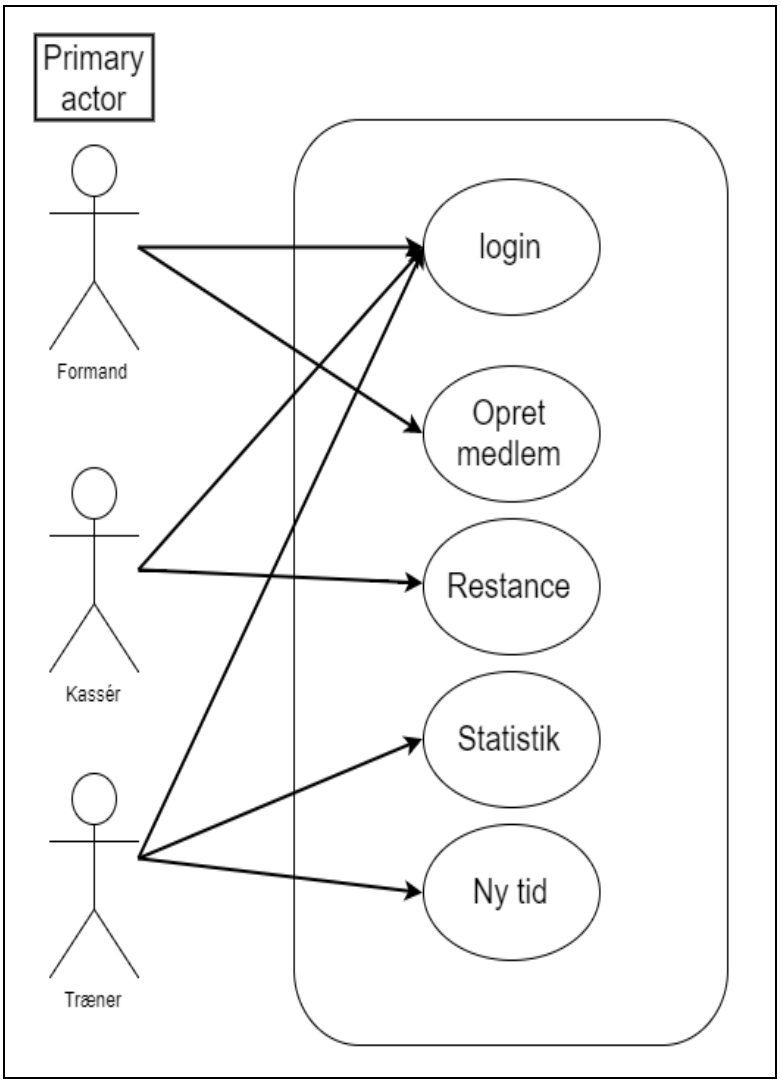
Navneord/Variabler:

Navn
Alder
Aktivitetsform
Junior
Senior
Konkurrence
Motionist
Kontingent
Formand
Kassere
Træner
Disciplin
Træningsresultater
Stævneresultater
Oversigt
Oplysning
Medlem
Rabat
Tid

Udsagnsord/Metoder:

Indmelde
Registrere
Udtage
Vise
Deltage
Styre
Skylde (restance)

Use-Case Diagram



Vores use-case diagram version 2.0 viser at vi har valgt at have 3 primary actors som er: **formand**, **kassér**, og **træner**. dette har vi valgt fordi vi i vores opgave formulering har læst det som at hver titel skal kunne tilgå sin egen menu i programmet med hver deres funktioner.

Use-Cases

* = Ikke til stede i kode (ønsketænkning)

Title: Login
Primary Actor: Formand/kasserer/træner
Main Success Scenario:

1. Bruger logger ind med personligt kodeord.
2. Bruger får adgang til 1 af 3 forskellige menuer alt efter hvilken stilling personen har.

Alternative Flow (Extensions):

***Til enhver tid, hvis systemet fejler:**

1. Vis fejlmeddelelse.
2. Reset system.
3. Send crash data til udviklere.

***Hvis bruger glemmer login:**

1. Bruger vælger glemt kodeord.
2. Bruger indtaster email.
3. Bruger modtager kodeord.

Her er vores casual Use case for “login”. Vores alternative flow er dog kun ønsketænkning og er ikke med i koden. igen er main success scenario taget udfra den måde vi har læst opgaveformuleringen på og det der gav mest mening i forhold til hvad svømmeklubben ville have.

Title: Opret medlem

Primary Actor: Formand

Main Success Scenario:

1. Bruger vælger at oprette medlem.
2. Bruger indtaster div. info omkring medlem.
3. Medlem oprettes og tilføres.

Alternative Flow (Extensions):

***Til enhver tid, hvis systemet fejler:**

1. Vis fejlmeddelelse.
2. Reset system.
3. Send crash data til udviklere.

Hvis medlem er Konkurrencesvømmer:

1. Medlem tildeles en træner.
2. Aktive discipliner registreres.

Title: Restance

Primary Actor: Kasserer

Main Success Scenario:

1. Bruger vælger at se hvem der er i restance.
2. Bruger indkræver betaling.

Alternative Flow (Extensions):

***Til enhver tid, hvis systemet fejler:**

1. Vis fejlmeddelelse.
2. Reset system.
3. Send crash data til udviklere.

***Medlem kan ikke betale:**

1. Bruger retter medlem status.
2. Bruger fjerner medlem fra system.

Title: Statistik

Primary Actor: Træner

Main Success Scenario:

1. Bruger vælger trænings eller konkurrence statistik.
2. Bruger vælger disciplin.
3. Bruger kan se de 5 hurtigste tider i pågældende disciplin.

Alternative Flow (Extensions):

***Til enhver tid, hvis systemet fejler:**

1. Vis fejlmeddelelse.
2. Reset system.
3. Send crash data til udviklere.

Title: Ny tid

Primary Actor: Træner

Main Success Scenario:

1. Bruger vælger ny tid.
2. bruger vælger konkurrence.
3. Bruger indtaster disciplin.
4. Bruger indtaster navn og tid.

Alternative Flow (Extensions):

***Til enhver tid, hvis systemet fejler:**

1. Vis fejlmeddelelse.
2. Reset system.
3. Send crash data til udviklere.

Hvis bruger vælger konkurrence

1. Bruger vælger disciplin.
2. bruger vælger konkurrence
3. Bruger indtaster navn, tid, dato, stævne og placering for medlem

Vi endte med 5 use-cases i alt. et “login” til div. primary actors. Dernæst hver sin funktion dertil og til sidst “ny tid” grunden til at vi delte træneres funktioner i 2 use-cases, er at der er fundamentalt forskel på om træneren ønsker at se “statistik” eller træneren vil oprette en ny tid.

SSD

I vores SSD charts havde vi startet med at tage udgangspunkt i vores første udkast af Use Cases og planlægge dem til at matche så vores SSD’er havde fået struktur.

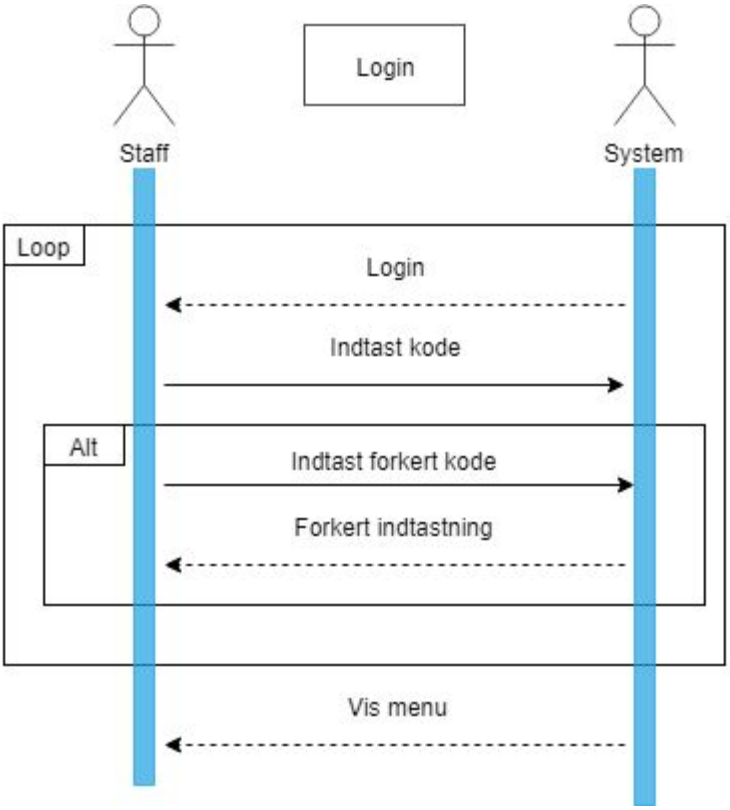
Efterfølgende gik vi igang med den færdige version, og reducerede vores SSD’er så der ikke længere var brug for en “Ret medlem”. Vi ændrede også på vores “**Statistik**” hvor den før havde “**Konkurrence statistik**” og “**Træningsstatistik**”, for at det vil reducere vores programmering i det at dette kunne gøres i samme kode.

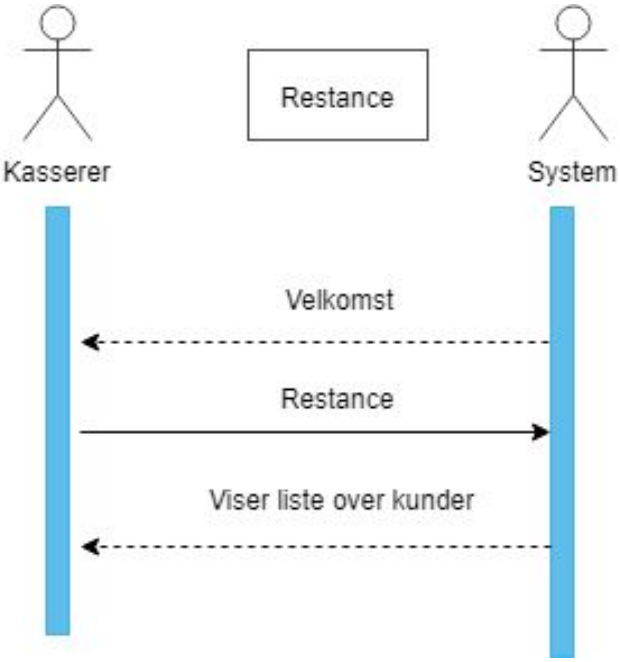
Vores “**Opret medlem**” SSD fik vi også udformet mere til hvad dens fulde funktion skulle indebære, hvilket heriblandt ville fra starten af bede om kundens informationer om de er motionist eller konkurrencesvømmer.

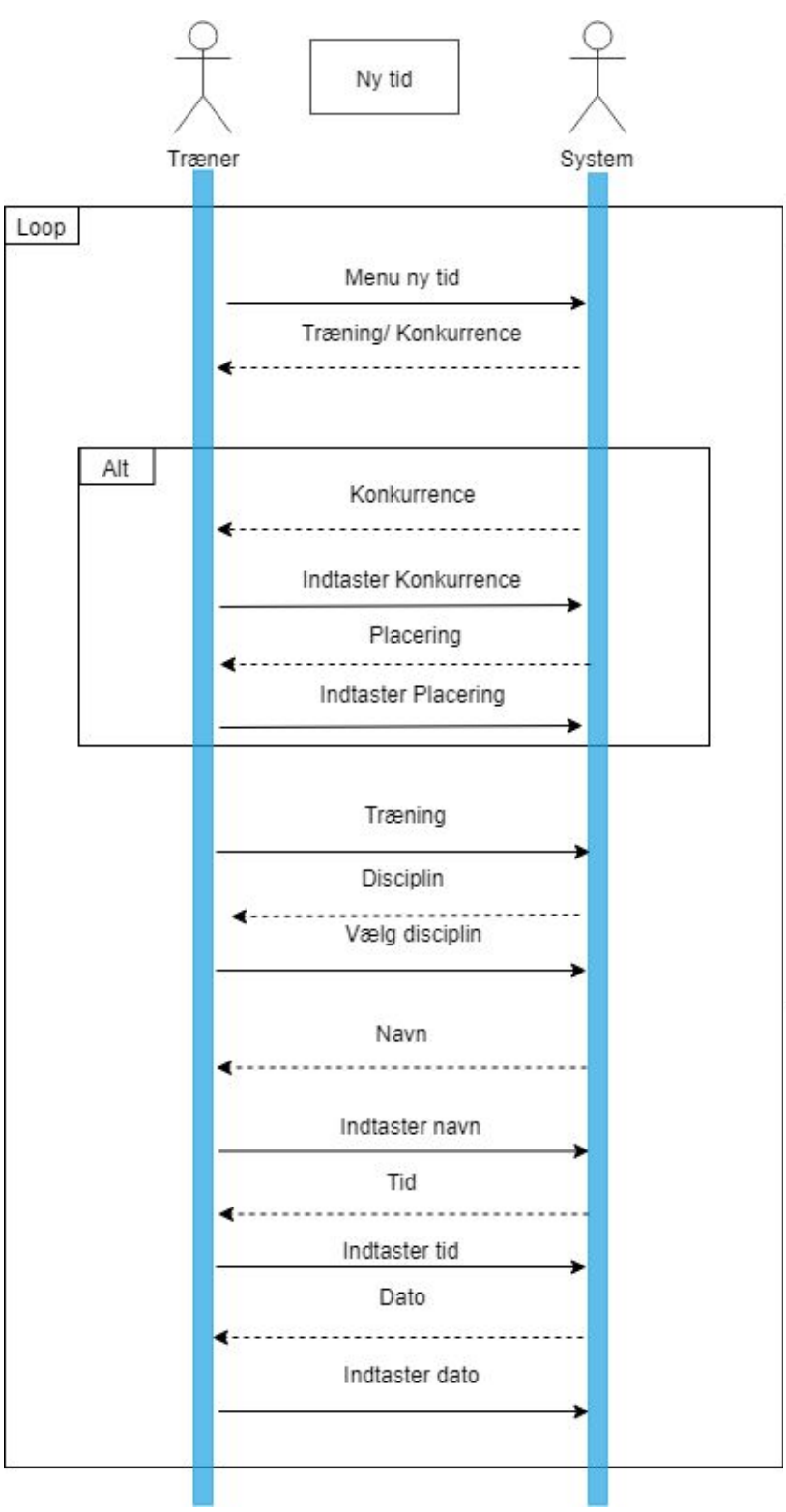
Vi tilføj en ny SSD kaldet “**Ny tid**” for de tider der ville blive registreret indenfor diverse svømmediscipliner.

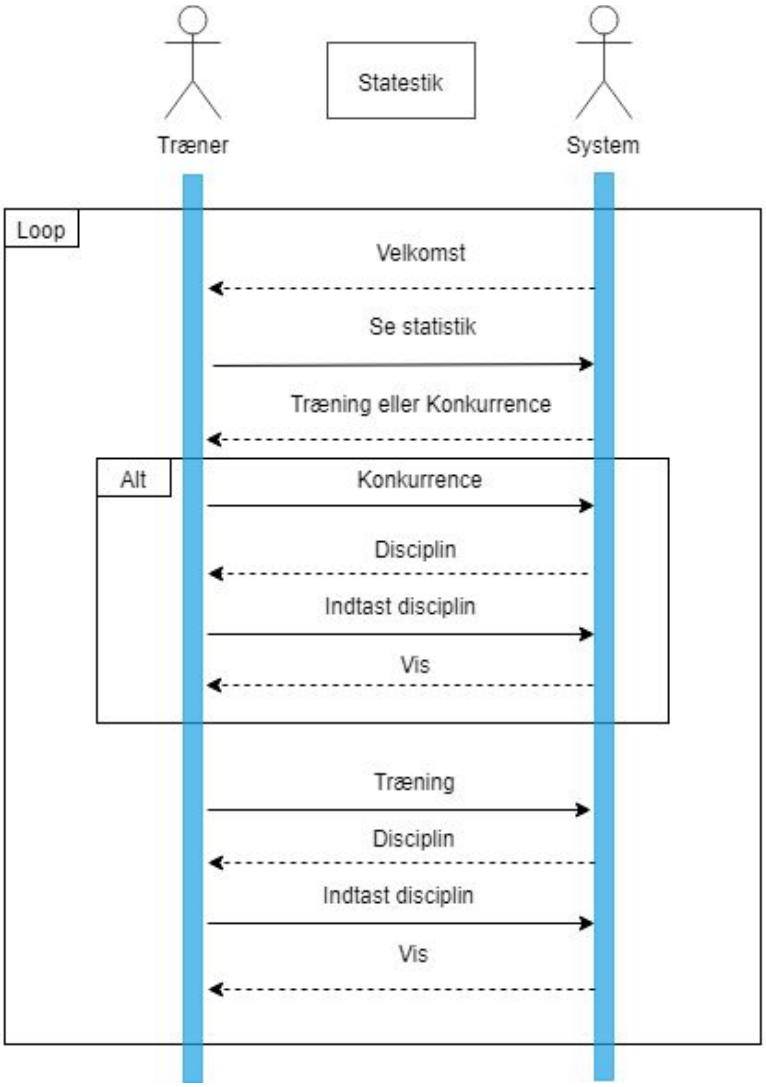
I den færdiggjorte SSD “**Ny tid**” er selve processen for brugeren vist, hvordan system og brugeren snakker sammen. Diagrammet er opsat til at forklare trænerens ønsker at indskrive en ny tid som input, og systemet spørger efterfølgende om det er for træning eller konkurrence. Default vil den gå til træning hvor man bliver bedt om at indskrive hvilken disciplin og svømmer det er, samt til sidst at udfylde med tid og dato.

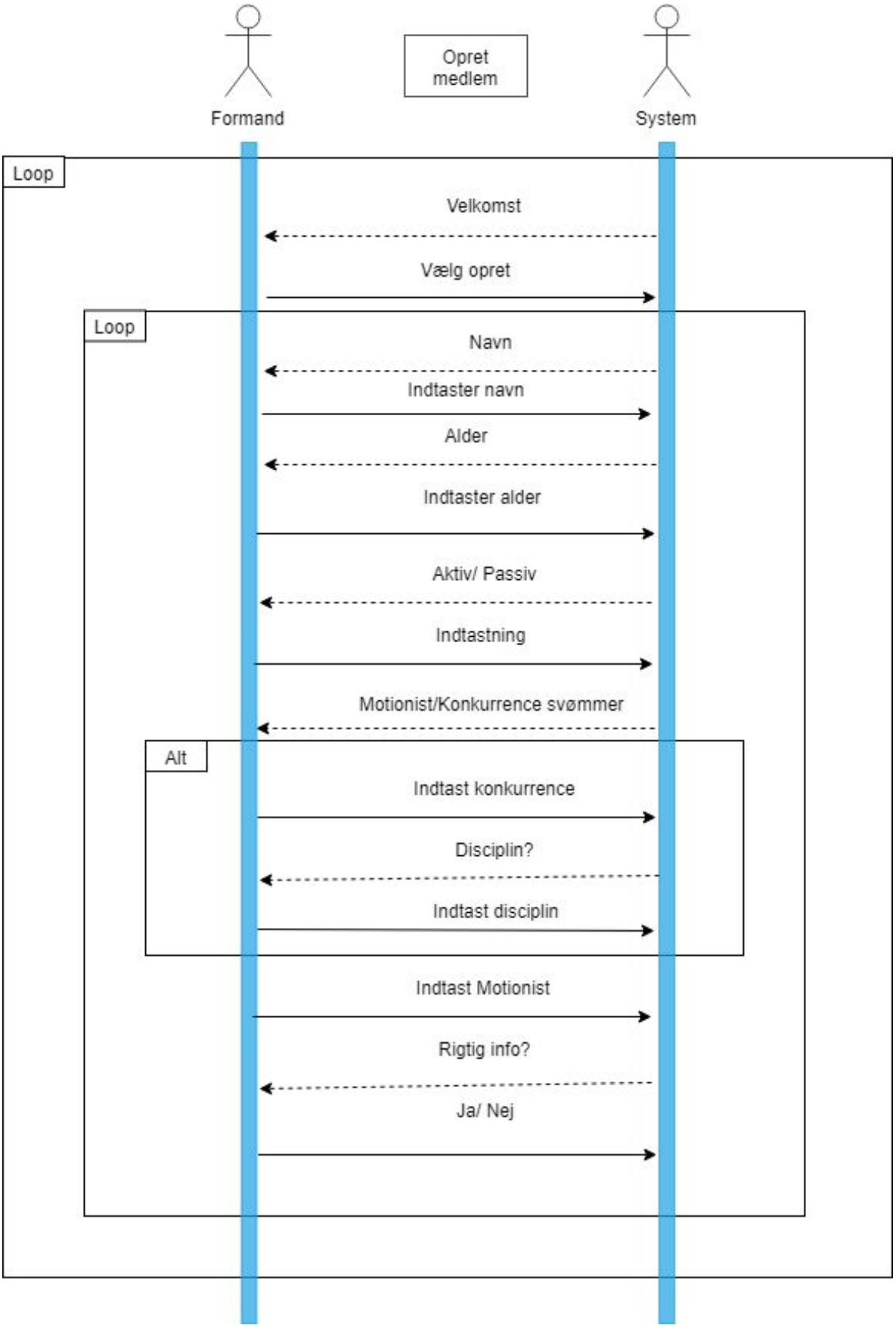
Alternativt hvis man vælger at det ikke er træning, men konkurrence vil den også spørge omkring konkurrencen og hvilken placering svømmeren endte på.


Login
Staff
System
Loop
Login
Indtast kode
Alt
Indtast forkert kode
Forkert indtastning
Vis menu

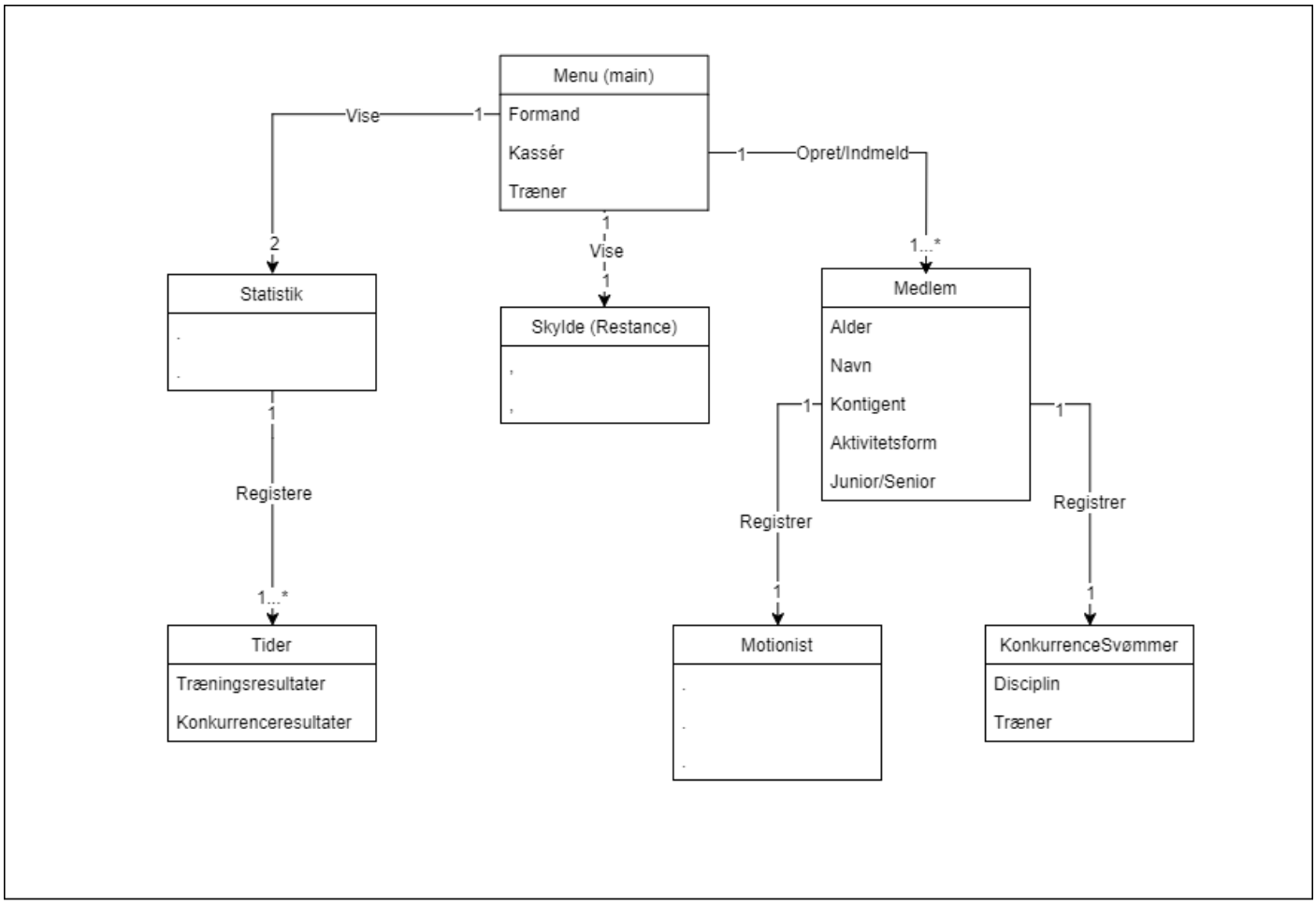

Restance
Kasserer
System
Velkomst
Restance
Viser liste over kunder


Ny tid
Træner
System
Loop
Menu ny tid
Træning/ Konkurrence
Alt
Konkurrence
Indtaster Konkurrence
Placering
Indtaster Placering
Træning
Disciplin
Vælg disciplin
Navn
Indtaster navn
Tid
Indtaster tid
Dato
Indtaster dato


Statestik
Træner
System
Loop
Velkomst
Se statistik
Træning eller Konkurrence
Alt
Konkurrence
Disciplin
Indtast disciplin
Vis
Træning
Disciplin
Indtast disciplin
Vis


Opret medlem
Formand
System
Loop
Velkomst
Vælg opret
Loop
Navn
Indtaster navn
Alder
Indtaster alder
Aktiv/ Passiv
Indtastning
Motionist/Konkurrence svømmer
Alt
Indtast konkurrence
Disciplin?
Indtast disciplin
Indtast Motionist
Rigtig info?
Ja/ Nej

Domæne Model



I vores domænemodel havde vi taget udgangspunkt i vores navne og udsagnsord liste og kom frem til den generelle struktur i vores første udkast. Vi havde her sat **Medlem** som vores primær klasse hvor **Junior** / **Senior** ville blive opdelt til hver deres klasse, så vi kunne skelne dem fra hinanden, og give **Senior** en attribut, som endegyldigt var unik for for den klasse.

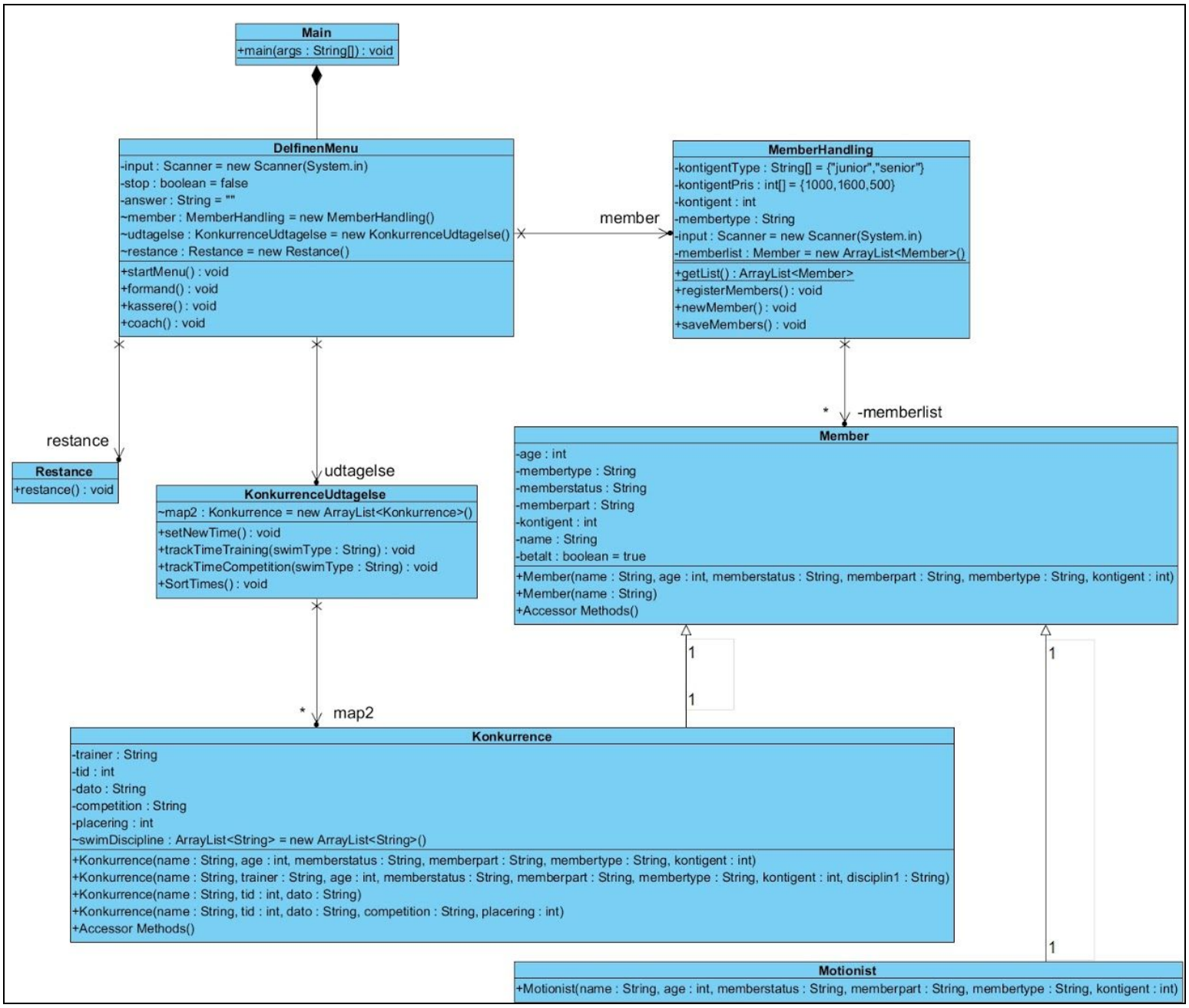
Begge klasser ville efterfølgende blive sat videre til **Konkurrencesvømmer** for at kunne inddele dem i disciplin, tider og placering. Dette ville gå videre til at blive sendt over i **Statistik** som træneren ville kunne gøre brug af, som kunne blive vist fra **Menu.**

I vores færdige version havde vi lavet få ændringer og skåret lidt fra, hvor vi efter **Medlem** direkte efter vil kunne inddele tilmelder i enten **Motionist** eller **Konkurrencesvømmer.**
I **Menu** havde vi givet den 3 attributter til de forskellige ansvarlige der kommer til at skulle benytte sig af programmet, **Formand**, **Kassér** og **Træner**. Hver især kan de tilgå forskellige funktioner, hvor et eksempel ville være at kasséren kan tilgå restance kunder for en liste, og **Træneren** kan indskrive nye tider og se statistik.

Den færdige domænemodel beskriver hvordan vi ser programmets umiddelbare opbygning.
De konceptuelle klasser viser opdelingen af programmet som vi har forstået det. En main der indeholder de 3 brugere som primært har tilgang til systemet, hver af dem skal bruge de forskellige funktioner som kommer fra main. Formanden, skal kunne oprette et medlem af typen motionist eller konkurrence, der begge arver fra superklassen member.

Kasseren skal kunne se om folk er i restance via restance klassen. Til sidst vil træneren for konkurrence svømmerne kunne indtaste nye trænings og konkurrence tider, for at kunne se hvor hurtige hans svømmere er i forhold til hinanden.

Klassediagram



Dette klassediagram er vores færdige version i forhold til hvordan koden endte med at se ud. Hvis vi starter fra toppen har vi vores "**Main**" som kører koden og det er sådan set også det eneste den gør fordi vi valgte at have en klasse kaldet "**DelfinenMenu**" som står for håndteringen af div. klasser og

objekt-kald. Det valgte vi at gøre da det er nemmere at holde styr på "**Main**" og undgår for meget fyld i den.

I "**DelfinenMenu**" har vi 4 metoder hvori 3 af dem kun kalder objektet deri og kører metoderne, fra andre klasser. Som vist tidligere i vores use-case diagram tilgår hver actor deres egen menu efter de har logget ind. F.eks. logger man ind som Formand, kører metoden "**formand**" som så kører klassen "**MemberHandling**" deri har vi valgt at oprette vores medlemmer i en array-liste da vi skal gemme/læse oplysninger omkring personen som vi ønsker at oprette. Dette bliver så kaldt i "**Member**" og der har vi valgt at bruge inheritance som gør vores "**Member**" klasse til en super-class og det betyder at dens 2 sup-classes: "**Konkurrence**" og "**Motionist**" arver alle attributter og metoder fra "**Member**" dette gør at man slipper for en masse boilerplate kode og gør det nemmere at rette eller tilføje til de 2 typer af medlemmer der er nu. eller hvis man ønsker at tilføre en tredje type medlem.

Dernæst har vi "**KonkurrenceUdtagelse**" som læser konkurrence array-listen som indeholder informationer på svømmere og deres præsentationer. Vi har valgt at bruge en separat array-liste end den fra vores "**Member**" da de indeholder en anden længde af informationer. Til sidst har vi "**Restance**" indeholder en random funktion fra "math biblioteket" og udvælger 2 tilfældige fra vores "**Member**" array-liste som skylder penge. Ud fra opgavebeskrivelsen, så står der ingen ønsker omkring at kasséren skal kunne rette i det og vi ønskede at vise sådan set kun at vise funktionen. Hvis vi ser på pilene så har vi brugt en compositions pil til vores "**Main**" klasse da resten af programmet ikke ville kunne fungere uden den. udover det bruger vi specialization pile som indikere at "**konkurrence**" og "**Motionist**" arver fra "**Member**".

Derudover har vi multiplicity til som viser mængden af hvad der oprettes af objekter i vores array-lister. Dette vises ved at skrive (1 - 1) hvis forbindelsen er f.eks. en person som kun kan køre en bil. I vores klasse diagram bruger vi (* - memberlist) som betyder at vi opretter et objekt af vores array-liste og forekomsten/størrelsen af det er fra stjerne* (ukendt) til vores størrelse af memberlist.

Kode

Generel idé

Da opgaveformuleringen blev læst, snakkede vi om hvordan koden skulle opbygges.
Vi besluttede at have en overordnet klasse som hed member, hvor 2 typer af members skulle nedarve fra, en motionist og en konkurrence. Disse to member typer skulle adskilles fordi deres interne info var forskellige fra hinanden og ikke skulle håndteres på samme måde.

Der blev også klart udtrykt i opgaven at enkelte personer i delfinen skal kunne håndtere forskellige opgaver i systemet, så derfor var det smart at dele klasser op således at der var en til hvert virkeområde:
Formanden har "MemberHandling", der kan tilføje nye medlemmer og klassen håndterer også indlæsning og udskrivning af medlemmer til de tilhørende data filer.

Dernæst, kasseren der skal kunne se alle medlemmer i restance, så derfor blev der oprettet en Restance klasse som håndterer dette.
Til sidst skal der kunne ses statistikker over de hurtigste tider. Her bestemte vi at siden der var tilknyttede trænere til alle konkurrence svømmere, var det træneren der skulle tilgå denne klasse. Selve statistikken var til at udtage medlemmer til stævner så vi kaldte denne klasse, "KonkurrenceUdtagelse". Den klasse håndterer også tilføjelser af nye tider til konkurrence medlemmer og at kunne udskrive de hurtigste tider efter sortering.

Alle klasserne styres af en hovedmenu klasse, som ikke selv indeholder metoder der håndterer data. Menuen kalder kun fra de andre klasser. Selvfølgelig er der en Main klasse for sig selv, der bare starter menuen.

Ud over disse blev der bestemt der skulle være 2 arrays, et for kontingenttype og et for kontingentpriser, da disse var specificerede. Systemet er sat op på en måde så svømmediscipliner ikke er en bestemt mængde i et array, men kan skrives ind som klubben selv ønsker det, hvis de har en speciel betegnelse for discipliner. Når statistikken skal vises så skal der huskes den disciplin betegnelse som blev givet.

Ved en medlemsoprettelse giver medlemmet selv sine oplysninger til kende til klub formanden. Her angives navn, alder, om man er aktiv eller passiv og om man vil være motionist eller konkurrencesvømmer. Systemet tildeler selv derefter kontingent type og pris samt om man er junior eller senior og om man skal have rabat for at være over 60 år gammel.
Hvis man er konkurrencesvømmer får man også automatisk tildelt en træner.

Main

Main klassen er kun til for at puste liv i programmet og køre vores menu, ved først at lave en static main metode:

```
public static void main(String[] args)throws FileNotFoundException, InterruptedException{
```

Her er der et par exceptions som bliver thrown pga. at vores program ikke kan være sikker på de filer det bruger, eksisterer. Vi har manuelt oprettet disse, og ved derfor de er hvor de skal være. Interruptedexception er fordi vi bruger en timer, vi lover java her at vi ikke bliver interrupted da det eneste tilfælde det ville ske er hvis brugeren selv gør dette.

Main laver derefter et objekt af klassen "DelfinMenu" og eksekvere første menu entry i denne:

```
//Start menu
DelfinenMenu menu = new DelfinenMenu();
menu.startMenu();
```

Menu

Menu klassen håndterer bruger menuen for hhv. formand, kasserer og træner, den har ikke funktions metoder i sig men kalder de andre klassers metoder som skal bruges.

I klassen er der oprettet nogle fields som f.eks answer til at styre menuen, og de input der gives, samt objekter af de klasser som tilgås. (Se kildekode for præcise fields).

Menuen er delt op i metoder, hvor den første styrer de logins til brugerne, og de næste 3 er specifikke undermenuer svarende til det login som gives.

```
//First menu entry
public void startMenu()throws InterruptedException, FileNotFoundException{
```

Det første menupunkt som kan blive set ovenover, indlæser som det første medlemsliste filen ved at køre register members fra "Memberhandling" klassen, så vi er sikker på at alt er læst ind, inden medlemslisten skal bruges og evt. redigeres.
Herefter sker alle redigeringer til members i selve array-listen og dermed foregår de i realtid så indlæsningen kun er nødvendig denne ene gang.

StartMenu, kører også det while loop, som holder programmet kørende. Der kan kun hoppes ud af loopet ved at bruge de exit menupunkter som bliver fremstillet.

Nu bliver brugeren bedt om at logge ind:

```
System.out.println("~ Welkommen til klub Delfinen ~");
System.out.println();
System.out.print("Login: ");
```

Alt efter hvilket login der bliver indtastet, bliver brugeren sendt til specifikke undermenuer, en af de følgende;

```
//Formand specific menu
public void formand()throws InterruptedException,FileNotFoundException{

//Kasser specific menu
public void kassere()throws InterruptedException, FileNotFoundException{

//Coach specific menu
public void coach()throws InterruptedException,FileNotFoundException{
```

Undermenuerne har hver deres sektion som viser hvilke muligheder brugeren har, f.eks.

```
//Formand specific menu
public void formand()throws InterruptedException,FileNotFoundException{
   System.out.println("Velkommen 'Formand'");
   System.out.println("[1] Opret nyt medlem");
   System.out.println("[2] Exit");
```

derefter er det sat op med en if-else som tjekker hvad der er blevet valgt. Valget fra brugeren kalder en metode fra en separat klasse som kan håndtere det som skal håndteres.

```
if(answer.equals("1")){
    member.newMember();
    member.saveMembers();
}
```

De andre menu specifikke kald kan ses i kildekoden.

Alle menupunkter har også en try-catch der håndterer et forkert input. Dvs at i menuerne kan brugeren kun indtaste det forventede input ellers fås der en error message med “wrong input”. Koden for dette kan ses under.

```
}catch(InputMismatchException i){
    System.out.println("'Forkert input'");
    System.out.println("");
}
```

Der er selvfølgelig en try over selve menu koden, og vi tjekker specifikt på input mismatch pga. det er den error der fås når man f.eks forventer en int input men der gives en String.

Member

Member klassen har en masse attributter, som vi bruger til at lave en constructor. Member klassen har getters og setters til at tilgå de forskellige attributter.

```
1  public class Member{
2
3      //Fields
4      private int age;
5      private String membertype;
6      private String memberstatus;
7      private String memberpart;
8      private int kontigent;
9      private String name;
10     private boolean betalt = true;
11
12     //Constructors
13     public Member(String name,int age, String memberstatus, String memberpart,String membertype, int kontigent){
14        this.age = age;
15        this.memberpart = memberpart;
16        this.memberstatus = memberstatus;
17        this.membertype = membertype;
18        this.kontigent = kontigent;
19        this.name = name;
20     }
21
22     public Member(String name){
23        this.name = name;
24     }
25
26     //Getters and Setters
27     public int getAge(){
28        return age;}
29     public String getMemberType(){
30        return membertype;}
31     public String getMemberStatus(){
32        return memberstatus;}
33     public String getMemberPart(){
34        return memberpart;}
35     public int getKontigent(){
36        return kontigent;}
37     public String getName(){
38        return name;}
39     public void setBetalt(boolean betalt){
40        this.betalt = betalt;}
41     public boolean getBetalt(){
42        return betalt;}
43 }
```

Motionist

Motionist klassen extender member klassen og har samme constructor som sin super klasse.

```
public class Motionist extends Member{

   //Constructor
   public Motionist(String name,int age,String memberstatus, String memberpart,String membertype, int
      super(name,age,memberstatus,memberpart,membertype,kontigent);
   }
}
```

Konkurrence

Konkurrence klassen extender også member klassen, men den har tilføjet sine egne attributter og en array-liste. Vi importere java.util.*; for at få array-listers funktionerne.

```
import java.util.*;

public class Konkurrence extends Member{

   //Fields
   private String trainer;
   private int tid;
   private String dato;
   private String competition;
   private int placering;
   ArrayList<String> swimDiscipline = new ArrayList<String>();
```

Vi har igen hentet attributter fra super klassen, men har også lavet klassens egen constructor, hvor vi bruger overloading.

I den ene constructor fra super klassen, hvor vi bruger arraylisten fra tidligere til at tilføje discipliner, har vi brugt et array og et for-loop for at splitte disciplinerne op med et komma. Dette bliver brugt til at splitte disciplinerne op i vores txt fil, for at vi nemmere kan læse dem ind senere.

```
//Constructors
public Konkurrence(String name,int age,String memberstatus, String memberpart,String membertype, int kontigent){
   super(name,age,memberstatus,memberpart,membertype,kontigent);
}

public Konkurrence(String name,String trainer,int age,String memberstatus, String memberpart,String membertype, int kontigent, String disciplin1){
   super(name, age,memberstatus,memberpart,membertype,kontigent);

   String[] elements = disciplin1.split(",");

   for(int i=0 ; i<elements.length;i++){
      swimDiscipline.add(elements[i]);
   }

   this.trainer = trainer;
}

public Konkurrence(String name,int tid, String dato){
   super(name);
   this.tid = tid;
   this.dato = dato;
}

public Konkurrence(String name,int tid, String dato, String competition,int placering){
   super(name);
   this.tid = tid;
   this.dato = dato;
   this.competition = competition;
   this.placering = placering;
}
```

Af getters og setters, har vi meget standard getters og setters til at tilgå de attributter, undtagen vores getDisciplin, hvor vi har et loop til at gå igennem vores array-liste til at hente alle disciplinerne.

```
   //Getter and Setters
   public String getDisciplin(){
      String discipline = "";
      for(int i=0; i<swimDiscipline.size(); i++){
         discipline += swimDiscipline.get(i)+",";
      }
      return discipline;
      }

   public String getTrainer(){
      return trainer;}
   public int getTid(){
      return tid;}
   public String getCompetition(){
      return competition;}
   public int getPlacering(){
      return placering;}
}
```

MemberHandling

MemberHandling har oprettet 2 arrays, 1 med 2 strings, junior og senior, som vi bruger senere til at bestemme hvilket medlemskab medlemmet skal have. Array nummer 2 har priserne for medlemskaber som vi tildeler senere. Der er oprettet en array-liste til at tilføje nye medlemmer, da vi ikke ved hvor mange medlemmer de har og reelt kommer til at have. Til sidst opretter vi også et scanner objekt, til at tage input fra konsollen.

```
import java.util.*;
import java.io.*;

public class MemberHandling{

   //Fields
   private String kontigentType[] = {"junior","senior"};
   private int kontigentPris[] = {1000,1600,500};
   private static ArrayList<Member> memberlist = new ArrayList<Member>();
   private int kontigent;
   private String membertype;
   private Scanner input = new Scanner(System.in);
```

Vi har så tilføjet en getter til ”memberlist” sådan at andre klasser og metoder kan tilgå den.

```
//Getter for memberlist
public static ArrayList<Member> getList(){
   return (memberlist);
   }
```

Vi har så oprettet en metode til at hente de forskellige medlemmer ind, hvor vi tilgår ”motionmember.txt” og ”konkurrencemember.txt” via. 2 separate scanners.
Dertil bruger vi så et while loop på begge scanners, til at hente dataen fra dem.

```
public void registerMembers()throws FileNotFoundException{
   Scanner inputmotion = new Scanner(new File("motionmember.txt"));
   Scanner inputkonkurrence = new Scanner(new File("konkurrencemember.txt"));
   while (inputkonkurrence.hasNext()) {
      String name = inputkonkurrence.next();
      int age = inputkonkurrence.nextInt();
      String type = inputkonkurrence.next();
      String status = inputkonkurrence.next();
      String part = inputkonkurrence.next();
      int kontigent = inputkonkurrence.nextInt();
      String trainer = inputkonkurrence.next();
      String disciplin = inputkonkurrence.next();
      memberlist.add(new Konkurrence(name,trainer,age,status,part,type,kontigent,disciplin));
   }
   while (inputmotion.hasNext()) {
      String name = inputmotion.next();
      int age = inputmotion.nextInt();
      String type = inputmotion.next();
      String status = inputmotion.next();
      String part = inputmotion.next();
      int kontigent = inputmotion.nextInt();
      memberlist.add(new Motionist(name,age,status,part,type,kontigent));
   }
}
```

Til at oprette medlemmer har vi lavet en metode dertil, vi har brugt en try/catch syntax til at forhindre fejl, som f.eks. “inputmismatchexception”.

I metoden bliver du bedt om at indtaste informationer omkring dig selv, hvorefter programmet automatisk tildeler dig den rigtige kontingent baseret på din alder.

```
//Add new member
public void newMember(){
   try{
      System.out.println("");
      System.out.print("navn: ");
      String name = input.next();
      System.out.print("alder: ");
      int age = input.nextInt();
      System.out.print("aktiv/passiv: ");
      String status = input.next();
      System.out.print("motionist/konkurrence: ");
      String part = input.next();
      if(status.equals("passiv")){
         kontigent = kontigentPris[2];
         if(age <= 18){
            membertype = kontigentType[0];
         }else{
            membertype = kontigentType[1];
         }
      }else if (age <= 18 && status.equals("aktiv")){
         kontigent = kontigentPris[0];
         membertype = kontigentType[0];
      }else if ((age >= 18 & age <= 59) && status.equals("aktiv")){
         kontigent = kontigentPris[1];
         membertype =  kontigentType[1];
      }else if (age >= 60 && status.equals("aktiv")){
         kontigent = kontigentPris[1]-400;
         membertype = kontigentType[1];
      }
      }else if (age <= 18 && status.equals("aktiv")){
         kontigent = kontigentPris[0];
         membertype = kontigentType[0];
      }else if ((age >= 18 & age <= 59) && status.equals("aktiv")){
         kontigent = kontigentPris[1];
         membertype =  kontigentType[1];
      }else if (age >= 60 && status.equals("aktiv")){
         kontigent = kontigentPris[1]-400;
         membertype = kontigentType[1];
      }
      if(part.equals("motionist")){
         System.out.println("Navn: " + name + " " + "Alder: " + age + " " + "Status: " + status + "
         System.out.print("Er det de rigtge informationer?: ");
         String answer;
         answer = input.next();
            if(answer.equals("ja")){
               memberlist.add(new Motionist(name,age,status,part,membertype,kontigent));
            }else{
               newMember();
```

```
            }
        }else if(part.equals("konkurrence")){

            String trainer;

            if(age < 18 ){
               trainer = "Bear_G.";}
            else{trainer = "Arnold_S.";}
            System.out.println("hvor mange svomme discipliner vi har: (eks. Crawl,Butterfly,Rygsvomning)");
            System.out.print("Skriv discipliner kun adskilt af komma: ");
            String antal_disciplin = input.next();
            memberlist.add(new Konkurrence(name,trainer,age,status,part,membertype,kontigent,antal_disciplin));
        }
    }catch(Exception i){
        System.out.println("Forkert input");
    }
```

Til sidst har vi en metode til at gemme den information vi har om de forskellige members, som er blevet tilføjet via. et for loop.

```
//Save memberlist in output file
public void saveMembers()throws FileNotFoundException{
    PrintStream output = new PrintStream("motionmember.txt");
    PrintStream output1 = new PrintStream("konkurrencemember.txt");
    for(Member m : memberlist){
        if (m instanceof Motionist){
            output.println(m.getName() + " " + m.getAge() + " " + m.getMemberType() + " " + m.getMemberStatus() + "
        }else if (m instanceof Konkurrence){
            output1.println(m.getName() + " " + m.getAge() + " " + m.getMemberType() + " " + m.getMemberStatus() +
        }
    }
}
}
```

KonkurrenceUdtagelse

Denne klasse indeholder og håndterer forskellige tider som er sat enten til træning eller til et stævne. Til at starte med oprettes der en array-liste som hedder map2, der ligge klar til at indeholde den data der nu bestemmes i menuen.

```
ArrayList<Konkurrence> map2 = new ArrayList<Konkurrence>();
```

En ny hurtigere tid kan indtastes ved at vælge menupunktet “indtast ny tid”, og derefter bestemme hvilken disciplin og om det er en trænings eller konkurrence tid.

Dette er vigtig da trænings og konkurrence tiderne ikke gemmer den samme mængde information og også ligger gemt i 2 forskellige data filer.
Dvs. der indlæses 2 scannere for de 2 filer og alt efter hvad der bliver valgt vil outputtet blive hhv til den ene eller den anden fil som set herunder:

```
File file = new File("trainingtider.txt");
File file1 = new File("konkurrencetider.txt");
PrintStream output = new PrintStream(new FileOutputStream(file,true));
PrintStream output1 = new PrintStream(new FileOutputStream(file1,true));
```

For træning:

```
output.println(disciplin + " " + name + " " + tid + " " + date);
```

For konkurrence:

```
output1.println(disciplin + " " + name + " " + tid + " " + date + " " + cup + " " + placering);
```

Denne funktion som hedder setNewTime(), er indrammet i en try catch for at fange evt. input fejl.

De næste to metoder styrer udskrivningen af de hurtigste svømme tide for enten træning eller konkurrence, og hedder:

```
public void trackTimeCompetition(String swimType)throws FileNotFoundException{
public void trackTimeTraining(String swimType)throws FileNotFoundException{
```

Den eneste forskel på disse to er indlæsningen af den tilhørende fil, da konkurrence har ekstra info om stævne og placering i stævnet. Print funktionen er også forskellig på den måde at for konkurrence svømmerne vil der kun blive set de 5 hurtigste tider, hvor en svømmer kun forekommer en gang, for at der kan blive set på hvem der skal udtages til stævner i fremtiden.
Træningstider derimod skal der ses den hurtigste tid for hver person der har sat en tid i en given disciplin.

Det er vigtigt her at for vi kan vælge at se statistik for både træning og konkurrence i samme omgang i programmet, at man clearer array-listen da den bliver brugt i begge tilfælde. Ellers ville alt data blive lagt i forlængelse af hinanden alle de gange man valgte menupunktet for at se statistikkerne.

```
map2.clear();
```

Indlæsning af dataen fra filerne blev valgt til at blive indlæst som et objekt af konkurrence svømmere (klassen som er forklaret tidligere). Da sorteringen af tider kan gøre på en smart måde. Selve indlæsningen er standard og forløber således:

```
Scanner input = new Scanner(new File("trainingtider.txt"));
while (input.hasNext()) {
   String word = input.next();
   if (word.equals(swimType)){
      String name = input.next();
      int tid = input.nextInt();
      String dato = input.next();
      map2.add(new Konkurrence(name,tid,dato));
   }else{
      input.nextLine();
   }
```

Den sidste else med en input.nextLine(), eksisterer fordi at alle tider skal ses i henhold til en specifik disciplin så vi søger gennem data filen for den første String på hver linje og hvis den ikke er equal med det disciplin navn vi vil se, bliver den ikke indlæst.
Derefter tilføjes de linjer som er vigtige til listen.

Nu eksisterer alle de tider der skal ses på i listen “map2” ,så skal de sorteres efter de hurtigste, og der skal sørges for intet navne forekommer 2 gange, kun den hurtigste tid for en person.
Her kaldes metoden SortTimes().
Denne metode tager listen og går igennem alle positioner, trækker navnet ud, ser om det er equal med andre i listen og hvis det er kigger loopet på tiden i de to positioner og remover den langsomste fra listen:

```
for(int i = 0 ; i < map2.size() ; i++){
   String name = map2.get(i).getName();
   for(int j = i+1 ; j < map2.size() ; j++){
      if(name.equals(map2.get(j).getName())){
         int tidi = map2.get(i).getTid();
         int tidj = map2.get(j).getTid();
         if(tidi < tidj){
            map2.remove(j);
            i=i-1;
         }else{
            map2.remove(i);
            i=i-1;
         }
   }
```

Så til sidst er der en liste hvor en person kun forekommer en enkelt gang (med sin hurtigste tid). Grunden til at hver gang der removes en entry i listen, at i tælleren bliver sat en tilbage er fordi array-lister er dynamiske og hvis vi ikke gik en tilbage og kiggede efter hvert remove ville elementer blive sprunget over.

Nu skal listen sorteres sådan at den hurtigste person skal ligge på de første element og nedefter skal tiderne blive langsommere.
Der gøres på samme måde med 2 for-loops som ovenover dog bruges kommandoen swap til at positionere de forskellige personer:

```
for(int j = 0 ; j<map2.size() ; j++){
   for(int i = 0; i<map2.size()-1 ; i++){

      int tid = map2.get(i).getTid();
      int tid2 = map2.get(i+1).getTid();

      if(tid2<tid){
         Collections.swap(map2, i+1, i);
      }
   }
}
```

Det er en implementering af en klassisk sorteringsmetode (bubble-sort) bare brugt med objekter.
Så man tager det første element og kigger på det næste, hvis nr 2 er mindre en nr1, swap placering, sådan foregår det hele vejen ned gennem listen og det går man så j antal gange.

Så er listen sorteret og der skal bare printes ud i konsollen, som sagt for træningstider printes alle personer, men i konkurrence vil der kun ses de 5 bedste i disciplinen, der er der lavet en if der tjekker størrelsen på listen, og hvis den er over 5, printes der kun de 5 første elementer:

```
if(map2.size() <= 5){

   for(Konkurrence m : map2){
      System.out.println("Navn: "+m.getName() + "\t" + "Tid: "+m.getTid()+" Sekunder");
   }}
else{
   for(int i = 0 ; i <5 ; i++){
      System.out.println("Navn: "+map2.get(i).getName() + "\t" + "Tid: "+map2.get(i).getTid()
   }
}
Sys
```

Restance

I restance klassen som kun kasseren kan tilgå, ligger der en funktion, som i dette tilfælde sætter samt tjekker medlemmer som er i restance.

Da opgaveformuleringen ikke specificerede at der skulle tages betaling elektronisk har vi valgt at lave et åbent system. Dvs at alle medlemmer har en boolean som hedder betalt der tjekker for om værdien er true.
For at vise at vi kan håndtere dette og at det virker har vi lavet en funktion der tager 2 random personer fra medlemslisten og sætter betalt til false, under er vist hvordan dette er gjort:

```
int rand = (int)(Math.random()*list.size());
list.get(rand).setBetalt(false);
```

Hvor rand er en random tal variables fra 1-størrelsen af listen så vi altid får et tilfældigt element(medlem) indenfor listen.
List er medlemslisten der er blevet indlæst fra den gemte fil. Denne er blevet indlæst fra klassen memberhandling ved at oprette en ny liste og bruge en getter fra memberhandling:

```
ArrayList<Member> list = new MemberHandling().getList();
```

Derefter printes alle medlemmer hvor værdien er false.

Som et sikkerhedsnet sætter vi alle medlemmers betalt til true igen efter dette.
I tilfældet af at der bliver indført elektronisk betaling skal dette fjernes og når medlemslisten gemmes i sin fil skal betalt variablen skrives med ned i denne.

Refleksion/konklusion

Der er selvfølgelig mangler i vores system program både pga. dette skulle anses som en prototype fordi tiden til at lave programmet var kort, dog også pga. hvad kravene egentlig var til selve systemet.
Der bliver gemt en masse data om medlemmer, som aldrig bliver vist eller tilgået i denne version, det kunne have været ideelt at have en “rediger medlem”, eller måske, at vise et medlem så man kunne se deres info.
I den virkelige verden er dette en process vi vil tage med kunden undervejs så de små ting kunne blive løst og sikre begge parter har samme mål. Man kunne godt have lavet disse funktioner, men vi vurderede at siden kunden ikke har specificerede disse krav, forventede vi ikke at vil blive “betalt” for dette, og derfor ville vi kun lavede hvad der var krav på fra vores kunde.

Der er et par småting som kunne være blevet lavet lidt bedre end det er på det nuværende system. F.eks hvis man opretter et nyt medlem skal man angive om medlemmet er aktivt eller passivt pga. kontingent prisen, men man kan faktisk svare alt i en string og så vil prisen ikke blive tildelt rigtigt, dette skal laves så det er muligt kun at besvare de to “rigtige” svar.

Hvis man vil se en statistik over tider kan man godt søge efter tider i en disciplin hvor der ikke er blevet sat nogle, i det nuværende program skriver funktionen bare ikke noget ud på skærmen og fortsætter, der kunne ønskes en fejlmeddelelse som siger “ingen tid fundet”.
Samtidig når man lægger en ny tid ind i systemet ønskes en dato for et evt. stævne hvor tiden er blevet sat. Der er på nuværende tidspunkt ingen begrænsning på hvordan datoen bliver skrevet ind på, det er ok nu da den ikke skal bruges, men en begrænsning ville være godt så de altid ville være ens.
Hvis dette projekt var med en kunde i virkeligheden havde vi valgt at mødes igen og indskærpe kravene, det er gået op for os hvor besværligt det egentligt er nu når vi synes disse krav ikke virkede skarpe nok.
Vi føler at der er blevet tænkt over opsætningen af programmet på sådan en måde at hvis der kommer ekstra krav eller ændringer til eksisterende er det åbent og nemt at se hvor disse ændringer skulle indsættes. Det er ikke blevet skrevet/opsat sådan at det kun virker og er smart for de nuværende krav. F.eks restance klassen og motionist nedarvningen der nærmest ikke gør noget, når de står for sig selv, men det er blevet lavet alligevel fordi for forståelse er de smarte og det giver mening hvis nu en ny funktion skulle tilføjes hvis ændring i krav med kunden skulle opstå.
Alt i alt har systemet de funktioner som var kravene til det, dét skal kunne, der er et par småting der kan rettes, men ikke noget der ødelægger oplevelsen for brugeren eller ødelægger funktionerne af de nuværende funktioner.
Alle diagrammer er med og viser deres betydning for programmet.
Der har ikke været brug for risiko håndteringen og alle ITO analyser holdte, i forhold til hvad vi regnede med. Selve tidsplanen er blevet overholdt, og endda nok i lidt bedre tid end forventet.

Litteraturliste

https://stackoverflow.com/

https://google.com/

Craig Larman. Applying uml and patterns. Third edition. Prentice Hall

Hans Jørgen Skriver, Peter Storm-Henningsen & Erik Staunstrup(2017). ORGANISATION 6. Udgave. Trojka.

Robert Liguori og Patricia Liguori. Java 8 Pocket Guide. O'REILLY Media

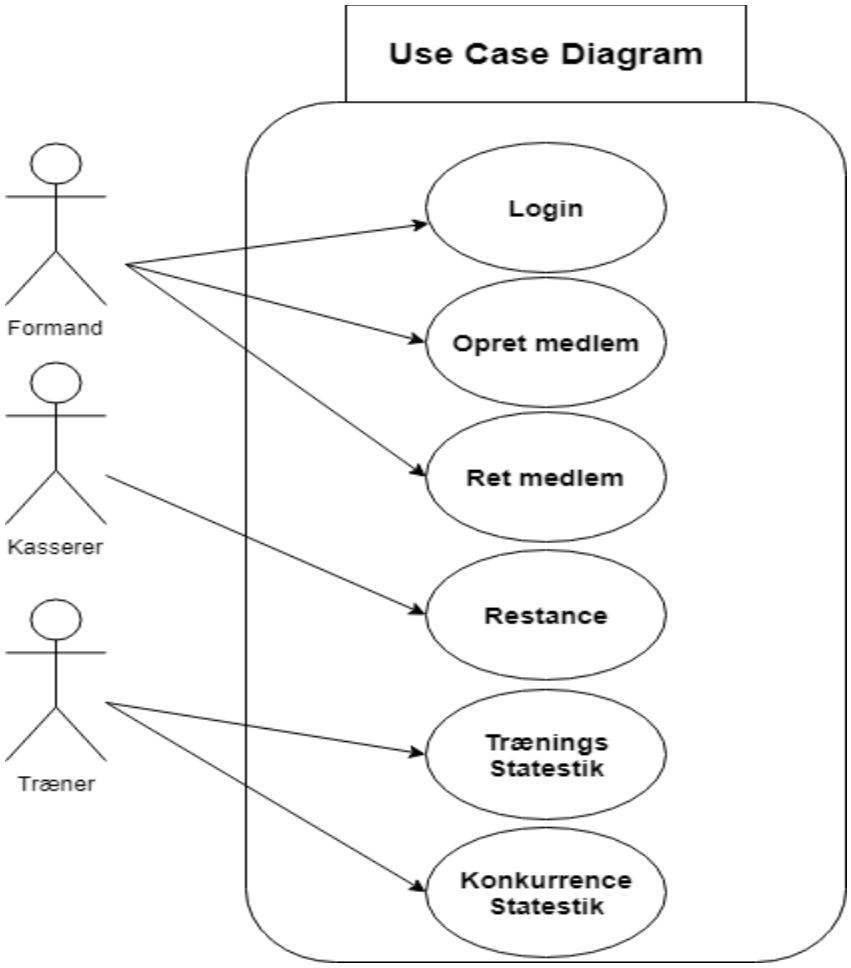
Stuart Reges & Marty Stepp. Building java programs - a back to basics approach. Fourth edition. Pearson

Bilag

Gantt Kort

TASK NAME	START DATE	END DATE	DURATION (WORK DAYS
ITO			
SWOT Analyse 1. version	20/11-2018	22/11-2018	2
SWOT Analyse 2. version	23/11-2018	27/11-2018	3
Risiko Analyse 1. version	20/11-2018	22/11-2018	2
Risiko Analyse 2. version	23/11-2018	27/11-2018	3
Interessant Analyse 1. version	20/11-2018	20/11-2018	2
Interessant Analyse 2. version	23/11-2018	27/11-2018	3
Software Design			
Use Case første version	20/11-2018	20/11-2018	1
Use Case færdiggjort	23/11-2018	23/11-2018	1
Use Case Diagram første version	20/11-2018	20/11-2018	1
Use case Diagram færdiggjort	26/11-2018	26/11-2018	1
Domæne Model første version	20/11-2018	23/11-2018	2
Domæne Model færdiggjort	27/11-2018	28/11-2018	2
Klasse Diagram første version	20/11-2018	23/11-2018	2
Klasse Diagram færdiggjort	27/11-2018	28/11-2018	2
SSD første version	21/11-2018	21/11-2018	1
SSD færdiggjort	26/11-2018	27/11-2018	2
Kode Skrivning	21/11-2018	7/12/2018	13
Noun / Verb	22/11-2018	23/11-2018	2
Andet			
PowerPoint	5/12/2018	7/12/2018	3
Rapport	22/11-2018	7/12/2018	12
Brainstorm	20/11-2018	20/11-2018	1
Gennemtjekning af alle filer for mangler og rettel	3/12/2018	10/12/2018	6
Aflevering af opgave	11/12/2018	11/12/2018	1

Use Case Diagram ver. 1



Use cases ver. 1

Title: Login

Primary Actor: Formand/kasserer/træner

Secondary actor: Admin

Main Success Scenario:

1. Bruger logger ind med personligt kodeord.
2. Bruger får adgang til 1 af 3 forskellige menuer alt efter hvilken stilling personen har.

Alternative Flow (Extensions):

Til enhver tid, hvis systemet fejler:

1. Vis fejlmeddelelse.
2. Reset system.
3. Send crash data til udviklere.

Hvis bruger glemmer login:

1. Bruger vælger glemt kodeord.
2. Bruger indtaster email.
3. Bruger modtager kodeord.

Admin har tilgang til fuld menu:

1. Admin logger ind med admin kodeord.

Title: opret medlem

Primary Actor: Formand

Main Success Scenario:

1. Bruger vælger at oprette medlem.
2. Bruger indtaster div. info omkring medlem.
3. Medlem oprettes og tilføres.

Alternative Flow (Extensions):

Title: Ret medlem

Primary Actor: Formand/Kasserer

Main Success Scenario:

1. Bruger vælger at søge efter medlem i menu.
2. Bruger vælger hvilket medlem der skal redigeres.
3. Bruger retter div. info/status ved medlem.
4. Bruger vælger om rettet info skal gemmes.

Alternative Flow (Extensions):

Hvis bruger vælger ikke at gemme oplysninger:

1. Gemmer ikke ændringer og sender bruger tilbage til startmenu.

Hvis bruger vælger at gemme oplysninger:

1. Gemmer ændringer og sender bruger tilbage til startmenu.

Title: Manglende betaling
Primary Actor: Kasserer
Main Success Scenario:

1. Bruger vælger at se statistik over medlemmer.
2. Bruger ser hvem der skylder kontingent.
3. Bruger indkræver betaling og retter bruger derefter.

Alternative Flow (Extensions):
Til enhver tid, hvis systemet fejler:

1. Vis fejlmeddelelse.
2. Reset system.
3. Send crash data til udviklere.

Medlem kan ikke betale:

1. Bruger retter medlem status.
2. Bruger fjerner medlem fra system.

Title: Træning statistik
Primary Actor: Træner
Main Success Scenario:

1. Bruger vælger træningsstatistik i menu.
2. Bruger vælger disciplin.
3. Bruger kan se alle medlemmers hurtigste tid, pågældende dato.

Alternative Flow (Extensions):
Til enhver tid, hvis systemet fejler:

1. Vis fejlmeddelelse.
2. Reset system.
3. Send crash data til udviklere.

Hvis medlems tid bliver bedre:

1. Bruger vælger medlem.
2. Bruger vælger disciplin.
3. Bruger retter tider og dato.

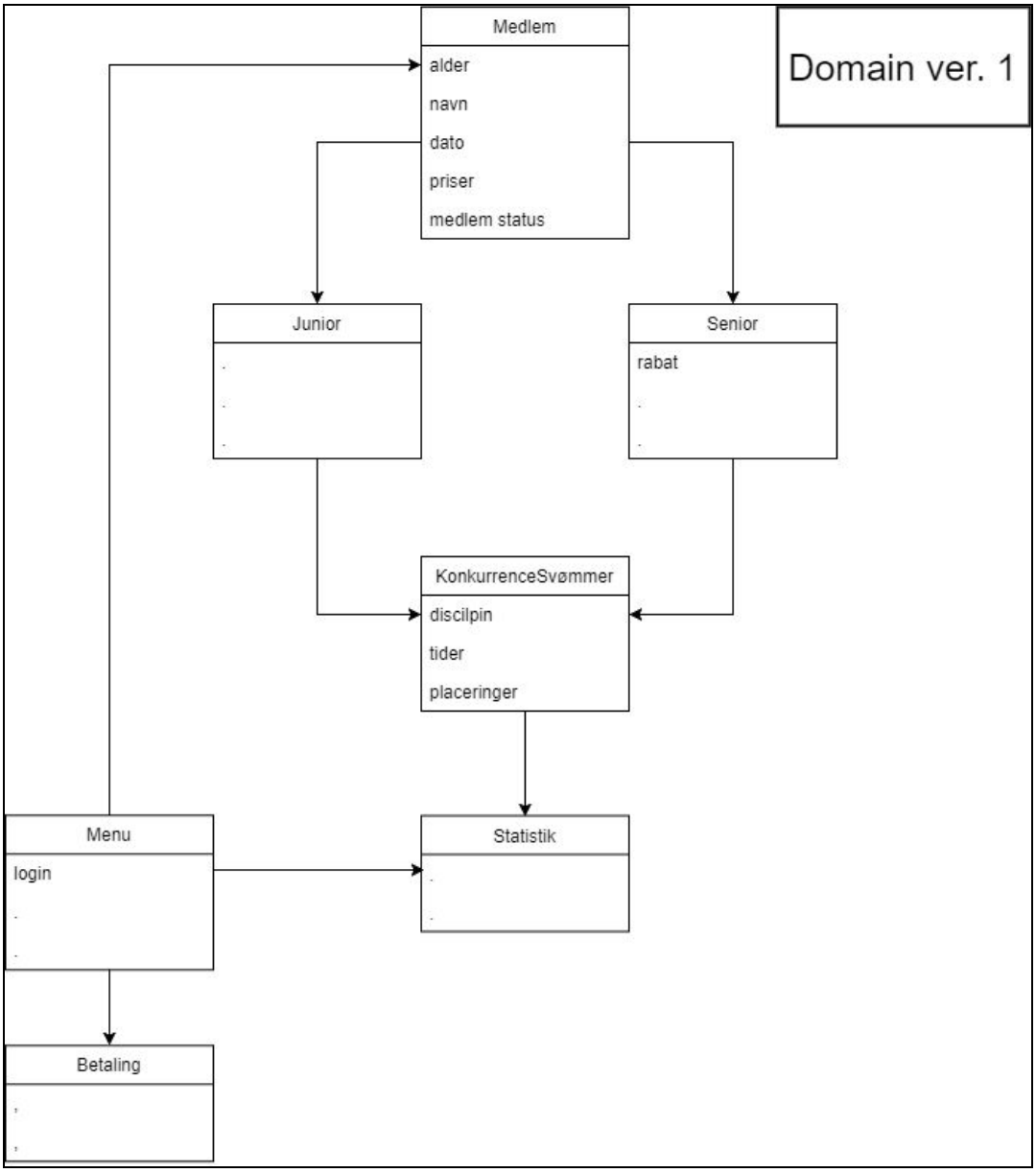
Title: Konkurrence statistik
Primary Actor: Træner
Main Success Scenario:

1. Bruger vælger konkurrence statistik i menu.
2. Bruger vælger disciplin.
3. Bruger kan se medlems hurtigste tid, placering og stævne.

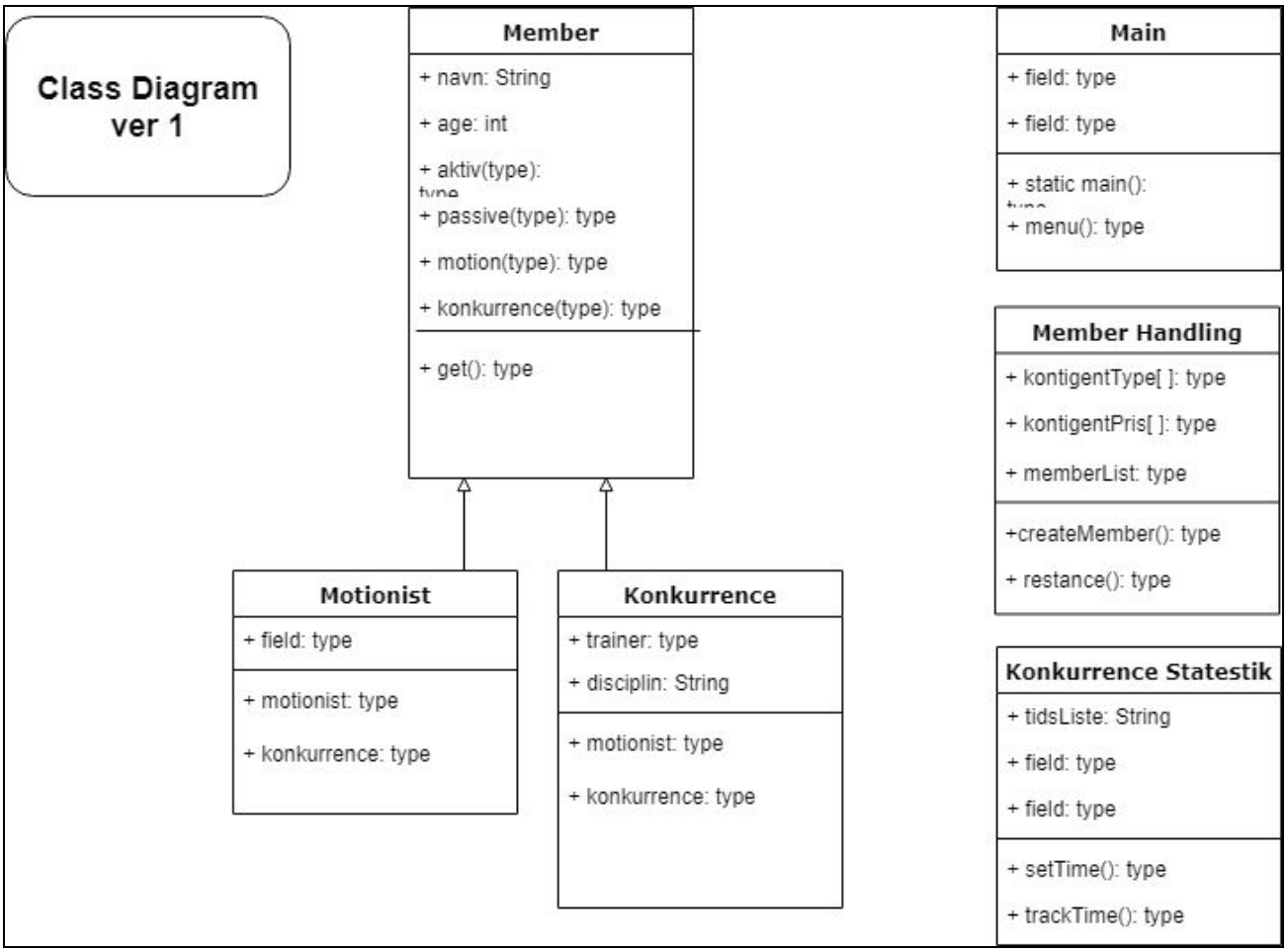
Alternative Flow (Extensions):
Hvis medlems tid bliver bedre:

1. Bruger vælger medlem.
2. Bruger vælger disciplin.
3. Bruger retter tider og dato.

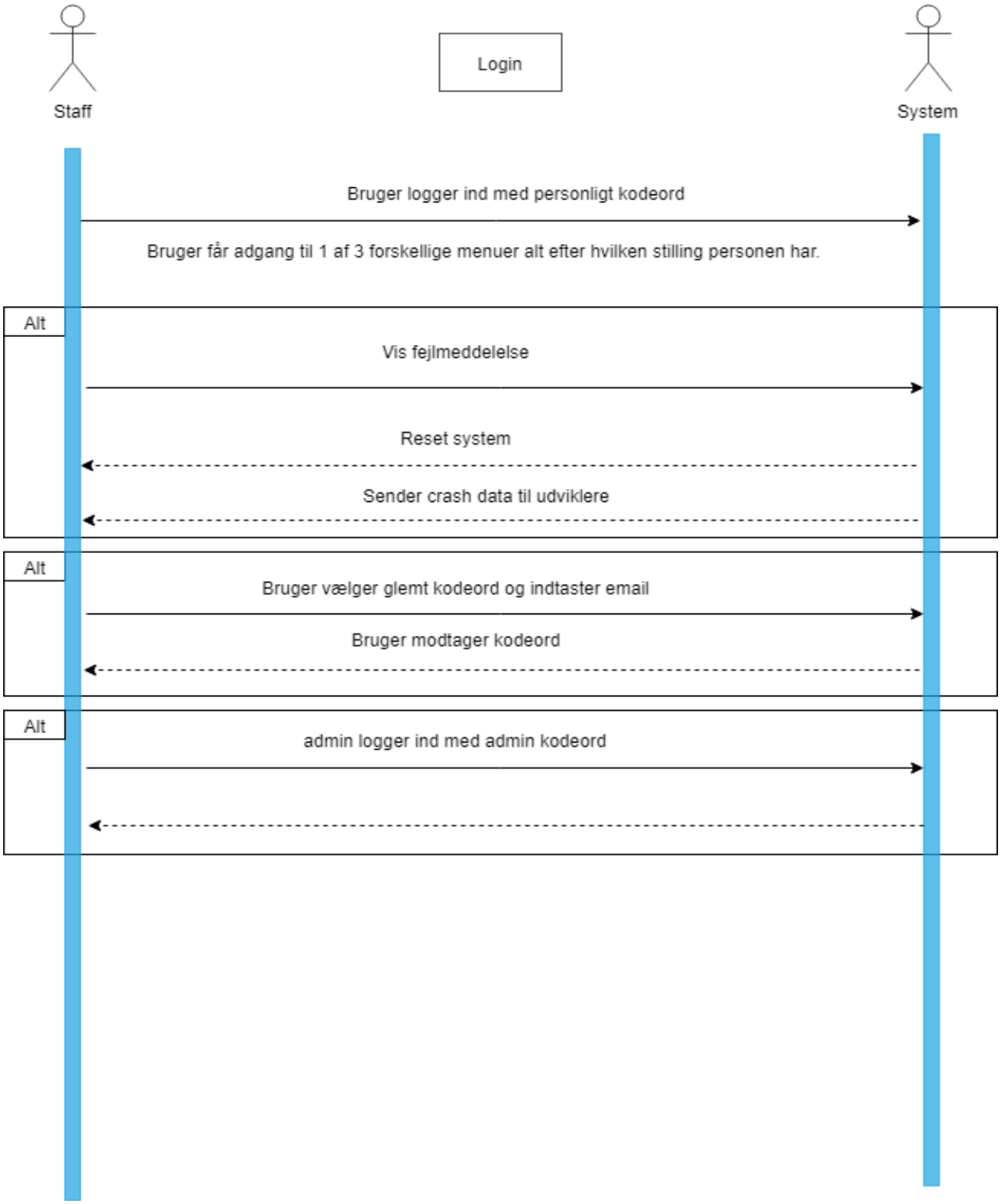
Domænemodel ver. 1

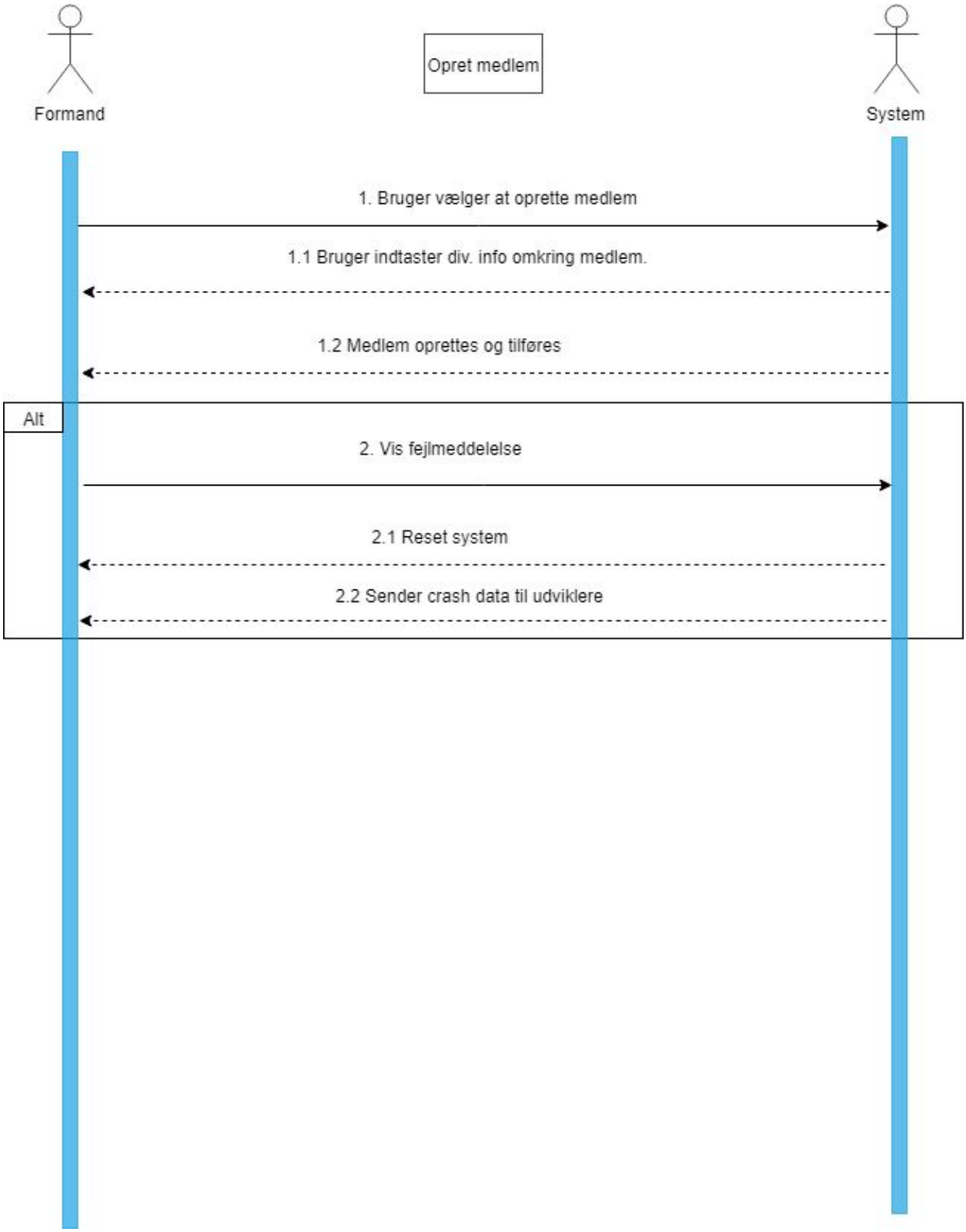


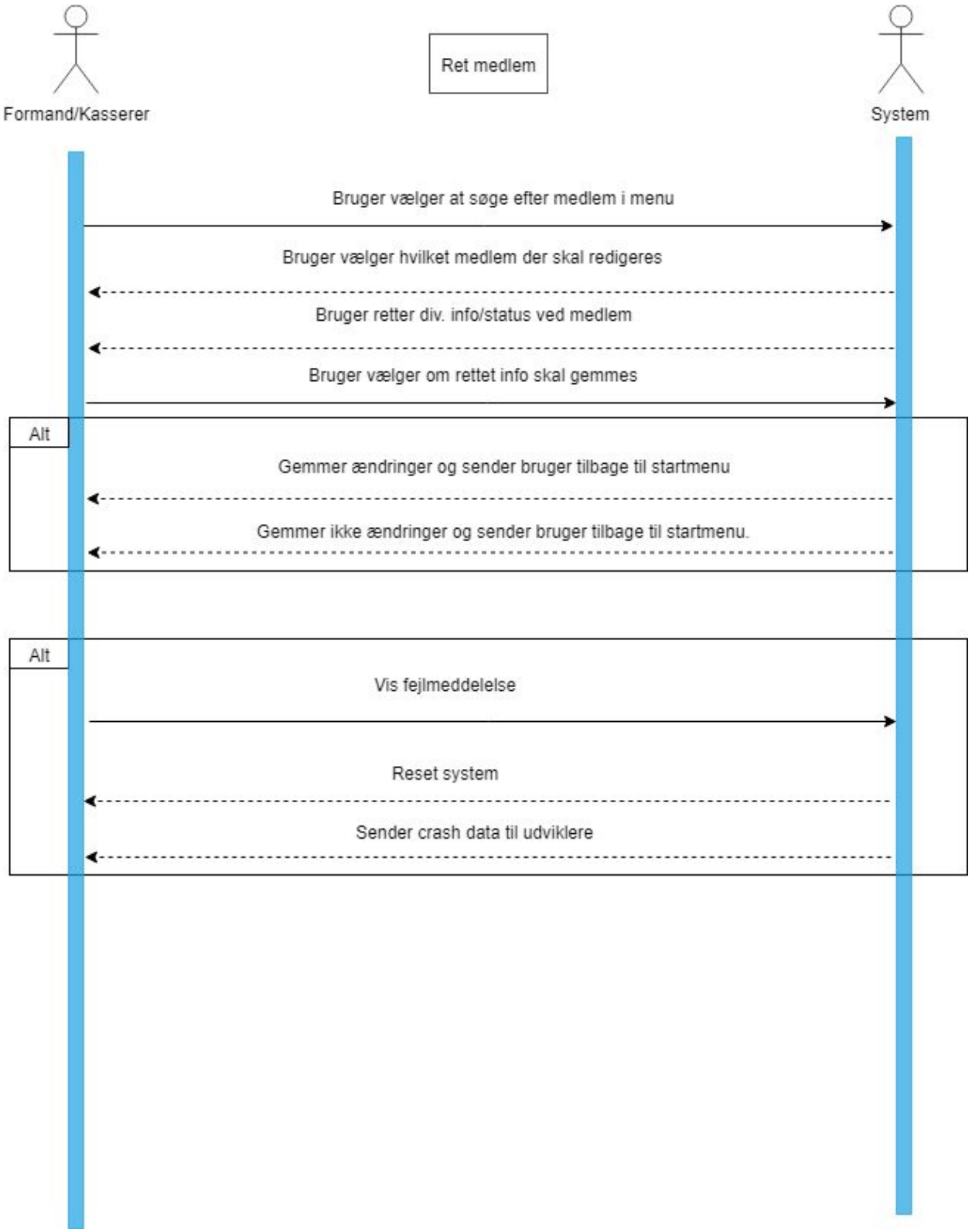
Klassediagram ver. 1

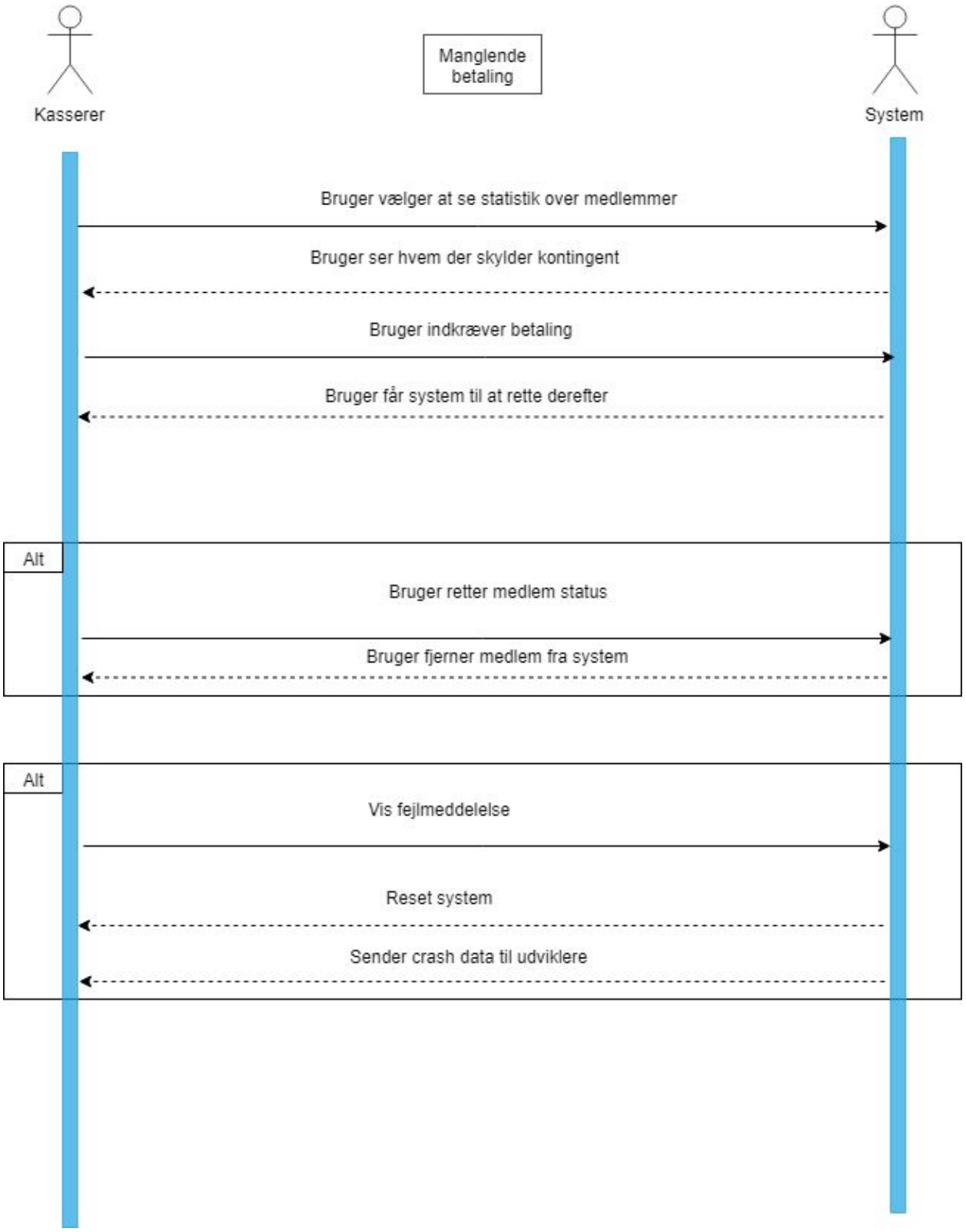


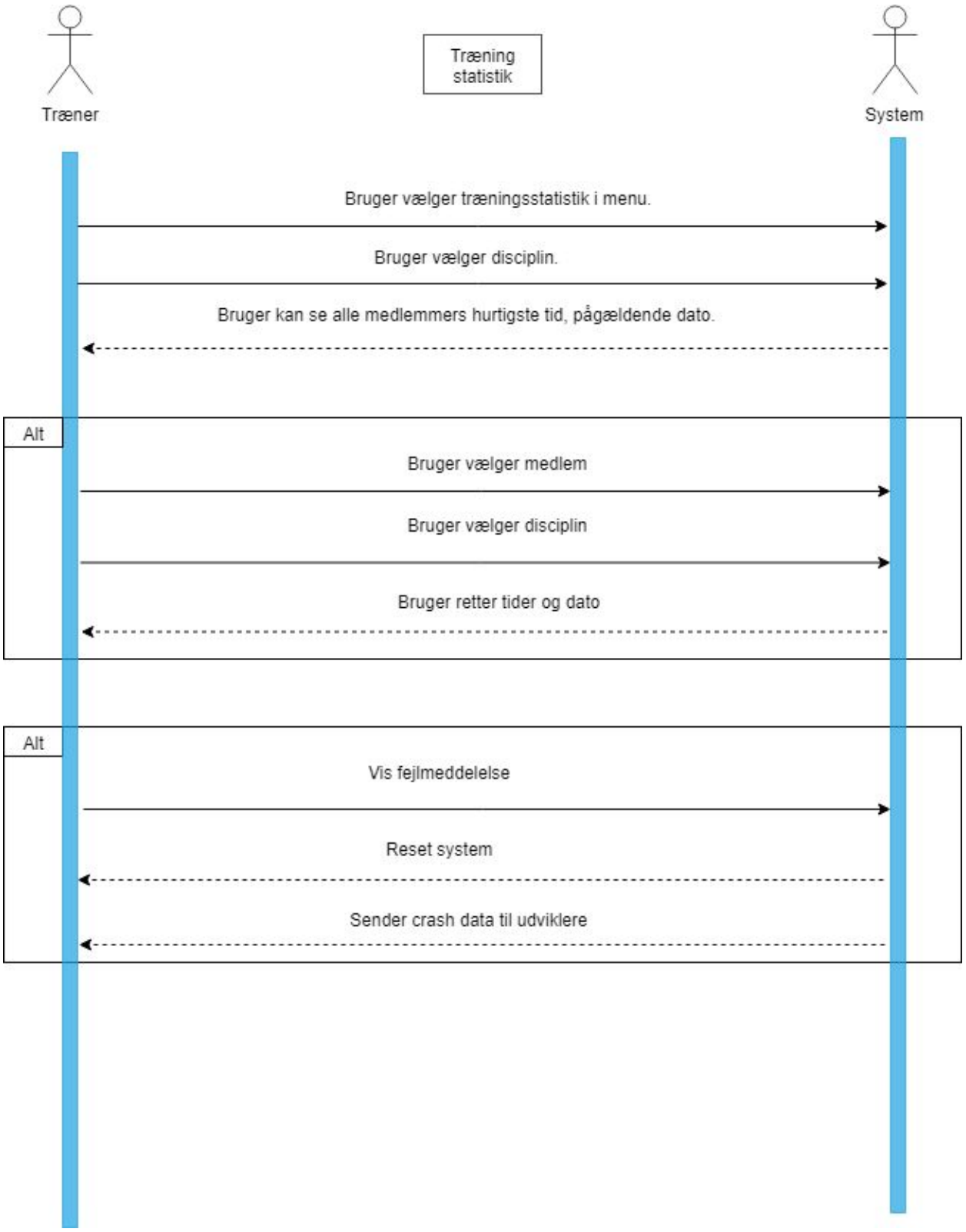
SSD version: 1.0

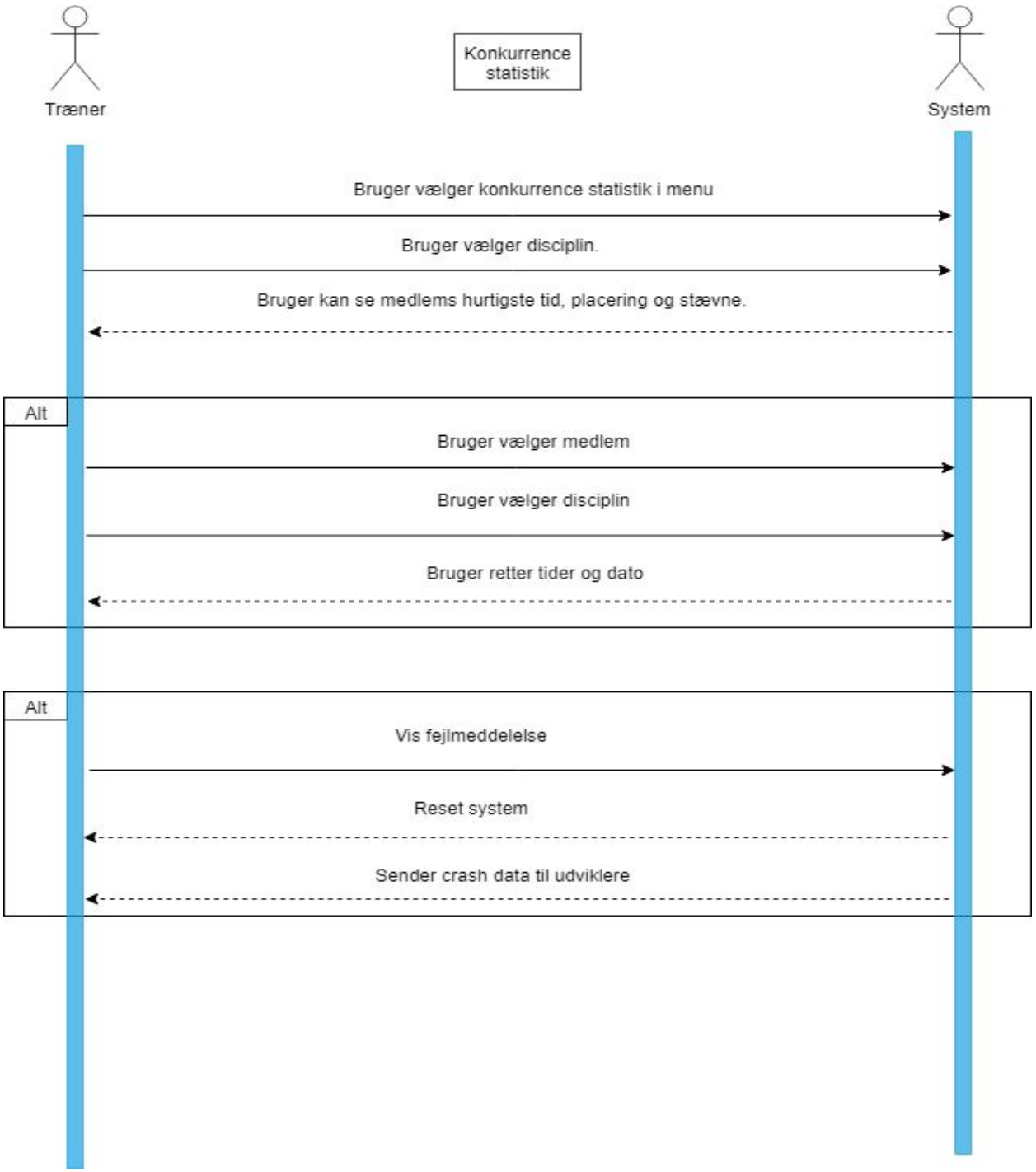



Opret medlem
Formand
System
1. Bruger vælger at oprette medlem
1.1 Bruger indtaster div. info omkring medlem.
1.2 Medlem oprettes og tilføres
Alt
2. Vis fejlmeddelelse
2.1 Reset system
2.2 Sender crash data til udviklere


Ret medlem
Formand/Kasserer
System
Bruger vælger at søge efter medlem i menu
Bruger vælger hvilket medlem der skal redigeres
Bruger retter div. info/status ved medlem
Bruger vælger om rettet info skal gemmes
Alt
Gemmer ændringer og sender bruger tilbage til startmenu
Gemmer ikke ændringer og sender bruger tilbage til startmenu.
Alt
Vis fejlmeddelelse
Reset system
Sender crash data til udviklere


Manglende betaling
Kasserer
System
Bruger vælger at se statistik over medlemmer
Bruger ser hvem der skylder kontingent
Bruger indkræver betaling
Bruger får system til at rette derefter
Alt
Bruger retter medlem status
Bruger fjerner medlem fra system
Alt
Vis fejlmeddelelse
Reset system
Sender crash data til udviklere


Træning statistik
Træner
System
Bruger vælger træningsstatistik i menu.
Bruger vælger disciplin.
Bruger kan se alle medlemmers hurtigste tid, pågældende dato.
Alt
Bruger vælger medlem
Bruger vælger disciplin
Bruger retter tider og dato
Alt
Vis fejlmeddelelse
Reset system
Sender crash data til udviklere


Konkurrence statistik
Træner
System
Bruger vælger konkurrence statistik i menu
Bruger vælger disciplin.
Bruger kan se medlems hurtigste tid, placering og stævne.
Alt
Bruger vælger medlem
Bruger vælger disciplin
Bruger retter tider og dato
Alt
Vis fejlmeddelelse
Reset system
Sender crash data til udviklere