Svømmeklubben Delfinen

Andreas Nissen
Jonas Ladefoged Holm
Tobias Kith Thomsen
DAT17v1

Eksamensprojekt i Software development

Indledning

Ledelsen i Svømmeklubben Delfinen ønsker et administrativt system til at styre medlemsoplysninger, kontingenter og svømmeresultater.

Problemformulering

Klubbens ønske er at hjælpe Formanden med håndtering af medlemmer, herunder aktivitetsstatus, altså om de er passive eller aktive, om de er på junior- eller seniorhold, og om de er motionister eller konkurrencesvømmere og evt. i hvilken disciplin de konkurrerer i.
Kasseren skal kunne se, hvem der mangler at betale, og hvad de evt. skylder.
Træneren skal se en liste af konkurrencesvømmernes resultater sorteret efter tid for hver disciplin og aldersgruppe.

Med udgangspunkt i softwareudviklingskravene bruger vi use cases, use case diagram, domain model, SSDer og SDer og klasse diagram til at arbejde ud fra.

Afgrænsning

Vi har valgt at svømmerne betaler, når de bliver tilmeldt og derefter ved årsskiftet, uanset hvornår de tilmelder sig. Det har vi valgt for at gøre det nemmere at vurdere, hvornår et medlem er i restance.

For at inddele medlemmerne i junior- og seniorhold, har vi valgt kun at holde styr på alder, i modsætning til fødselsdato. Da vi formelt ikke har været omkring `Date` og `Calendar` objekter i Java, har vi nedprioriteret den metode, for at bruge tiden på mere relevante funktionaliteter.

Login er desuden også blevet fravalgt, da vi har vurderet, at det ikke er vigtigt ud fra den problemformulering, vi har fået. Det ville dog være nødvendigt i en virkelig arbejdsopgave.

Vi har ikke direkte kunne tolke på problemformulering, om der skulle være inkluderet en funktionalitet til at betale kontingenter, og vi har derfor fravalgt det.
I restance har vi antaget, at man maksimalt kan være et år i restance, idet man bliver smidt ud af klubben, hvis man går over det. Så vores program regner kun på max et års restance.

Selvom en grafisk brugerflade ikke er noget, vi har haft på første semester, så er vi opmærksomme på, at en en virkelig kunde ville forvente mere end et terminal interface.!"

I vores `findMember` og `findCompetitiveSwimmer` leder vi efter et navn, en liste af gangen. Det gør, at hvis der er en der hedder det samme i `member` og `CompetitiveSwimmer,` så kommer den ikke længere end til `ArrayList<Member>`. Vi ved godt, det ikke er optimalt, men har valgt at lave det sådan pga. manglede tid.

Design

Use cases:

Vi har lavet 7 use cases, og vi har udvalgt 2, som vi vil uddybe.

#5 Read: Se medlemmer i restance

Actor: kasserer
Story:
Kassereren får ved åbning af programmet en menu, hvor han kan vælge at se en liste over medlemmer, der ikke har betalt årligt medlemskab inden for betalingsfristen.

I use case 5 ser vi på, hvordan kasseren kan se hvilke medlemmer, der er i betalingsrestance, og hvordan vi tænker, det skal laves i programmet.

#7 Read: se top 5 hurtigste svømmetider

Actor: Træner

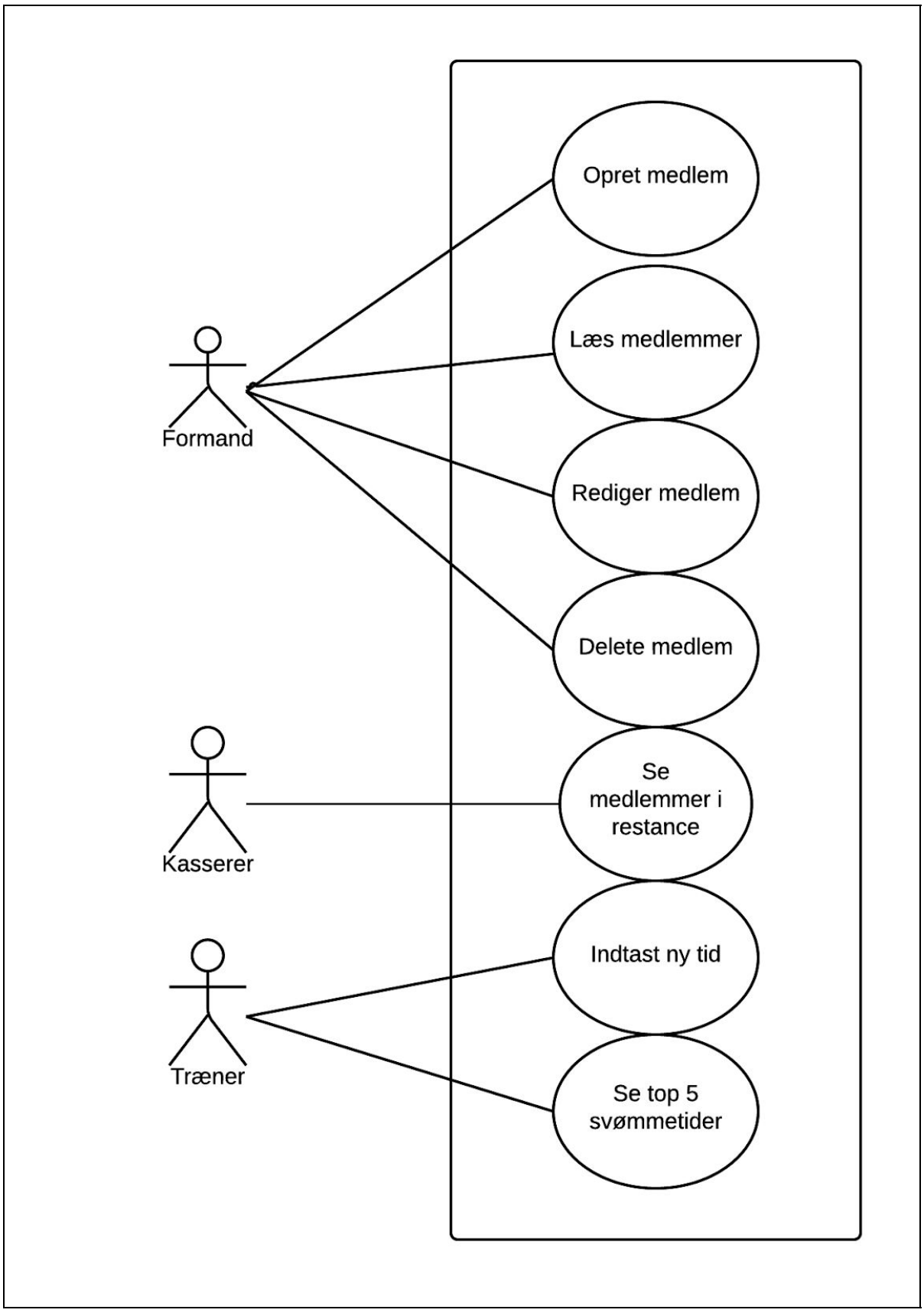
Story:

Når træneren går ind i systemet, kan hun se de 5 bedste tider i hver disciplin for henholdsvis junior- og senior-hold, med navne tilknyttet.

I use case 7 var det vigtigt at vide, hvilke faktorer der gjorde sig gældende for, hvad en "tid" var, og at der var forskellige svømmere og discipliner, der gjorde, at ikke alle tider var ens. Samtidigt var det også i vores overvejelser, at det skulle være hurtigt og nemt for svømmetræneren at få et overblik.

Use case diagram:

Vi har sat vores use cases ind i use case diagrammet for at danne os et overblik over aktørenes interaktioner med use cases. Vi synes dog ikke, det har den store relevans, da programmet ikke er stort nok til, at det giver mening i dette projekt.


Opret medlem
Læs medlemmer
Rediger medlem
Delete medlem
Se medlemmer i restance
Indtast ny tid
Se top 5 svømmetider
Formand
Kasserer
Træner

Navneord og udsagnsord:

Vi fandt følgende ord brugbare i vores use cases:

Navneord:

Formand, formular, personen-adresse, fødselsdato, medlemskabsaktivitet, medlemskabsstatus. *Programmet,* menu, liste, medlem, medlemsnummer, navn. oplysninger, *beslutning.*
Kassereren, medlemskab, betalingsfristen.
Træneren, *dagen,* præstationer, konkurrencesvømmer, tid, *systemet,* junior-hold, senior-hold.

Udsagnsord:

udfylder, specificere, *åbent* vælge, *se*, søge, redigere, ændre, sletter, bekræfter, *får*, betalt, *har*, noteret, indtaster, præsteret, *går*, tilknyttet

Domæner:

Formand, menu, medlem, Kasserer, Træner, præstationer, konkurrencesvømmer, betalingsfristen

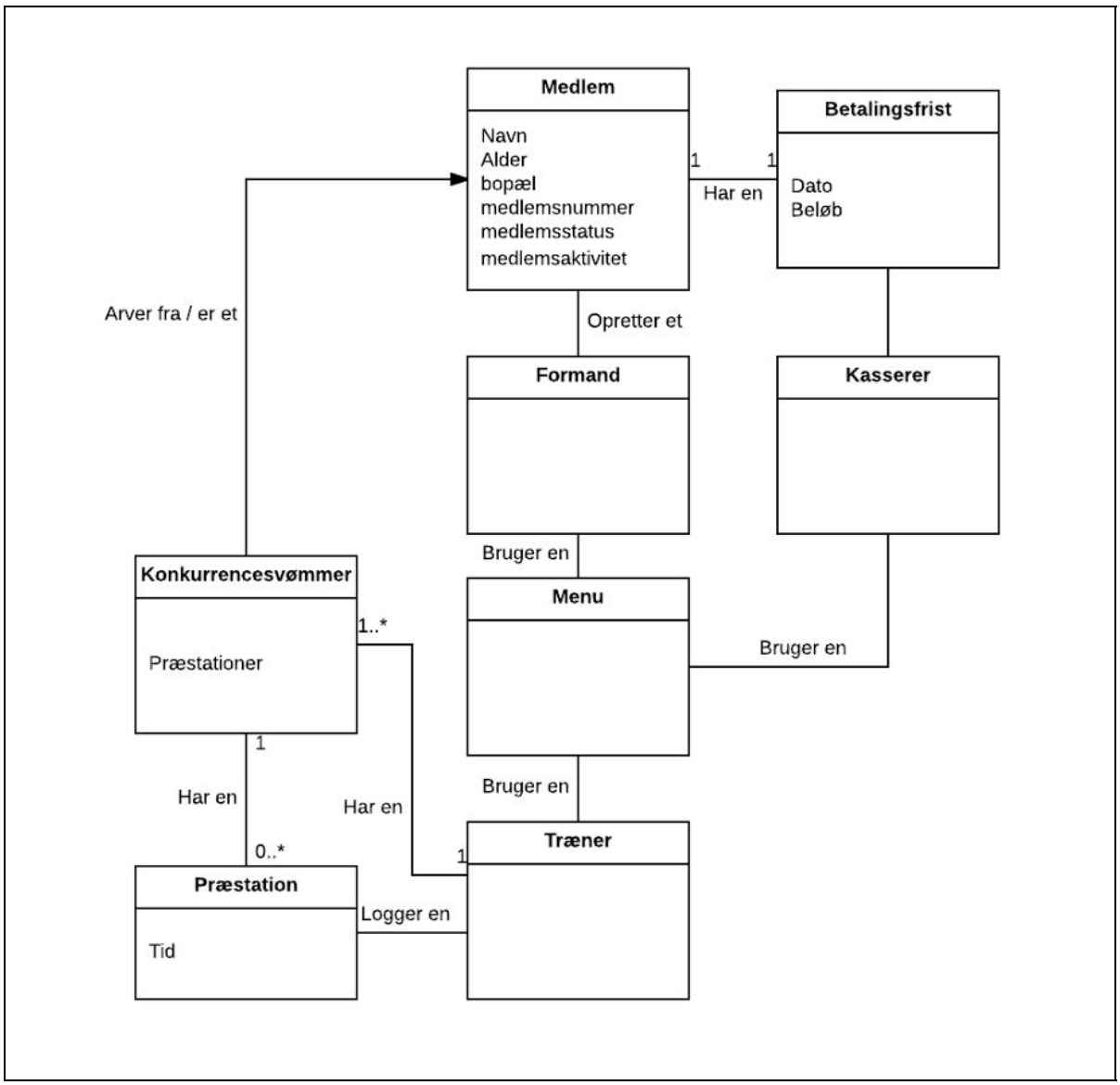
Attributter:

personen-adresse, fødselsdato, medlemskabsaktivitet, medlemskabsstatus, medlemsnummer, navn, medlemskab, junior-hold, senior-hold, tid

Vi har taget alle navneord og udsagnsord fra vores usecases, hvorefter vi fjernede dubletter og redundante ord. Vi endte så med en liste, som vi har brugt til domænemodellen og til at lave metoder ud fra. Man kan se processen her:

Domænemodel:

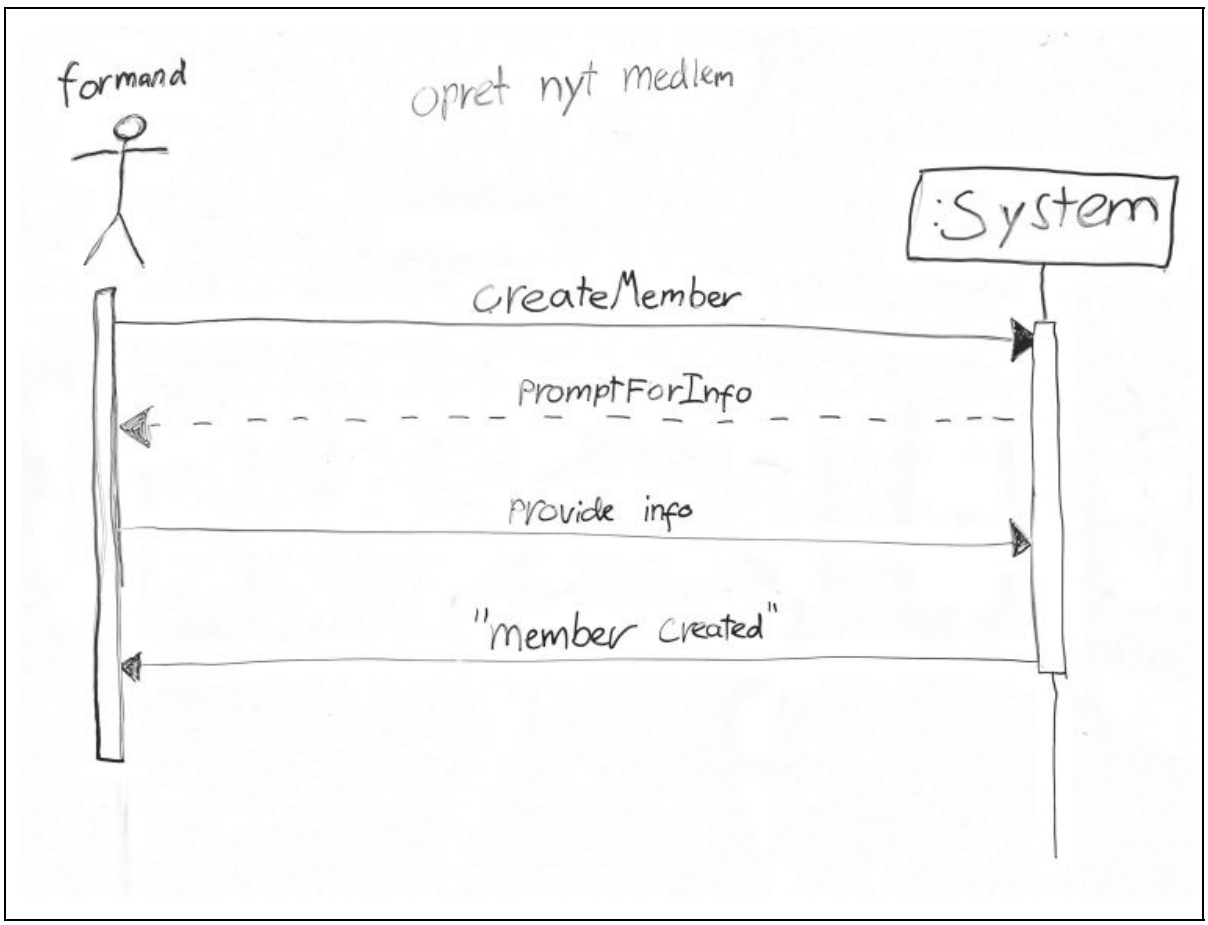
Her er den domænemodel vi har lavet ud fra vores navne- og udsagnsord

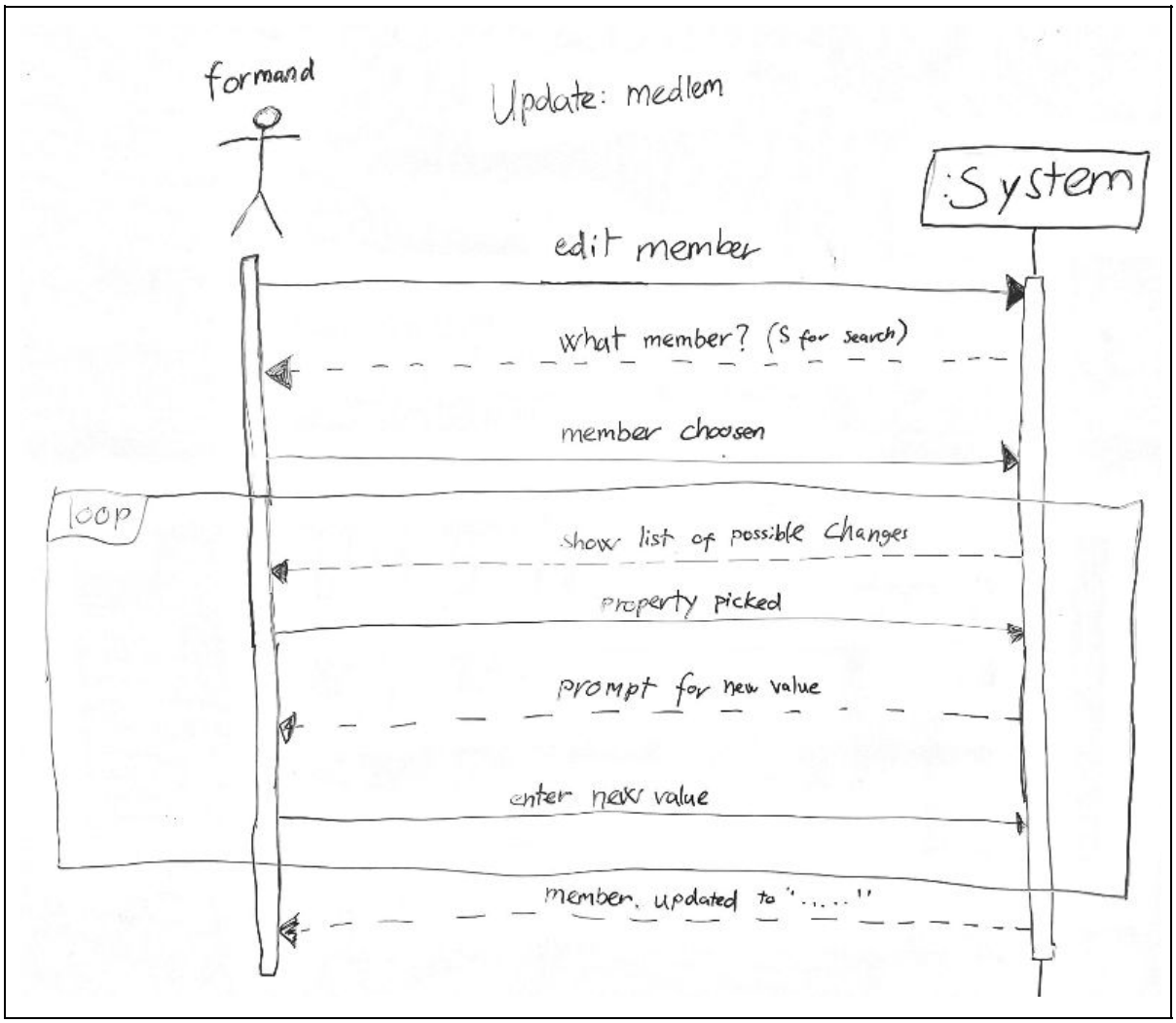


Vi har brugt domænemodellen til at vise interaktionerne i vores program. Vi har ikke brugt alle klasserne fra vores domænemodel i vores klassediagram. Men den har givet et godt overblik over, hvordan programmet hænger sammen. Samtidigt har vi denne gang fået sat attributter på de forskellige klasser for et mere nuanceret overblik, så kunden lettere vil kunne se, om vi har de rigtige informationer med.

SSD:

Her er de SSD vi har valgt at lave til vores program.

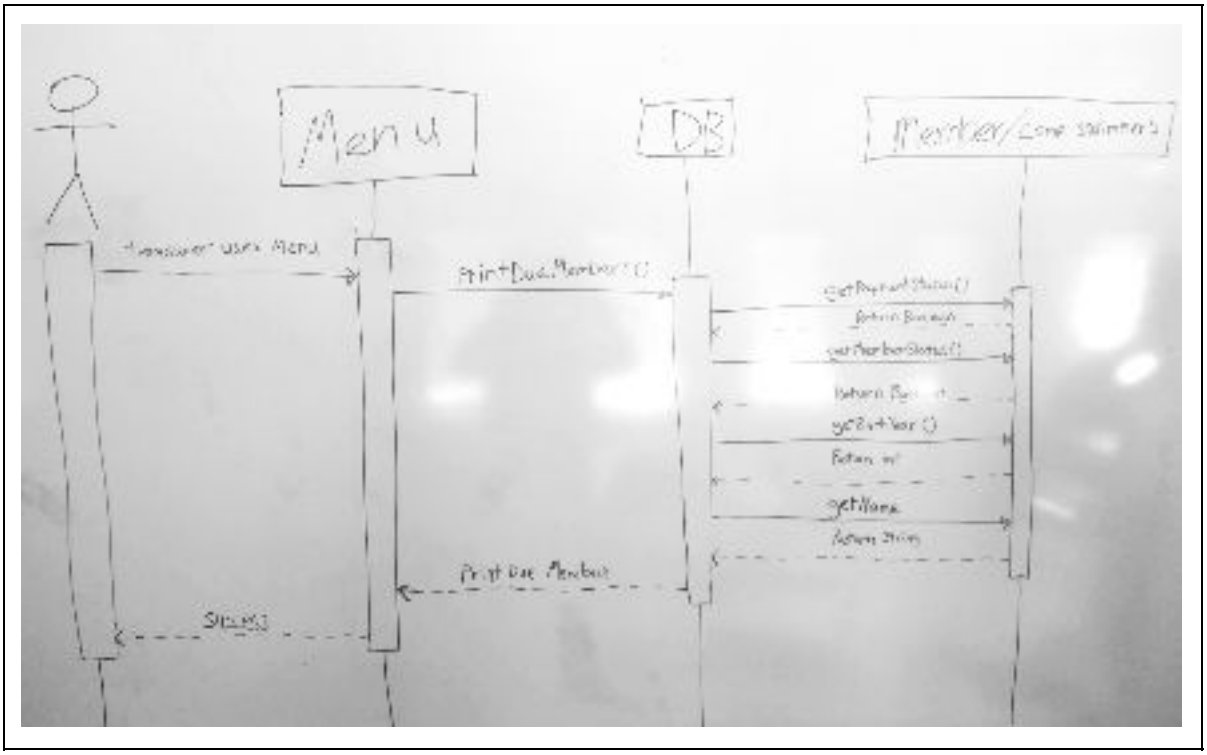




Vi har valgt at lave SSD over nogle af de vigtigere use cases i programmet. Vi synes dog, at med den længde og kompleksitetsgrad vores use cases har, at de er lidt overflødige. Men i et mere komplekst program kan vi godt se deres nødvendighed.

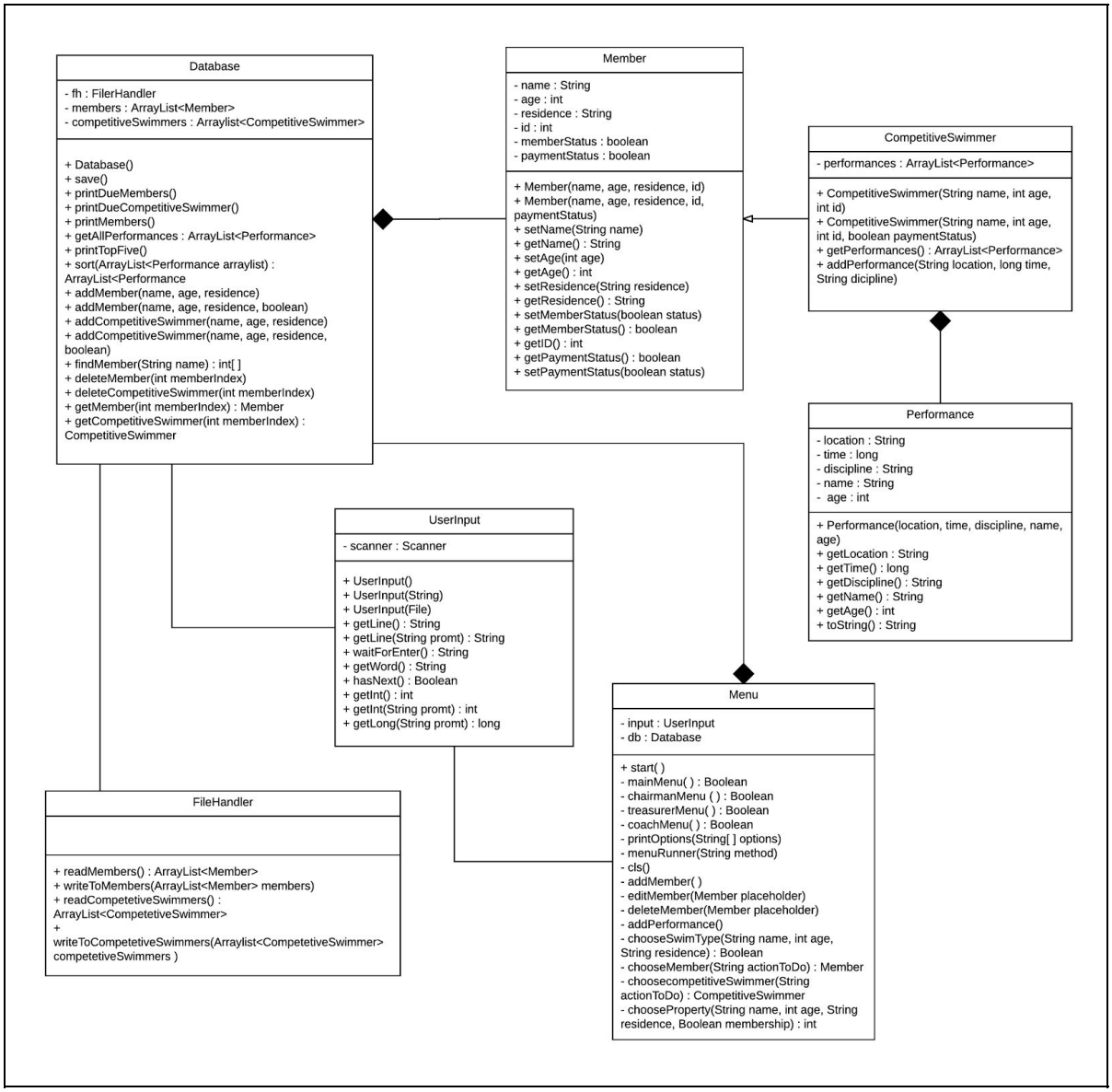
SD:

Diagrammet var et af de vigtigste værktøjer, vi havde til at kunne forstå og diskutere vores program.



Her kan man se, hvordan interaktionen mellem klasser fungerer, hele vejen fra useren, ind i systemet og tilbage igen. Korrektheden af denne måde at tegne diagrammet op, kan sikkert diskuteres, men det gav os et overblik at kunne følge flowet hele vejen til/fra actor.

Klasse Diagram:



Vores klassediagram er lavet på baggrund af vores domænemodel, men med en del forandringer. Vi har f.eks. fjernet formand, kasserer og træner, da vi ikke synes de gav god mening i vores klassediagram. Ellers har klassediagrammet givet os et godt overblik over, hvordan klasserne påvirker hinanden. Vi har i meget stor grad programmeret ud fra vores klassediagram og brugt det som et godt værktøj til at

sørge for, vi alle er helt med i, hvor vi er i vores program. Efter starten, hvor vi kun programmerede ud fra klassediagrammet, nåede vi senere i forløbet til en fase, hvor vi begyndte at kode mere frit, for så at tilføje det til klassediagrammet bagefter. Det gav os en mere agil måde at kode på, så vi ikke endte med en waterfall tilgang til programmet.

Design konklusion:

Vi har brugt de UML-diagrammer, som vi har lært i løbet af semesteret. Nogle af diagrammerne har hjulpet mere end andre. Vi har især fundet værdi i brugen af use cases, sequence diagrams og class diagrams. Det har givet os en god fornemmelse for naturen af programmet, og gjort det muligt at lave en konkret handleplan til kode-fasen.

Kode gennemgang

Serialization:

Vi har valgt at bruge serialization til at gemme data i vores program. Tidligere brugte vi input- og outputstream, men vi synes, det blev meget kringlet. Når vi instantiere `db,` så loader vi samtidig også dataen fra vores filer ind. For at gemme dataen når programmet slutter, har lavet en metode, `save()`, der så gemmer dem på filerne, når vi afslutter programmet. Save metoden gør brug af `writeToMember()` og `writeToCompetitiveSwimmer().` Her viser vi vores `writeToMember()` som gemmer dataen fra vores `ArrayList<Member>` i vores fil. Det sker via save-metoden som aktiveres, når vi lukker programmet.

```
public void writeToMembers(ArrayList<Member> members)
{
  try
  {
    FileOutputStream fileOut = new FileOutputStream("Members.db");
    ObjectOutputStream out = new ObjectOutputStream(fileOut);
    out.writeObject(members);
    out.close();
    fileOut.close();

  }
  catch (Exception e)
  {
    System.out.println("Can't write to file");
  }
}
```

Bubble sort

For finde top 5 svømmere, i hver disciplin og aldersgruppe, gør vi løst brug af bubble sort algoritmen. Bubble sort algoritmen fungere ved gentagne gange at gå gennem en liste, trin for trin, og sammenligne med tilstødende element i listen, hvorefter den bytter deres plads, hvis det ikke er i den ønskede rækkefølge. Bubble sort er ikke optimalt til store datasæt, da den har en complexitet på $O(n^2)$, hvor n er antallet af elementer.

```java
public ArrayList<Performance> sort(ArrayList<Performance> arraylist)
{
  if(arraylist.size() < 2)
  {
    return arraylist;
  }

  Boolean swappedLastTime = true;
  while (swappedLastTime)
  {
    swappedLastTime = false;
    for(int i = 0; i < arraylist.size() - 1;i++)
    {
      if(arraylist.get(i).getTime() > arraylist.get(i+1).getTime())
      {
        Collections.swap(arraylist, i, i+1);
        swappedLastTime = true;
      }
    }
  }
  return arraylist;
}
```

Alternativt kunne vi have implementere Quick sort. Quick sort er mere passende i større datasæt, da den fungere ved “divide and conquer”-princippet, hvilket betyder, at den ikke skal gennemgå hele listen flere gange. Quick sort har en complexitet på $O(n^2)$ i værste tilfælde, hvor listen allerede er sorteret og en complexitet på $O(n \log n)$ i bedste tilfælde. Vi fravalgte denne metode, da vi ikke følte, at vi ville vinde noget på det.

Menu optimering

I tidligere programmer endte menuen altid med at have alt for lidt kode-genbrug, og som konsekvens sad vi ofte og skrev redundant kode. For at minimere dette skrev vi en metode til at vise valgmuligheder i menuerne, hvilket før var noget af det, der virkeligt fyldte meget, med alt for mange `println()` statements.

Dette blev til printOptions, der bare tager imod et array af strings, hvor index 0 er menuoverskriften, og efterfølgende index er menupunkter.
Metoden tilføjer selv en exit valgmulighed, og tilføjer derefter dynamisk efter array længde, et antal af menupunkter med tilhørende tal.

```
private void printOptions(String[] options){
    System.out.println(options[0]);
    System.out.println("Options:");
    System.out.println("0: Go back/exit.");
    for(int i = 1; i < options.length; i++){
        System.out.println(i + ": " + options[i]);
    }
}
```

```
String[] menuOptions = {"Main menu",
                        "Chairman's menu",
                        "Treasurer's menu",
                        "Coach's menu"};
printOptions(menuOptions);
```

Som man kan se, er implementeringen, her i main menu, meget simplere, og skal man lave rettelser i f.eks. rækkefølgen eller antal af punkter, er det nu blot at tilføje eller flytte på punktet i arrayet. Man skal ikke længere tænke på at omskrive alle punkter så tallene passer.

Input håndtering - tøjling af Scanner klassen.

Der er indbyggede udfordringer ved at gøre brug af `Scanner` klassen direkte. `nextInt()` giver f.eks. en exception ved modtagelse af en `String`, og griber ikke newline characteren, (hvilket den jo heller ikke burde).
Dette giver dog problemer med at bruge `nextInt()` og `nextLine()` efter hinanden.

Da vi ofte beder om input fra brugeren, valgte vi at løse disse problemstillinger på en gang, i `UserInput` klassen.
Når vi beder om input, efterspørger vi ofte en bestemt slags data. Vi har derfor fundet det praktisk, at få input metoden til selv at kunne skrive, hvilken slags input den ønskede (f.eks. navn, adresse etc). Dette blev løst med en string parameter for navnet på den ønskede data, derved kunne vi spare en linie kode, hver gang vi bad om input.

```
public int getInt(String prompt){
      while(true){
          try{
              System.out.println(prompt + ": ");
              int integer = Integer.parseInt(scanner.nextLine());
              return integer;
          } catch (NumberFormatException e){
              System.out.println(">> Not a number, try again.");
          }
      }
  }
```

Ved at tage imod en hel linje af gangen, og kun videresende en `int` efterfølgende fra den linie, kunne vi slippe for problemet med forkert input, (brugeren får en pæn fejlmeddelelse, er fanget i et `while(true)` loop indtil inputtet er af ønsket type, og dermed returneres). Desuden var der ikke længere en vildfaren newline character at holde styr på.

Samlet konklusion:

Vi valgte fra starten af forløbet, at vi ville lave et færdigt program som ville opfylde problemformuleringen, hvilket vi også fik gjort. Men på grund af tidspres endte omfanget af vores afgrænsninger at blive relativt stort jf. *Afgrænsning*. Programmet opfylder derfor kun de krav, vi blev stillet i problemformuleringen, udover et par enkelte tilføjelser for interessens skyld, som f.eks. Bubble sort.

Som designværktøj har vi fundet LucidChart mere passende til vores arbejdsprocess, da vi har kunne arbejde i ét dokument på samme tid. Det har gjort designprocessen mere naturlig, og gjort det nemmere, løbende, at opdatere vores dokumentation.
Til versionsstyring har vi brugt værktøjerne git, github og GitKraken. Især GitKraken, en GUI, har givet os et godt overblik og nem kontrol over vores program, selv med flere, der skriver kode sideløbende.

Selvom at man kunne forvente, at der kunne forekomme gnidninger, især under tidspres, synes vi, at vi har været gode til at håndtere evt. problemer. Vi har desuden været eksemplariske til at overholde vores indbyrdes aftaler og har mødtes regelmæssigt, selv på dage, hvor vi ikke har haft undervisning.
Det har gjort, at vores tidsplan for projektet, er blevet overholdt.
Vi ikke har følt, at vi er blevet presset så meget, at det har kompromitteret vores program, udover de afgrænsninger vi selv har valgt.
Slutresultatet er et produkt, vi synes vi kan være tilfredse med.

Bilag

Bilag 1

Fælles Usecases

#1 Create: Opret nyt medlem

Actor: Formanden
Story:
Formanden udfylder en digital **formular** hvor han specificere **personens adresse**, **fødselsdato**, **medlemskabsaktivitet**, **medlemskabsstatus**.

#2 Read: Se alle medlemmer

Actor: Formanden
Story:
Når **formanden** har **programmet** åbent kan han se en **menu**, hvorfra han kan vælge at se en **liste** over alle **medlemmer** eller søge efter et bestemt **medlem** ved **medlemsnummer** eller **navn**.

#3 Update: Opdater medlemsoplysninger

Actor: Formanden
Story:

Formanden søger efter et bestemt **medlem**, ud fra f.eks. **navn** eller et **medlemsnummer**. Han får herefter en **liste** over de **oplysninger** han kan redigere, hvor han så kan *ændre* **oplysningerne**

#4 Delete: Slet medlem

Actor: Formanden
Story:
Formanden søger efter et bestemt **medlem**, ud fra f.eks. **navn** eller et **medlemsnummer**. Han sletter **medlemmet** og bekræfter hans **beslutning**.

#5 Read: Se medlemmer i restance

Actor: kasserer
Story:
Kassereren får ved åbning af **programmet** en **menu**, hvor han kan vælge at se en **liste** over **medlemmer** der ikke har betalt årligt **medlemskab** inden for **betalingsfristen**.

#6 Create: Indtast ny tid

Actor: Træner
Story:
Træneren har i løbet af **dagen**, noteret de forskellige svømmeres **præstationer**.
Hun søger efter en **konkurrencesvømmers navn** eller indtaster vedkommendes **medlemsnummer**. Og indtaster hvilken **tid** vedkommende har præsteret og hvor den er præsteret.

#7 Read: se top 5 hurtigste svømmetider

Actor: Træner
Story:
Når **træneren** går ind i **systemet**, kan hun se de 5 bedste **tider** i hver disciplin for henholdsvis **junior-** og **senior-hold** , med **navne**

Bilag 2

Navneord og udsagnsord

Navneord:

Formand, formular, personen-adresse, fødselsdato, medlemskabsaktivitet, medlemskabsstatus. *Programmet,* menu, liste, medlem, medlemsnummer, navn. oplysninger, *beslutning.*
Kassereren, medlemskab, betalingsfristen.
Træneren, *dagen,* præstationer, konkurrencesvømmer, tid, *systemet,* junior-hold, senior-hold.

Udsagnsord:

udfylder, specificere, *åbent* vælge, *se*, søge, redigere, ændre, sletter, bekræfter, *får*, betalt, *har*, noteret, indtaster, præsteret, *går*, tilknyttet

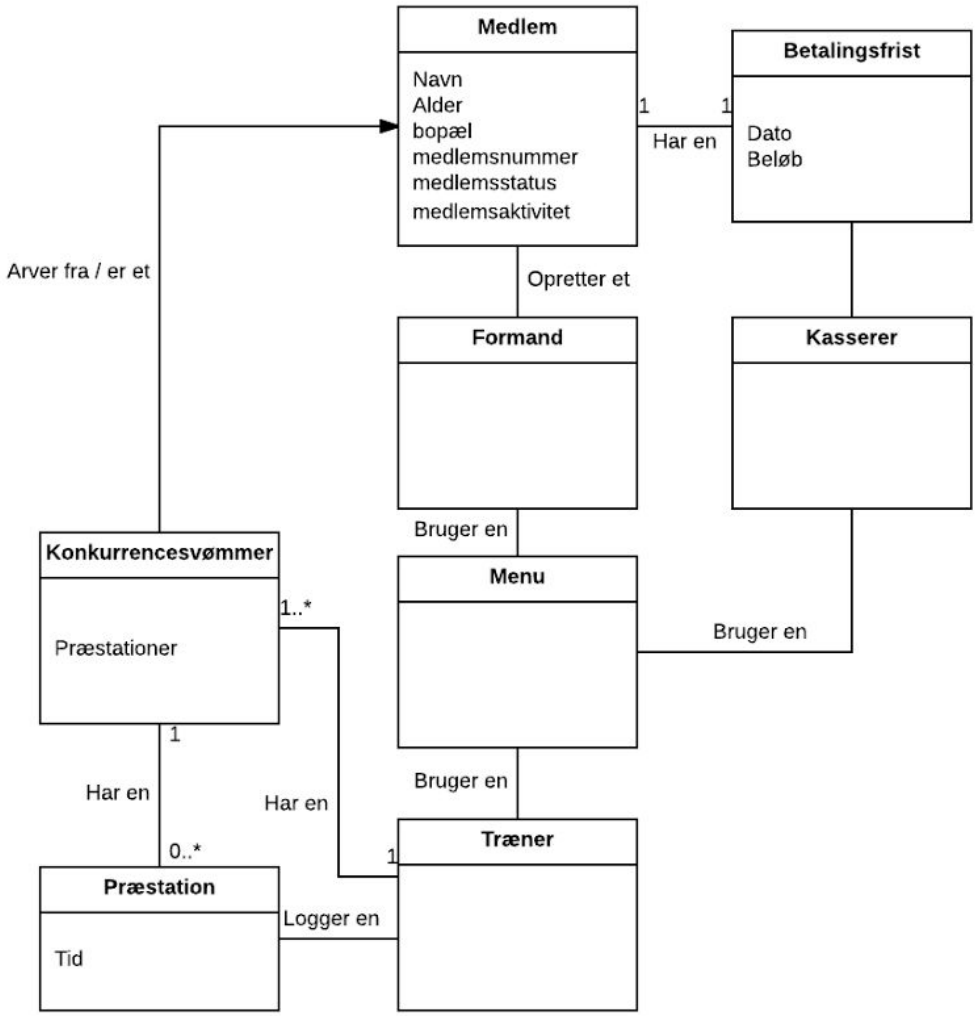
Domæner

Formand, menu, medlem, Kasserer, Træner, præstationer, konkurrencesvømmer, betalingsfristen

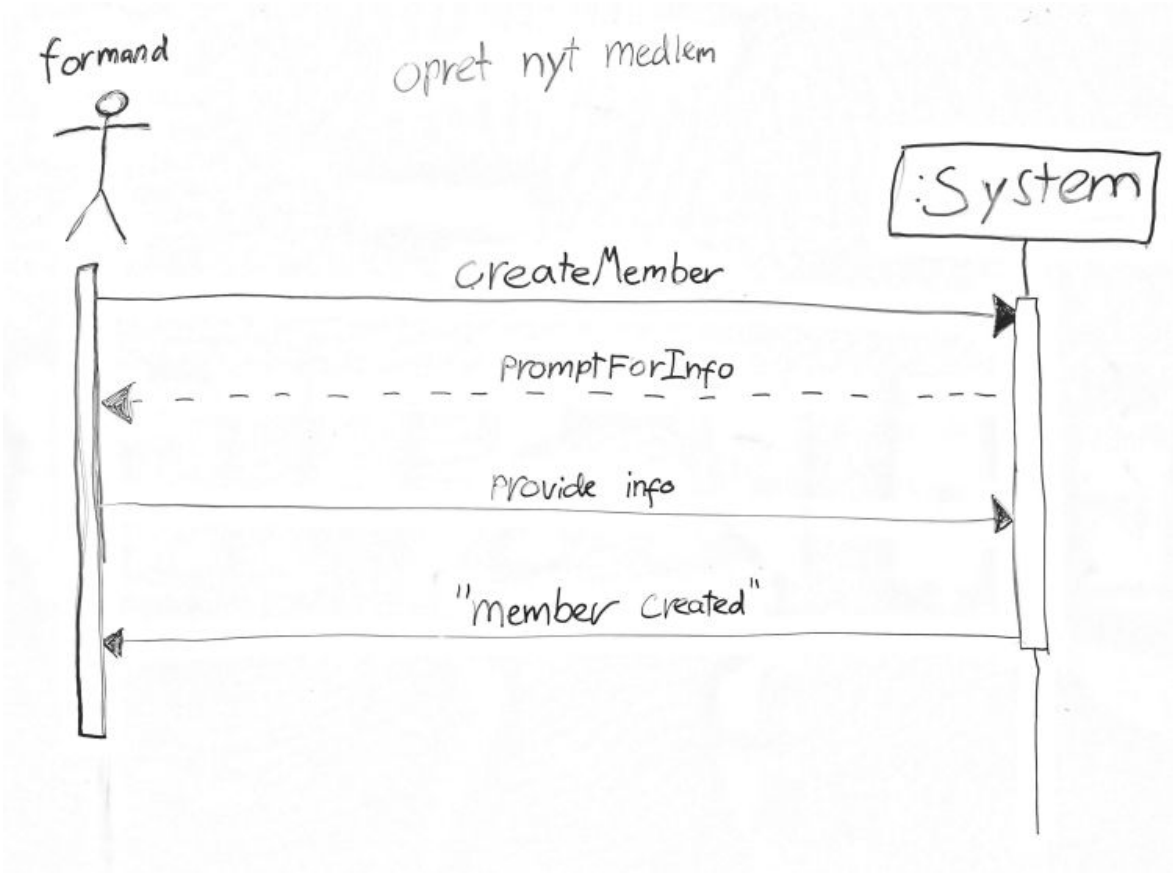
Attributter

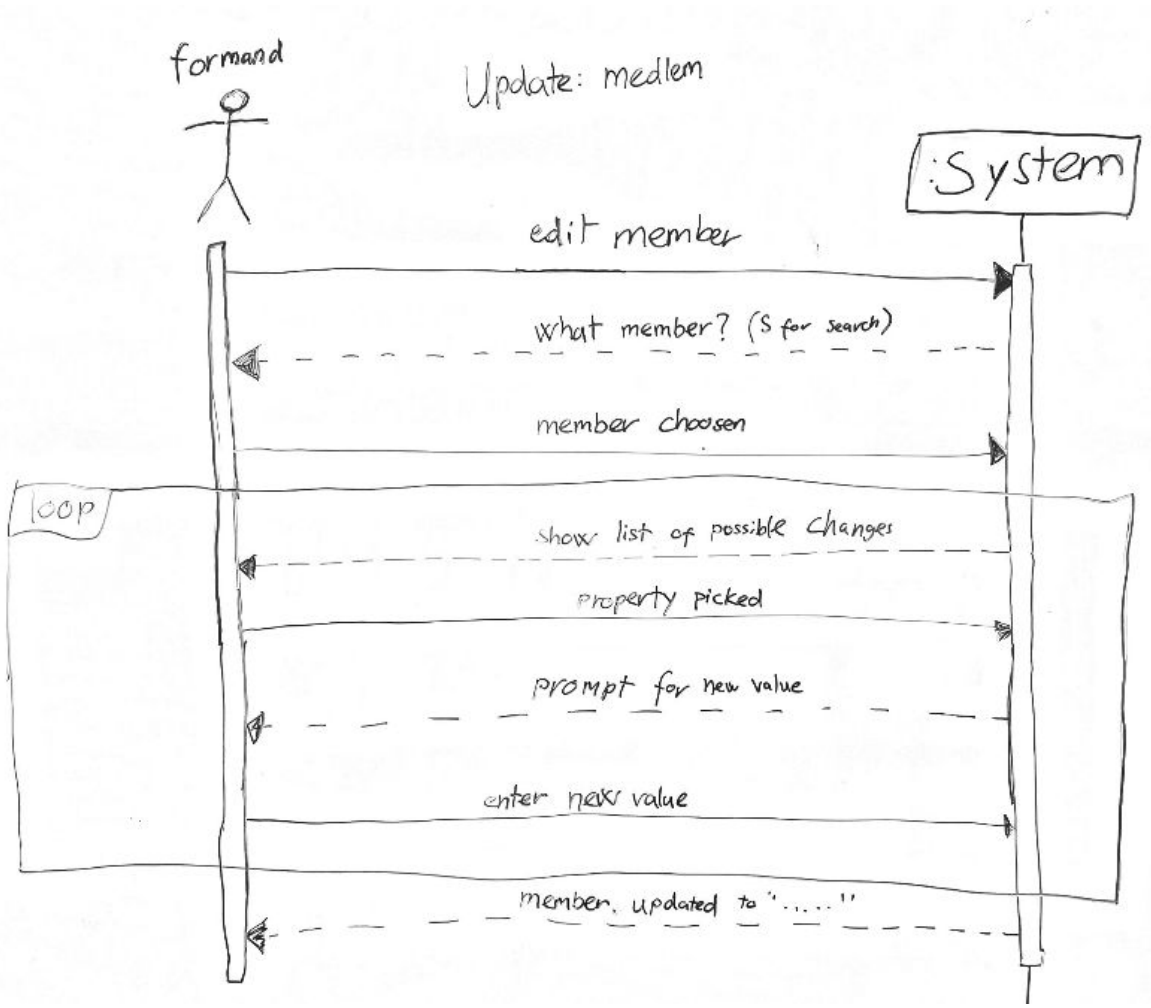
personen-adresse, fødselsdato, medlemskabsaktivitet, medlemskabsstatus, medlemsnummer, navn, medlemskab, junior-hold, senior-hold, tid

Bilag 3

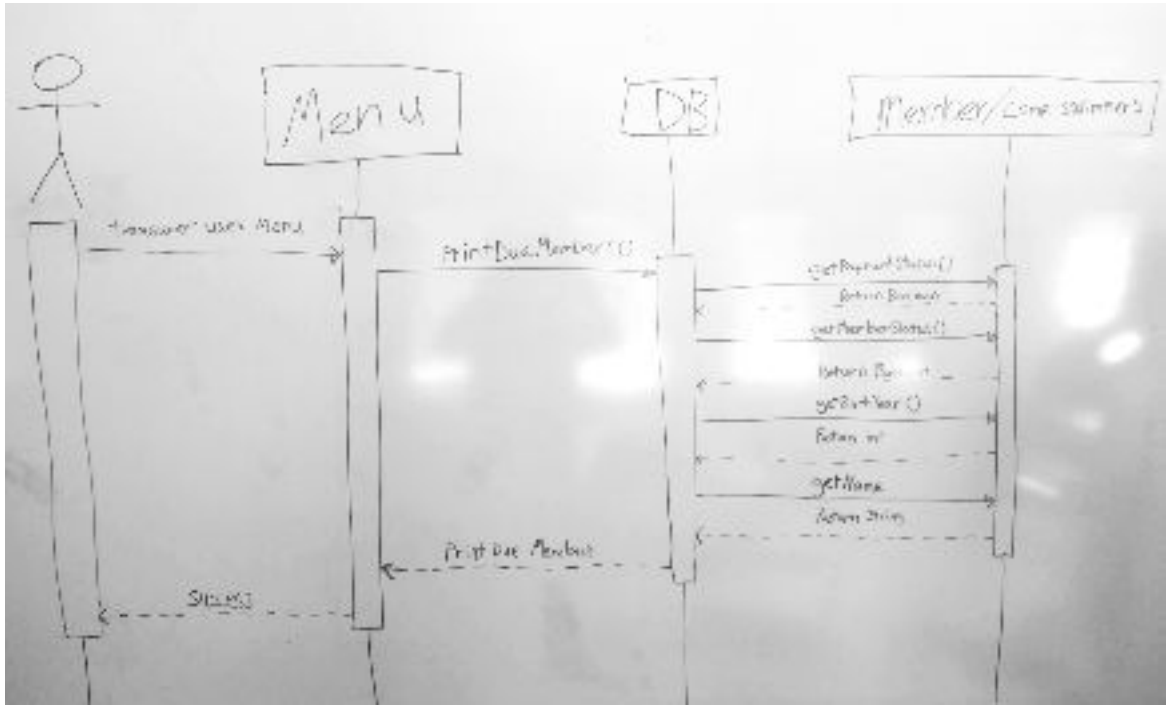

Medlem
Navn
Alder
bopæl
medlemsnummer
medlemsstatus
medlemsaktivitet
Betalingsfrist
Dato
Beløb
1
1
Har en
Arver fra / er et
Opretter et
Formand
Kasserer
Bruger en
Konkurrencesvømmer
Præstationer
Menu
1..*
Bruger en
1
Har en
Bruger en
Har en
Træner
0..*
1
Præstation
Tid
Logger en

Bilag 4

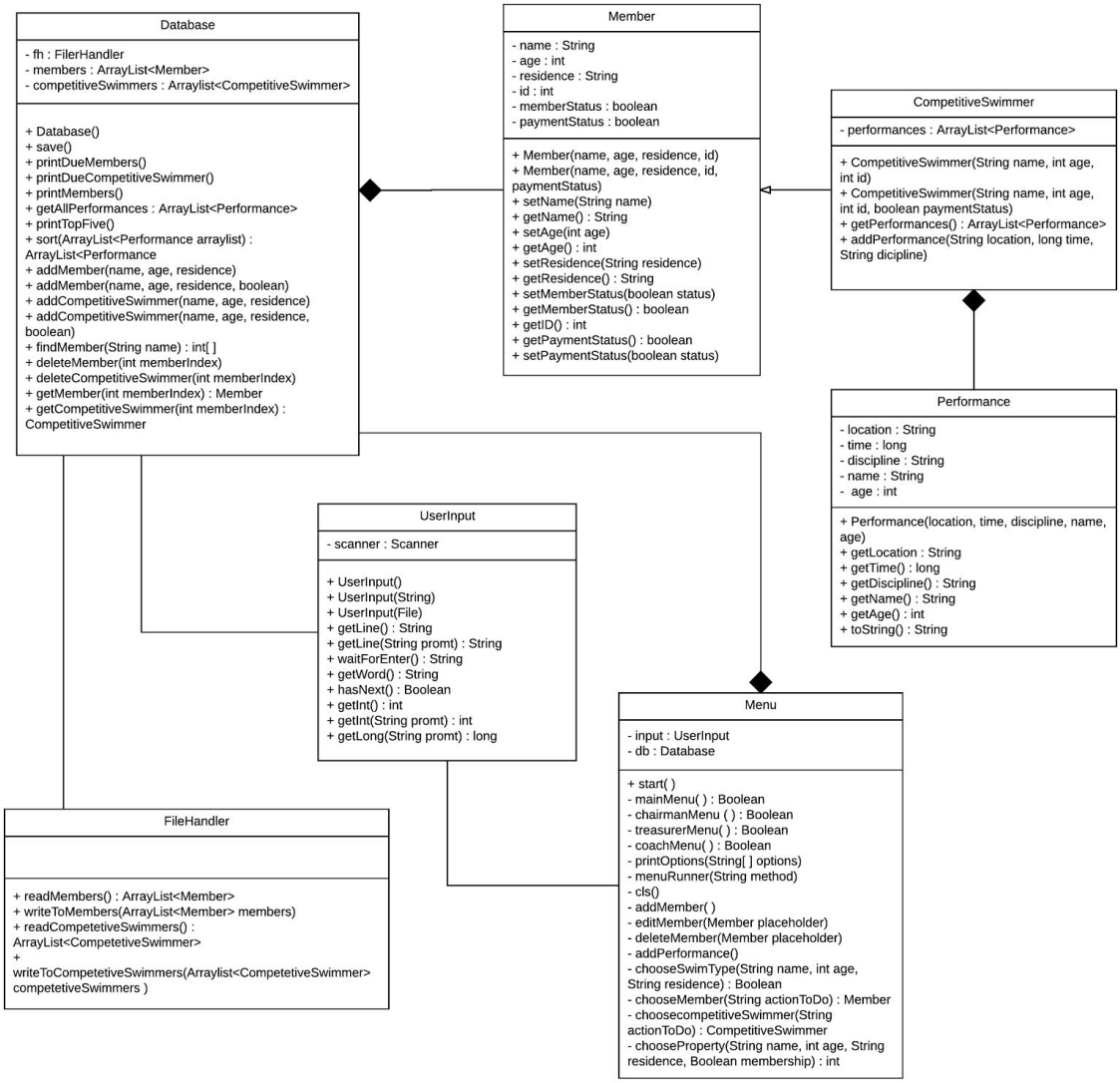

formand
Opret nyt medlem
:System
createMember
promptForInfo
provide info
"member created"


formand
Update: medlem
:System
edit member
what member? (S for search)
member choosen
loop
show list of possible changes
property picked
prompt for new value
enter new value
member updated to "....."

Bilag 5



Bilag 6


Database
- fh : FilerHandler
- members : ArrayList<Member>
- competitiveSwimmers : Arraylist<CompetitiveSwimmer>
+ Database()
+ save()
+ printDueMembers()
+ printDueCompetitiveSwimmer()
+ printMembers()
+ getAllPerformances : ArrayList<Performance>
+ printTopFive()
+ sort(ArrayList<Performance arraylist) : ArrayList<Performance
+ addMember(name, age, residence)
+ addMember(name, age, residence, boolean)
+ addCompetitiveSwimmer(name, age, residence)
+ addCompetitiveSwimmer(name, age, residence, boolean)
+ findMember(String name) : int[ ]
+ deleteMember(int memberIndex)
+ deleteCompetitiveSwimmer(int memberIndex)
+ getMember(int memberIndex) : Member
+ getCompetitiveSwimmer(int memberIndex) : CompetitiveSwimmer
Member
- name : String
- age : int
- residence : String
- id : int
- memberStatus : boolean
- paymentStatus : boolean
+ Member(name, age, residence, id)
+ Member(name, age, residence, id, paymentStatus)
+ setName(String name)
+ getName() : String
+ setAge(int age)
+ getAge() : int
+ setResidence(String residence)
+ getResidence() : String
+ setMemberStatus(boolean status)
+ getMemberStatus() : boolean
+ getID() : int
+ getPaymentStatus() : boolean
+ setPaymentStatus(boolean status)
CompetitiveSwimmer
- performances : ArrayList<Performance>
+ CompetitiveSwimmer(String name, int age, int id)
+ CompetitiveSwimmer(String name, int age, int id, boolean paymentStatus)
+ getPerformances() : ArrayList<Performance>
+ addPerformance(String location, long time, String dicipline)
Performance
- location : String
- time : long
- discipline : String
- name : String
- age : int
+ Performance(location, time, discipline, name, age)
+ getLocation : String
+ getTime() : long
+ getDiscipline() : String
+ getName() : String
+ getAge() : int
+ toString() : String
UserInput
- scanner : Scanner
+ UserInput()
+ UserInput(String)
+ UserInput(File)
+ getLine() : String
+ getLine(String promt) : String
+ waitForEnter() : String
+ getWord() : String
+ hasNext() : Boolean
+ getInt() : int
+ getInt(String promt) : int
+ getLong(String promt) : long
Menu
- input : UserInput
- db : Database
+ start( )
- mainMenu( ) : Boolean
- chairmanMenu ( ) : Boolean
- treasurerMenu( ) : Boolean
- coachMenu( ) : Boolean
- printOptions(String[ ] options)
- menuRunner(String method)
- cls()
- addMember( )
- editMember(Member placeholder)
- deleteMember(Member placeholder)
- addPerformance()
- chooseSwimType(String name, int age, String residence) : Boolean
- chooseMember(String actionToDo) : Member
- choosecompetitiveSwimmer(String actionToDo) : CompetitiveSwimmer
- chooseProperty(String name, int age, String residence, Boolean membership) : int
FileHandler
+ readMembers() : ArrayList<Member>
+ writeToMembers(ArrayList<Member> members)
+ readCompetetiveSwimmers() : ArrayList<CompetetiveSwimmer>
+ writeToCompetetiveSwimmers(Arraylist<CompetetiveSwimmer> competetiveSwimmers)

Bilag 7

Interessentanalyse

Vi analyserer de vigtigste interessenter.

Folk der har direkte kontakt med systemet og dets tilblivelse

- Klubbens ledelse/bestyrelse = kunden/finansering
- Formand
- Kasserer
- Træner

Folk der har indirekte kontakt med systemet (kontingent osv)

- Folk der betaler til klubben (forældre, voksne medlemmer etc)
- Konkurrencesvømmer pga tids data etc
- Passive og Aktive medlemmer

Nul kontakt med systemet

- Junior motionist svømmere (dvs ikke deres forældre)

High power

Keep Satisfied - Klubbens bestyrelse - Projektgruppen	**Manage Closely** - formand - kasserer - Træner
Monitor - passive medlemmer - Juniorsvømmere	**Keep Informed** - Konkurrence svømmere - forældre til juniorsvømmere - aktive medlemmer

Low interest Low Power High Interest

I vores matriks kan man se hvordan vi vægter de forskellige interressenter. de vigtigste er dem der til dagligt vil have med systemet at gøre, i det her tilfælde er det formand, kasserer og træner. så dem vil vi involvere i processen så de føler programmet vil leve op til det de skal bruge. efter dem ligger klubbens bestyrelse som kunden, som vi vil give et grundigt overblik over programmet så de føler de kan se hvad de får får pengene. efter dem kommer

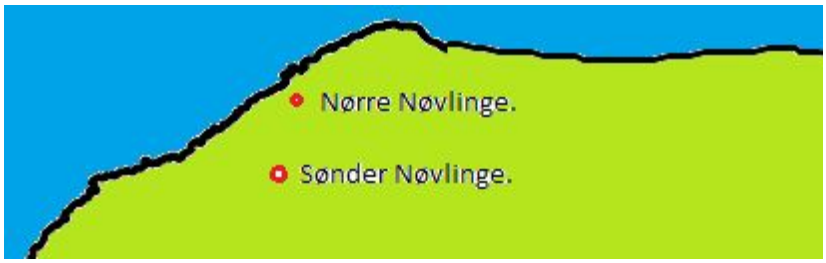
de aktive medlemmer og forældrene til juniorsvømmerne som vil blive holdt opdateret via svømmeklubbens nyhedsbrev og klubbens hjemmeside.

Interressentanalyse						
Interresenter	Deres mål	Tidligere reaktioner	Hvad der kan forventes	Indvirkning pos/neg	Mulige fremtidig reaktion	Ideer
Kasserer (Kurt)	Et system som gør betaling fra medlemmer let og overskuelig	Frustreret over manglende overblik	Bliver meget nemt frustreret, skulle systemet ikke virke.	Kan give en godt indblik i hvad der er vigtigt i et kasser system.	Hvis stabiliteten ikke er i orden bliver han meget besværlig.	Oversigt over medlemmer i restance
Konkurrence svømmerer	At komme med til stævner, systemet er deres middel.	skepsis over at det er et program der skal udvælge dem i stedet for træneren	vil hurtigt protestere, hvis der skulle opstå uoverensstemmelser mellem programmet og deres egne meninger om udtagelse	Glad hvis det giver et godt overblik. Pissed hvis ikke det giver et godt overblik.	chance for de ikke vil lade programmet lave udvælgelserne	træneren skal have endelig beslutning også selvom man ikke ligger på top 5, dvs man skal have en komplet liste som supplement.

til sidst har vi vist hvordan man kan bruge en mere tilbundsgående analyse af den enkelte interressent. her har vi valgt et par eksempler. især en vigtig interressent som kasserer eller formand er vigtig at få afdækket i dybten så man ved hvad man kan forvente i løbet af projektet.

Bilag 8

SWOT - Analyse



Klubben er baseret på frivillig arbejdskraft. Dog har man formand, kasser og næstformand og trænere på lønningslisten. Det vil sige at det ikke er alle der har uendelige mængder tid til at mødes med udviklingsholdet.

Klubben er i sig selv ikke en for profit organisation, dog har de penge på kistebunden til udvidelser mm. Klubben er startet i 1970'erne da der var fokus på fællesskab og sammenhold, og fik lavet helt nye lokaler dengang. Lokalerne er blevet renoveret løbende og derfor i god stand.

Klubbens liv bliver meget holdt i live af lokalsamfundet, i den lille by Sønder Nøvlinge 10 km fra stranden ved Nørre Nøvlinge (den konkurrerende landsby med meget sommer aktivitet pga den flotte sandstrand).

Sønder Nøvlinge folkeskole holder alle sine svømmeundervisningstimer i klubben. Dette er en god indtægtskilde for klubben. Dog har de seneste kommunal reformer gjort at mange af skolerne i nærområdet er blevet lukket eller sammenlagt. Hvilket kan i fremtiden stille spørgsmålstegn ved klubbens økonomi.

Derfor vil klubben begynde at kigge mere på den voksende senior befolkning i området, der har en tung pengepung, og derfor kan bringe mere økonomi til foreningen. Derudover satses der også på det talentfulde konkurrencesvømmer team, der er ledet af den dynamiske dansk amerikaner Michael Morrison, der leder teamet med et internationalt udsyn. Derfor bliver der satset på konkurrence penge og sponsorater.

SWOT Matrix

Strenghts	**Weaknesses**
- God organisationsstruktur - Mange erfaringerne frivillige - Flotte lokaler. - Godt samarbejde med den lokale skole. - Michael Morrison og det dygtige konkurrence svømmeteam.	- Er afhængig af frivillig arbejde. - Er uerfaren i forhold til konkurrencesvømning af internationale standarder. - Splittelser i bestyrelsen (den konservative linie vs den progressive linie)
Opportunities	**Threats**
- Stort senior segment - mulighed for flere og større præmiepenge fra stævner - sponsorater	- Indtægterne fra Skolesvømningen forsvinder - Global opvarmning gør stranden mere attraktivt. - Hvis seniorcenteret begynder svømmeklasse

Vi har sat svømmeklubben delfinens styrker/svagheder/muligheder og trusler et i et matriks, så man kan se hvor svømmeklubben kan forbedre sig og hvor den er stærk.
en af de største svagheder, især med hensyn til vores program, er at bestyrelsen er splittet i hvorvidt de vil bruge deres penge på et nyt program eller om de stadigvæk skal bruge de gamle metoder.
en trussel mod svømmeklubben er at seniorcenteret i området har snakket om at oprette deres egne svømmehold, da de ikke synes at svømmeklubben Delfinen har nok tilbud til pensionister. det vil kunne skære ind i indtjeningen.
Heldigvis har klubben også en de styrker da klubben har en meget stærk organisation med et stærkt bagland som gerne støtter op om klubben og ligger mange frivillige timer i klubben.

Konklusion:
vores konklusion er at klubbens styrker overvejer dens svagheder men at der er nogle trusler, som hvis de sker kunne blive et problem så det ville være en god ide at lave et bedre seniorhold. Udover det ville programmet til at styre medlemmer og tid være en god måde at spare timer på, så man vil kunne benytte de frivillige på den bedst mulige måde.