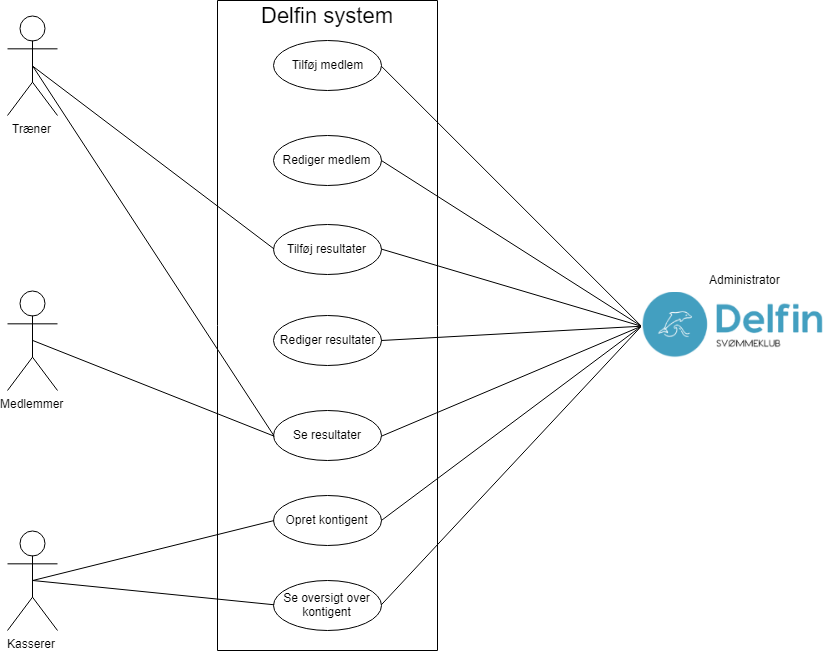
Kundedemo opgave

* hvilke funktionaliteter (use cases), som er blevet implementeret
  + dokumentér med screen dumps af skærmbilleder
* hvorledes programmet er sikret i forhold til de ikke-funktionelle krav:
  + robusthed, fx
    - exception handling og automatiserede unit test
  + vedligeholdelsesvenlighed, fx
    - navngivning af klasser, variable og metoder, dokumentation (javadoc, commits, uml)
  + genbrug, fx
    - modulært design, hjælpeklasser



# Funktionaliteter

### Admin(”Formand”) Use Case:

Systemadministratoren (”formanden”) har mulighed for at se en liste af alle medlemmer i system under menupunktet ”Members”. Listen viser alt information omkring medlemmerne (Cpr, navn, adresse, telefon og aktivitets information). Vinduet har tildelt 2 knapper ”Refresh” og ”Add”.

Refresh knappen vil hente ny data fra SQL Databasen og erstatte tabellens data.  
Add knappen vil vise et nyt vindue, hvor man kan oprette et nyt medlem.

Ved dobbelt klik på en linje vil Create vinduet komme op med informationen fra linjen, og knappen vil nu opdatere information på linjen.

### Træner Use Case:

Trænerne har mulighed for at oprette resultater for medlemmer når de har deltaget i et stævne. Vinduet vil vise alle resultater og en Combobox selekter til at vise top 5 resultater inden for en svømme disciplin. ”Refresh”, ”Add” og dobbeltklik funktioner fungerer på samme måde i dette vindue.

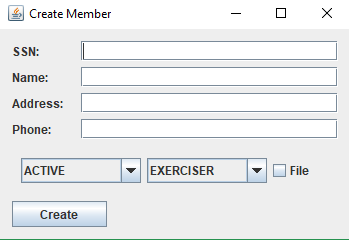
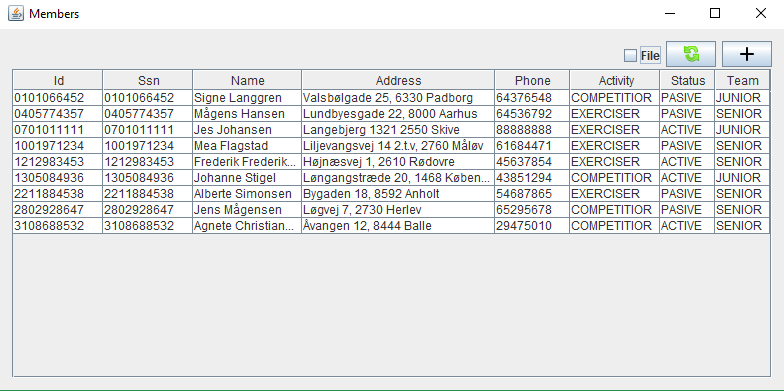
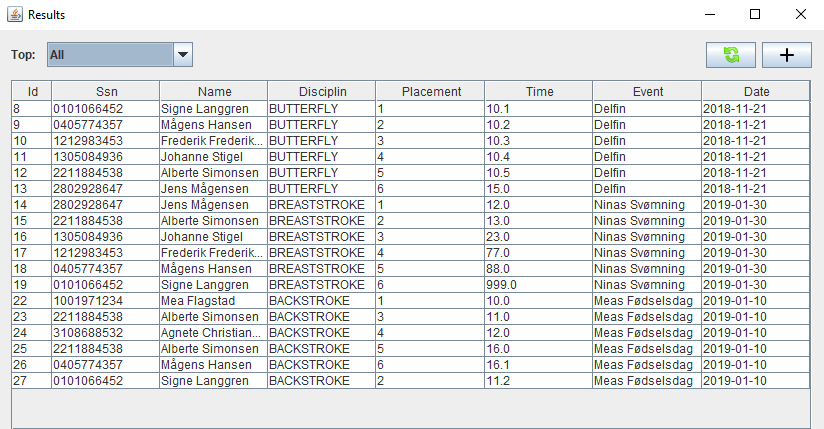
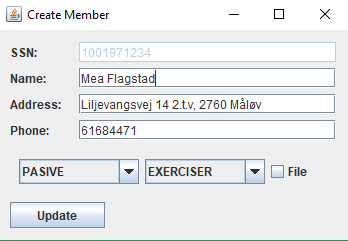
### Kassere Use Case:

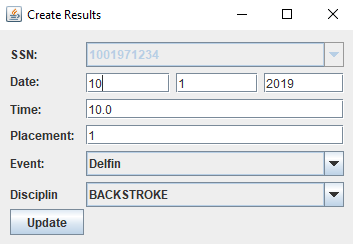
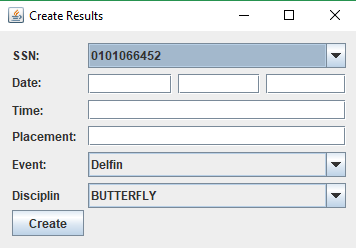
Kassere har mulighed for at tilføje og se kontingenter og betalinger på medlemmer. Vinduet viser alle kontingenter og hvad der er betalt af kontingentet. Pr. linje bliver der udregnet en gæld ud fra kontingentet og betalingen.

Når der skal oprettes kontingent gøres det på ”Subscription” knappen hvor man vælger Cpr nummer. Programmet udregner selv kontingentprisen ud fra medlemmet.

Under knappen ”Payment” kan der tilføjes betalinger til medlemmer når de har betalt. Dette gøres ved at vælge Cpr nummer og skrive betalingen ind.

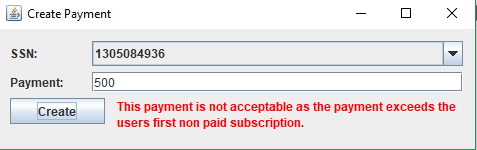
”Show Debt” knappen vil sortere alle betalte kontingenter fra og kun vise dem der står i gæld.

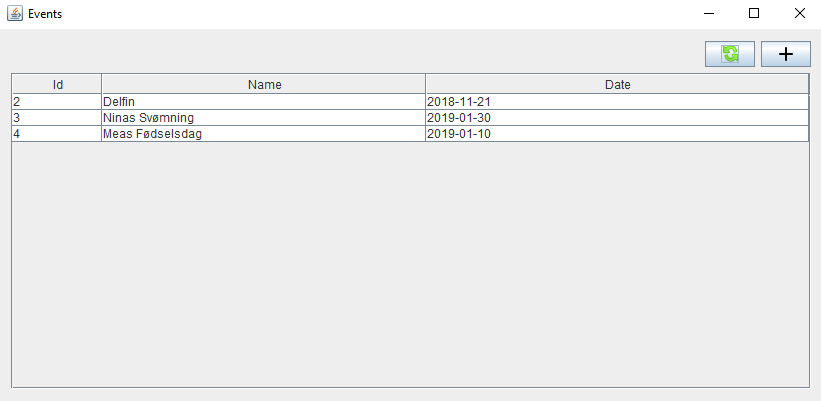
 



# 

# 





# 

# 

# Robusthed

Vores primære exception håndtering sker i datalaget hvor vi snakker med vores database. Vi håndterer også enkelte exceptions i vores logiklag, hvor vi prøver at oprette en forbindelse til vores datalag og derved vores database. Vi har ønsket at skabe en form for robusthed i vores program ved at håndtere bruger input med if sætninger sådan at vi tvinger brugeren til at angive det ønskede data korrekt. Hvis vi kigger på vores createMember metode under MemberController klassen kan vi se at vores if sætninger skaber den ønskede robusthed da vores program ikke stopper med at fungere men i stedet fortæller brugeren at inputtet ikke er korrekt og hvad det forventede input burde være.

# Vedligeholdelsesvenlighed

Vi har forsøgt at skrukturer vores program så det er let og overskueligt at vedligeholde i form af navngivning der giver mening. Dette gælder alle vores klasser eksempelvis Member her ville den logiske sans fortælle en at det nok er en medlemsklasse der er tale om. Det gælder vores attributter i vores klasser eksempelvis ssn eller name der vil ens logiske sans nok fortælle en at der er tale om social security number eller navn. Til sidst gælder det også vores metodenavne som eksempelvis getSsn eller getName hvor ens logiske sans nok ville fortælle en at der her er tale om at få fat på et ssn eller et navn på den givne person. Vi har forsømt lidt i form af kommentarer i vores program så det kan måske være svært for en udvikler lige at overskue hvad enkelte dele af vores program gør. Vi havde dog ingen problemer med at forstå enkelet dele af programmet, som var udviklet af andre gruppemedlemmer end en selv og derfor har vi ikke tænkt videre over at kommenterer på vores kode.

# Genbrug

Programmet er bygget op omkring 3 lags arkitektur. Første lag er præsentationslaget hvor GUI’s ligger, disse GUI’s bruges til at fremvise data til brugeren og til at indtaste data til systemet. I det andet lag som er Logiklaget ligger alle vores klasser og controllers, vi bruger interfaces for at sikre os at vi ikke glemmer funktionalister i vores klasser. Det sidste lag er Datalaget som bruges til at hente og skrive dataen ind i databasen. Her bruger vi også interfaces for at sikre at alle DataAccessors kan hente lister, hente enkelte linjer, oprette data og opdatere data.