

# Delfin2 - Svømmeklubben Delfinen

[GitHub](#)

[ReadMe](#)

*af Rúni Vedel Niclasen, Asger Bjarup og Camilla Jenny Valerius Staunstrup*

## Indhold

*Kort gennemgang*

*Implementerede use cases*

*Ikke funktionelle krav*

*Vedligeholdelsesvenlighed*

*Modulært design*

*Hjælpeklasser*

*Projektets mangler*

## Kort gennemgang

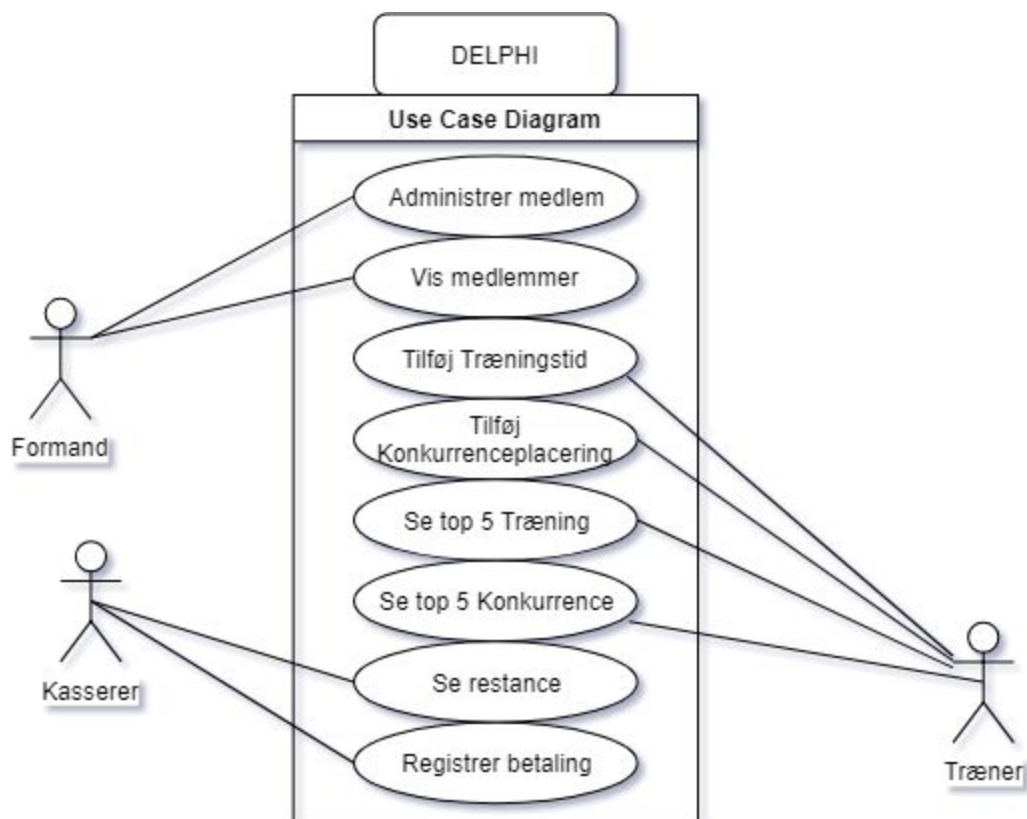
Programmet Delfin2 er et program til administration af svømmeklubben Delfinens medlemmer, så betaling af kontingent, overblik over medlemskabstyper og overblik over klubbens konkurrencer forsimples for administrationen i svømmeklubben. Programmet er skrevet i Java og GUI er opbygget af Netbeans SWT Designer og Swing Designer. Programmets data gemmes og læses fra en fil.

Det er tiltænkt at programmet har tre brugertyper:

- Formand
- Kasserer
- Træner

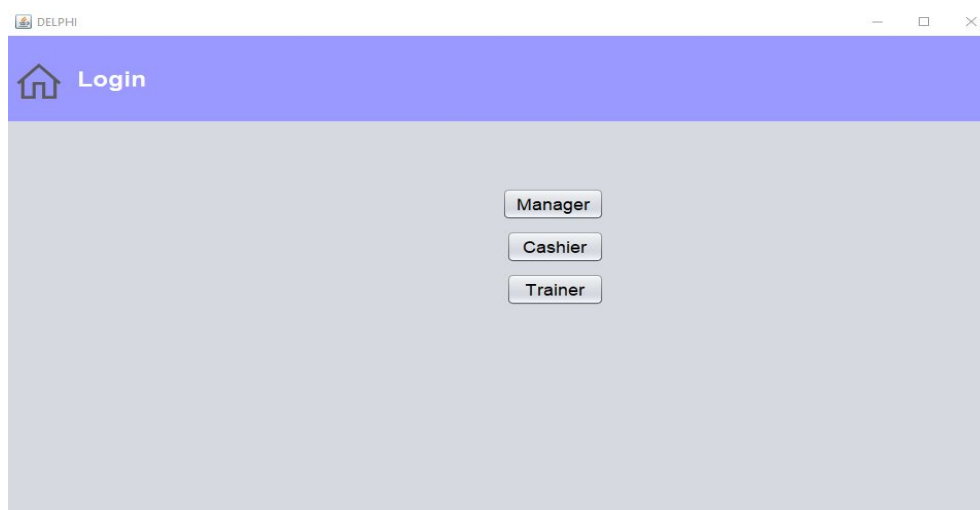
Disse tre brugertyper har hver især adgang til de funktioner i programmet der vedrører deres arbejdsgang i svømmeklubben.

## Implementerede use cases

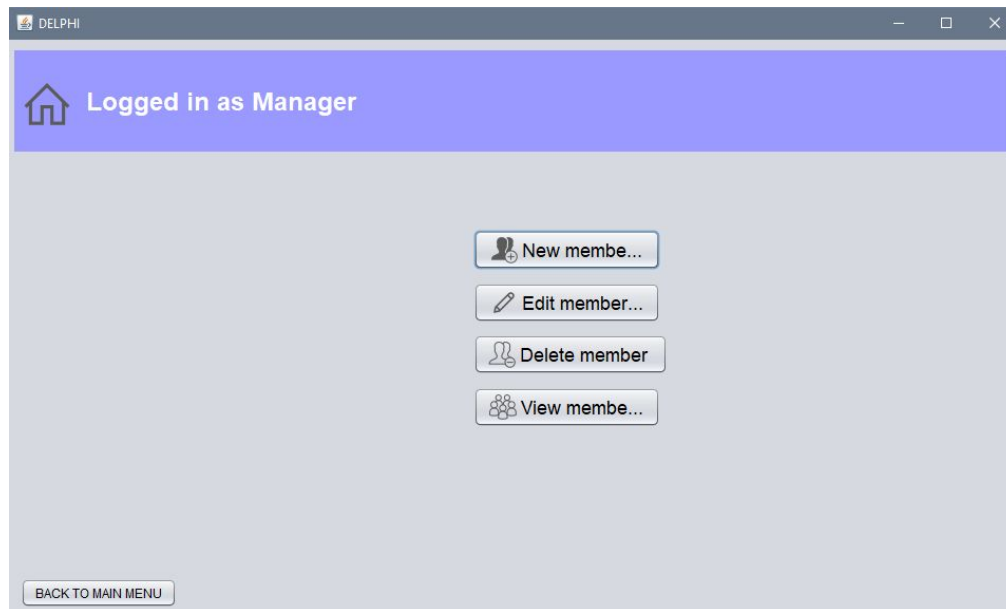


På de følgende sider kan der ses screenprints af alle programmets funktionaliteter.

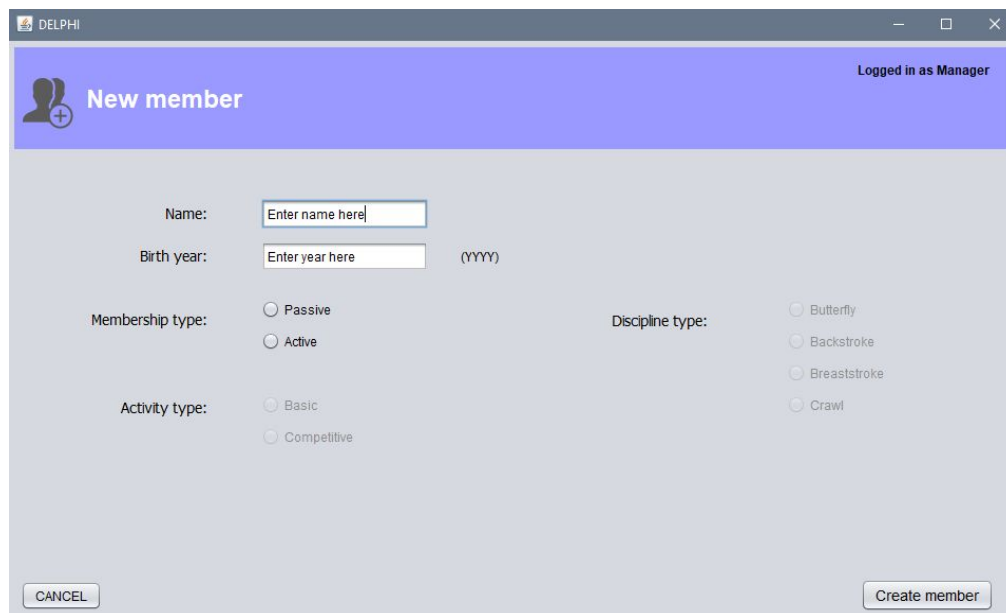
## Login



## Administrer medlem



The screenshot shows a window titled 'DELPHI' with a blue header bar. The header bar contains a house icon and the text 'Logged in as Manager'. Below the header, the main area is light gray and contains four buttons stacked vertically: 'New membe...' (with a person icon), 'Edit member...' (with a pencil icon), 'Delete member' (with a person icon), and 'View membe...' (with a group of people icon). At the bottom left, there is a button labeled 'BACK TO MAIN MENU'.



The screenshot shows a window titled 'DELPHI' with a blue header bar. The header bar contains a person icon with a plus sign and the text 'New member'. In the top right corner of the header, it says 'Logged in as Manager'. The main area is light gray and contains the following form fields and options:

- Name:** A text input field with the placeholder 'Enter name here'.
- Birth year:** A text input field with the placeholder 'Enter year here' and '(YYYY)' to its right.
- Membership type:** Two radio buttons labeled 'Passive' and 'Active'.
- Activity type:** Two radio buttons labeled 'Basic' and 'Competitive'.
- Discipline type:** Four radio buttons labeled 'Butterfly', 'Backstroke', 'Breaststroke', and 'Crawl'.

At the bottom left, there is a button labeled 'CANCEL'. At the bottom right, there is a button labeled 'Create member'.

DELPHI

— □ ×

 **Edit member** Logged in as Manager

Choose member


ID: 1, Name: Hans, Birthyear: 1902, Years Paid: [], MembershipType: PASSIVE

OK

CANCEL

DELPHI

— □ ×

 **Edit member** Logged in as Manager

Name:

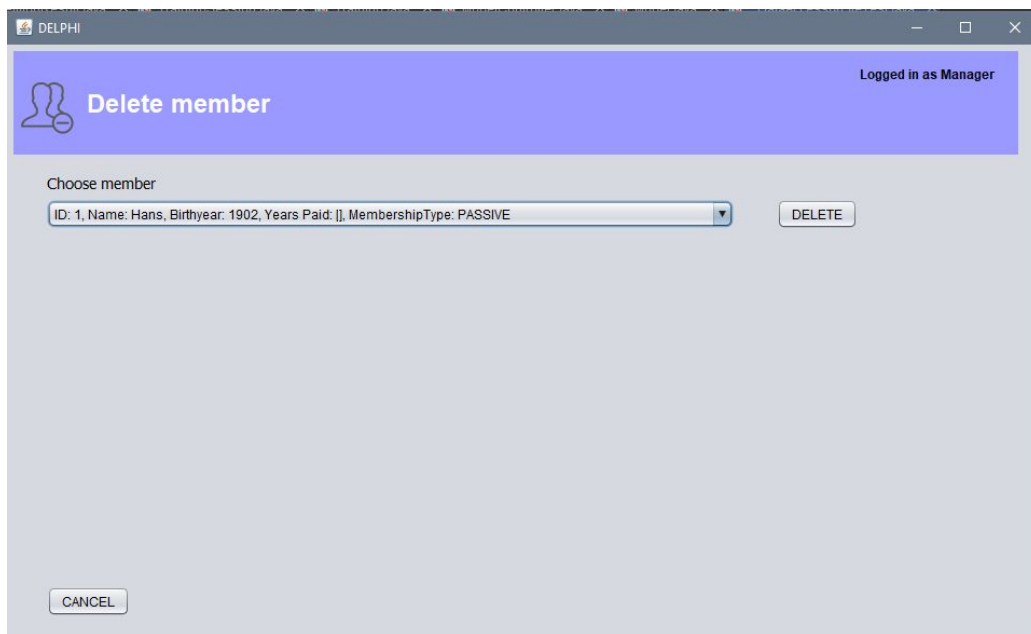
Birth year:  (YYYY)

Membership type: ☒ Passive ☐ Active

Activity type: ☐ Basic ☐ Competitive

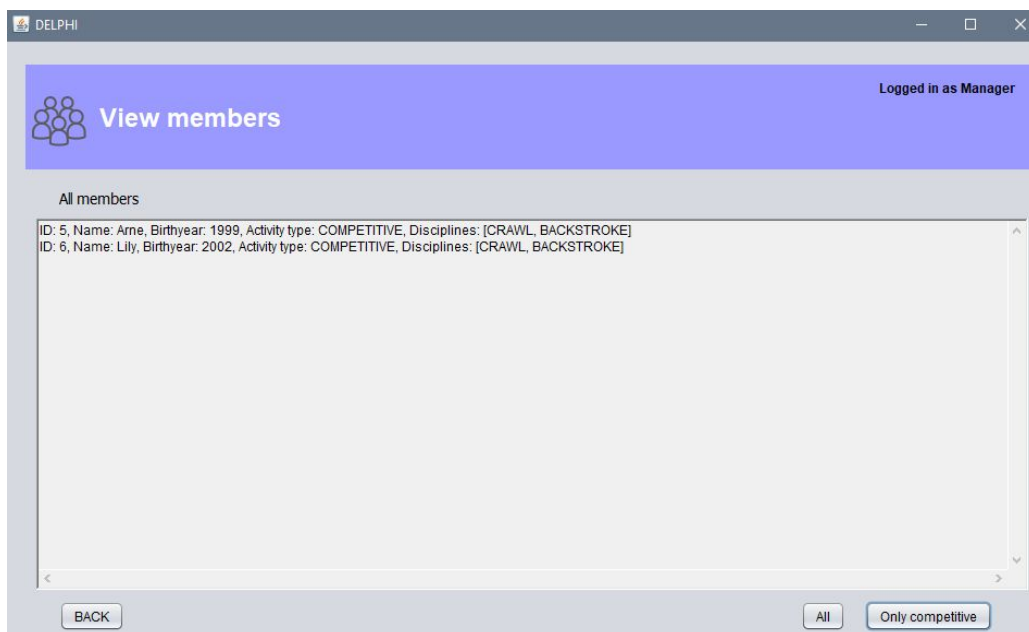
Discipline type: ☐ Butterfly ☐ Backstroke ☐ Breaststroke ☐ Crawl

CANCEL Confirm edit

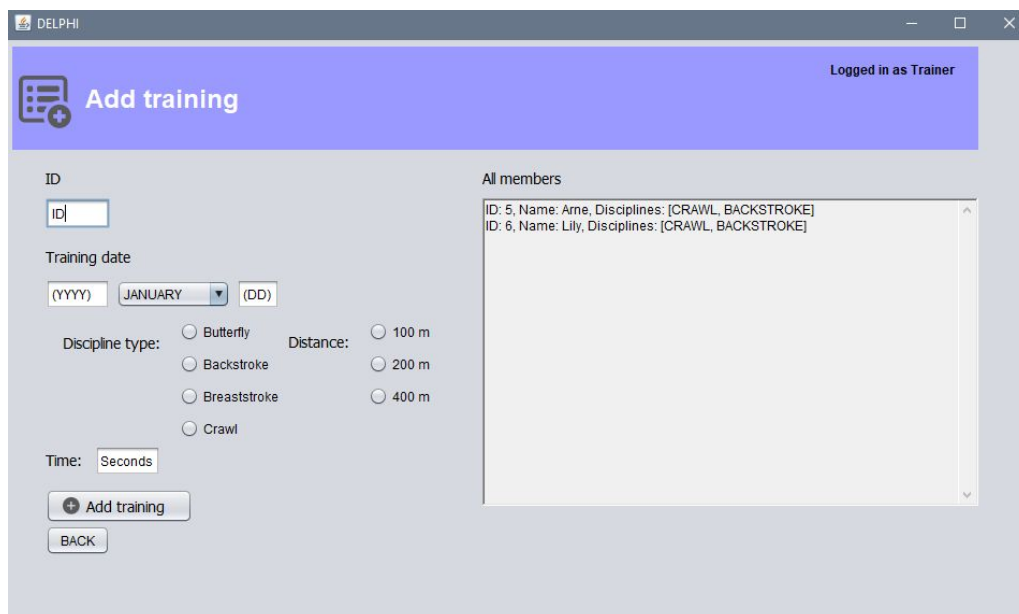


## Vis medlemmer





## Tilføj træningstid



## Tilføj konkurrenceplacering

DELPHI

Logged in as Trainer

### Add competition result

ID:  Placement:  Event name:

Competition date:  (YYYY)  (JANUARY)  (DD)

Discipline type: ☐ Butterfly ☐ Backstroke ☐ Breaststroke ☐ Crawl

Distance: ☐ 100 m ☐ 200 m ☐ 400 m

Time:  Seconds

All members

ID: 5, Name: Arne, Disciplines: [CRAWL, BACKSTROKE]  
ID: 6, Name: Lily, Disciplines: [CRAWL, BACKSTROKE]

## Se top 5 træning

DELPHI

Logged in as Trainer

### View top 5 members

Discipline type: ☐ Butterfly ☐ Backstroke ☐ Breaststroke ☒ Crawl

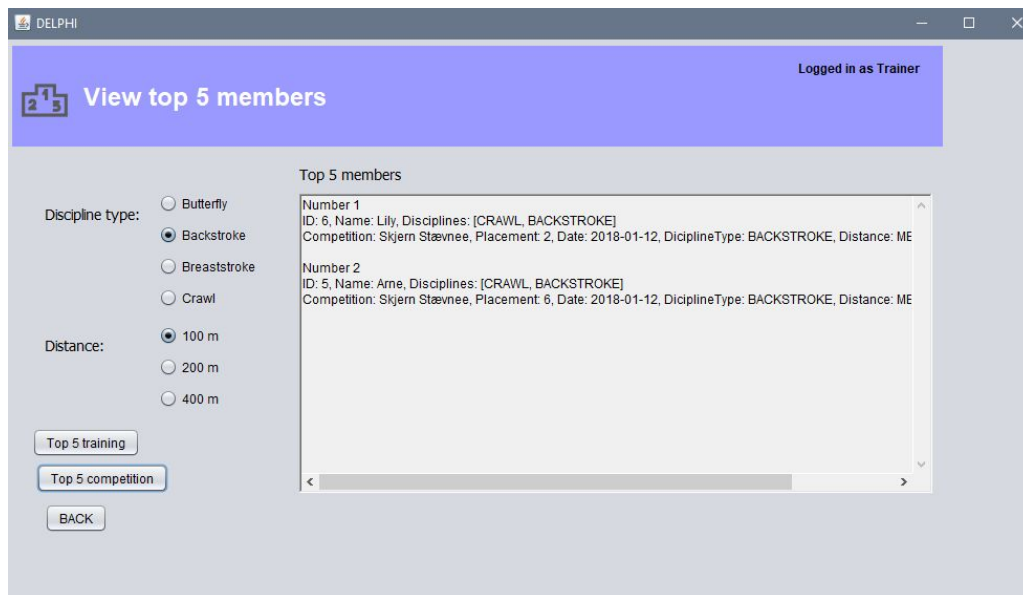
Distance: ☒ 100 m ☐ 200 m ☐ 400 m

Top 5 members

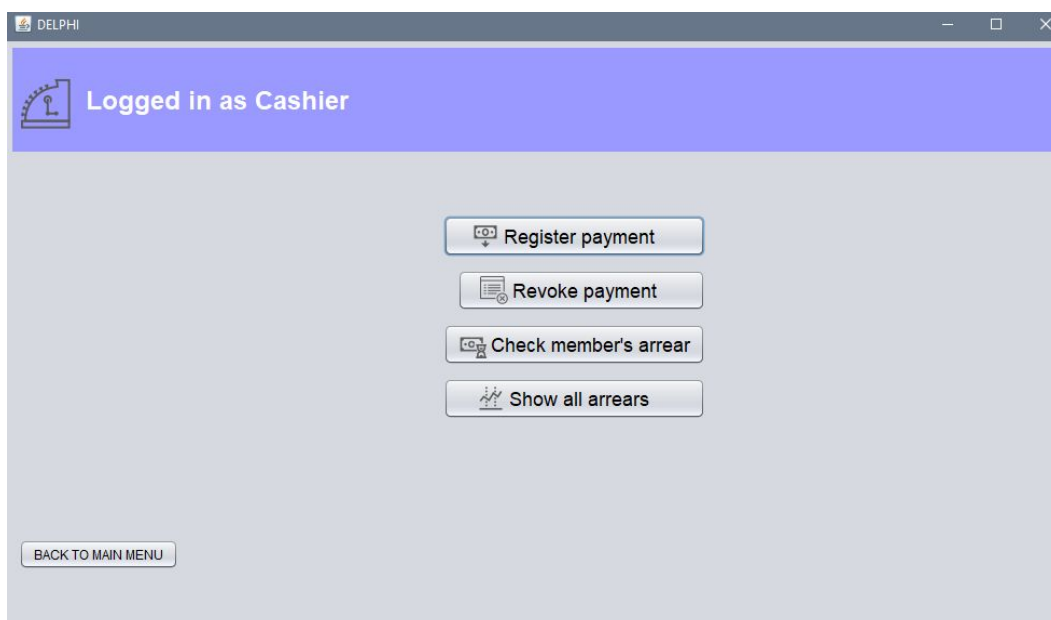
Number 1  
ID: 5, Name: Arne, Disciplines: [CRAWL, BACKSTROKE]  
TrainingSession: Date: 2018-04-22, DisciplineType: CRAWL, Distance: METERS\_100, Time: 365

Number 2  
ID: 6, Name: Lily, Disciplines: [CRAWL, BACKSTROKE]  
TrainingSession: Date: 2018-04-22, DisciplineType: CRAWL, Distance: METERS\_100, Time: 365

## Se Top 5 konkurrence



## Se restance





DELPHI

Single member arrear

Logged in as Cashier

ID

2

Current arrear

500

Show arrear

BACK

All members

ID: 1, Name: Hans, Years Paid: [2018]  
ID: 2, Name: Karl, Years Paid: [2017]  
ID: 3, Name: Signe, Years Paid: [2016]  
ID: 4, Name: Otto, Years Paid: [2018]  
ID: 5, Name: Arne, Years Paid: [2017]  
ID: 6, Name: Lily, Years Paid: [2018]

DELPHI

All arrears

Logged in as Cashier

All arrears

Total amount

3700

BACK

All members in arrear

ID: 2, Name: Karl, Years Paid: [2017]  
ID: 3, Name: Signe, Years Paid: [2016]  
ID: 5, Name: Arne, Years Paid: [2017]

## Registrér betaling

DELPHI

Registered in as Cashier

### Register payment

ID

Payment year

 (YYYY)

+ Register payment

BACK

All members

- ID: 1, Name: Hans, Years Paid: [2018]
- ID: 2, Name: Karl, Years Paid: [2017]
- ID: 3, Name: Signe, Years Paid: [2016]
- ID: 4, Name: Otto, Years Paid: [2018]
- ID: 5, Name: Arne, Years Paid: [2017]
- ID: 6, Name: Lily, Years Paid: [2018]

DELPHI

Registered in as Cashier

### Revoke payment

ID

Payment year

 (YYYY)

- Revoke payment

BACK

All members

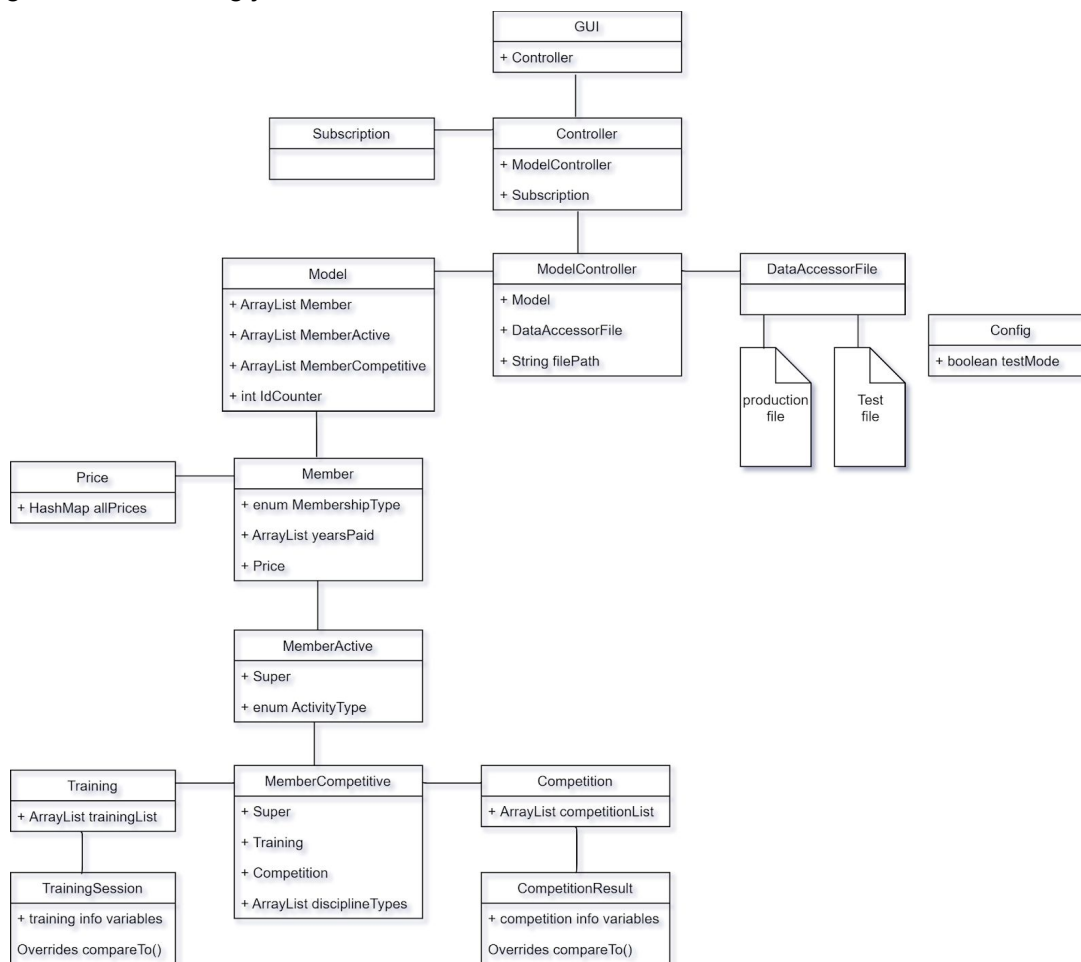
- ID: 1, Name: Hans, Years Paid: [2018]
- ID: 2, Name: Karl, Years Paid: [2017]
- ID: 3, Name: Signe, Years Paid: [2016]
- ID: 4, Name: Otto, Years Paid: [2018]
- ID: 5, Name: Arne, Years Paid: [2017]
- ID: 6, Name: Lily, Years Paid: [2018]

## Ikke-funktionelle krav

- Programmet indeholder junit tests af relevante klasser, i alt 40 junit tests. Disse er sat op i en TestSuite, som sørger for at de alle bliver kørt igennem.
  - Det er valgt at GUI ikke testes i junit-tests, men i stedet er gennemtestet af udviklerteamet under og efter projektet.
  - Alle logiktunge klasser og metoder i resten af programmet testes i junit-tests der forsøger at emulere brugen af disse.
- Der håndteres exceptions alle steder hvor det er relevant, eksempler på dette:
  - `FileNotFoundException` og `IOException` i `DataAccessorFile`
  - `IOException` i `Controller`
  - `DateTimeException`, `IllegalArgumentException`, `NullPointerException` og `NumberFormatException` i `GUI`
- Der er taget højde for de mest almindelige brugerfejl af programmet i kodningen, f.eks.:
  - Eksisterer en fil ikke, oprettes der en
  - Indtastes der bogstaver hvor der bedes om tal, bliver brugeren gjort opmærksom på fejlen via en pop-up besked og anvises til at indtaste på ny
  - Der oprettes en separat test-fil, til test af programmet så brugerdata ikke overskriver produktionsdata

## Vedligeholdelsesvenlighed

På nedenstående forsimplede UML klassediagram ses alle programmets klasser, med undtagelse af enums og junit-tests.



- Navngivning af klasser, metoder og variabler er forsøgt at emulere virkelige objekter, så det er nemt at forstå fra udefrakommende. Eksempler på metodenavngivning:
  - `writeToFile()`, `getTopFiveCompetition()`, `getArrearSingleMember()`, `registerPayment()`, `createMemberActive()`, `getAllMembersInBasicMemberFormat()`

- Vores commits er forsøgt holdt korte og beskrivende for overskuelighedens skyld. Eksempel på commits:

```
Fixed tests with error, equals, toString, and IDcounter
Added guard in GUI to make sure competitive members always have at least one discipline. Added secondary .toString() to MemberCompetitive. Made member's idCounter static
Removed unnecessary code from GUI
Merging for push
Reworked memberTest, memberActiveTest, memberCompetitiveTest & fixed testEquals. Refactored Member & Controller methods.
```

- Der er skrevet [JavaDoc](#) i programmet for at skabe overskuelighed for alle klasse og metoder. Eksempel på dette ses på billedet nedenfor:

All Classes
Packages
Delfinen.data
Delfinen.logic
Delfinen.presentation

All Classes
ActivityType
Competition
CompetitionResult
Config
Controller
DataAccessorFile
DisciplineType
Distance
GUI
Member
MemberActive
MemberCompetitive
MembershipType
Model
ModelController
Price
Price.priceType
Subscription
Training
TrainingSession

### Constructor Summary

Constructors

Constructor and Description
<b>DataAccessorFile()</b> DataAccessorFile creates a file that holds all the program data.

### Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method and Description	
Model	<b>readFile(java.lang.String filePath)</b> This method reads the file specified in the filePath string and returns the file content as a Model object.	
void	<b>setFILENAME(java.lang.String FILENAME)</b> Sets the FILENAME	
void	<b>writeToFile(Model model, java.lang.String filePath)</b> This method takes a Model object and writes it to the file, specified by the filePath and the FILENAME.	

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Constructor Detail

DataAccessorFile
public DataAccessorFile()
DataAccessorFile creates a file that holds all the program data. If the Config testmode boolean is set to true, a separate file for testing is created.

### Method Detail

readFile
public Model readFile(java.lang.String filePath)
This method reads the file specified in the filePath string and returns the file content as a Model object.
Parameters:
filePath - File path (Default: empty = project folder)

## Modulært design

Programmet er opbygget så det kan udvides yderligere fremover uden at koden skal skrives om. Al data gemmes i Model klassen i programmet, så tilføjelser kobles nemt herpå. Det kunne f.eks. være:

- Tilføjelse af træner (inklusive udvidelse af kontingentberegningslogik). Denne ville få en ArrayList i Model-klassen ligesom de forskellige medlemsklasser, der ville komme en tilføjelse til Price alt afhængigt af evt. træner-rabat og træneren ville få sin egen klasse.
- Ligeledes kunne man forestille sig at svømmeklubben selv vil afholde konkurrencer. Dette vil også nemt kunne tilføjes til programmet uden ændringer i den eksisterende kode. Her vil der igen ligge en ArrayList på Model klassen og ligesom med medlemmer og en evt. træner vil denne få sin egen klasse med sin egen logik.
- Ønskes en tilføjelse af en database frem for filskrivning vil der skulle ændres en smule i koden, da programmet er bygget op specifikt til filskrivning/læsning, men meget vil stadig kunne genbruges. Model klassen fungerer som en de facto database i dette program og det er hele Model objektet der gemmes i filen, som programmet fungerer nu.
- Price-klassen er sat simpelt op, i tilfælde af at priserne skal ændres. Her kunne man tilføje set-metoder og oprette et admin-panel, der kunne håndtere dette.
- Subscription klassen arbejder ud fra `getAllMembersInBasicMemberFormat()` som nemt kan ændres for fremtidige member- typer eller ændringer.
- Der kan nemt tilføjes flere distancer, discipliner, medlemskabstyper og aktivitetstyper.

Alle klasser i programmet har et begrænset ansvar hvilket gør programmet overskueligt nu og fremover.

## Hjælpeklasser

- Vores 4 enum-klasser:
  - MembershipType, ActivityType, DisciplineType & Distance
- Config klassen, som udelukkende styrer om der skrives til produktions- eller testfil

## Projektets mangler

- Vores GUI driller lidt, især på skærmopløsninger større end 1600x900, hvor `buttons` og `JPanelHeader` rykker sig rundt efter egen vilje.  
Desuden ændrer knapper størrelse og position alt efter hvilken computer, programmet bliver kørt på (testet med 3 forskellige computere), selvom man ikke har ændret i noget af GUI klassens kode.
- Vores `toString()` kunne i visse dele af programmet godt være *lidt* pænere.
- Vi ville gerne have undersøgt muligheden for at rykke vores enum-klasser ind i en enkelt klasse, i stedet for at være opdelt i 4 som de er nu.
- Price og Subscription er to klasser for sig selv, mens at `calculateArrear()` metoden ligger på Member. Dette kunne godt ses som forvirrende udefra.
- Vi ville gerne have tilføjet et admin-panel, men da det gik ud over kravene, valgte vi at lade være. Her ville man bl.a. kunne ændre på nogle af de mere hard-coded værdier (priser, f.eks.) og også sætte programmet i test/produktions mode (se: Config-klassen).
- Der er skrevet Javadoc til GUI og alle dens metoder, men dette ses ikke i den genererede Javadoc.