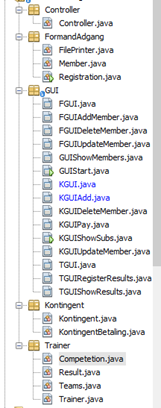
Delfin Program Documentation

This program is the final product of an assignment in the “Datamatiker” 1. Semester educational procedure. The program’s primary function is to assist the staff of an organisation. The purpose of the assignment was to design a GUI front-end and a .txt-based backend. The backend databases and the frontend GUI were programmed with the IDE Netbeans in the Java programming language. As part of the assignment our group was tasked to create a document explaining several main points of the program.

We started our programming by designing a UML Use Case diagram. Our diagram can be found in the appendix.



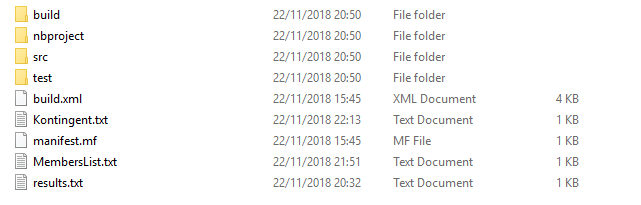
Most of the features specified in the Use Case Diagram are directly connected to the GUI classes in our program. We’ve shown this in the shape of the front letter of the GUI class name, e.g. FGUI is a screen reserved for the “Formand”, KGUI; the “Kasserer”. GUIStart and GUIShowMembers can be accessed by all the staff classes.

“Formand” in many ways has the same functionality as an administrator and has basic CRUD privileges, as well as access to the several of the other staff members’ lists. All these functions have been implemented.

“Træner”’s main function is to store information regarding the top swimmers of each team, as well as assigning them to competitions. These functionalities were only partially implemented. The screen called update members is missing.

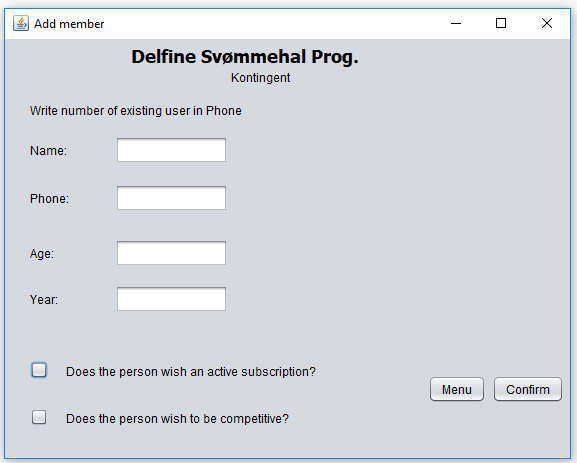
“Kasserer”’s main function is to store information about the members registered by the “Formand” and applying fees to them, depending on what year it is, their age and whether or not their subscription is active. All of these functions have been implemented, although they should be further polished in a real-life scenario.

Some important notations before using the program, would be how the information is stored. It was part of the assignment to use .txt files. Usually when you use software which require pathing, however, you’d want to define the path of in which such files would be saved. In our program, this information is stored in .txt files in the main folder of our program.



This information is necessary if the user wishes to manually reset the .txt databases. MemberList is dedicated to members in general, kontingent is dedicated to “kasserer” and results to “træner”.

The best example of something which should be polished, would be the kontingent add screen or KGUIAdd. Only phone and year has a direct influence on how you collect information from the MembersList, but because of how it was programmed; changing it would require restructuring of some core code.



JUnit tests

In our test class for the controller class we test our controller ability to register members by creating three instances of member, adding them to our registration method and then using the assertNotNull method to make sure that it works.

Through our JUnit test class called MemberTest we check that our CalcFee method creates three instances of member, expecting three results through what our logic in the calcFee method should end with. By using the assertEquals method to compare the expected value and the result our method returns, we can make sure that the values we get from our calcFee are correct.

With the TeamTest class we add four instances of member. Add them to an arraylist and through our addMemberToTeam method we use the integer values for age and the Boolean values for competitive, to determine which team each member should be on. We use the assertNotNull method four times. One for each team. With the JUnit test we have asserted that each team can return a member.

In our FGUIAddMember class in the GUI package we added an if statement, which helps the user by making sure they have entered a valid 8-digit phone number and that the name has at least two characters.

Documentation and Reusability

Despite the nationality of the users which the program has been intended for, the program variables, classes, comments and packages have been written in English.

For the GUI Forms, however, it was decided to use acronyms such as KGUI, TGUI and FGUI. These acronyms refer to the Danish translations of Cashier, Trainer and Chairman. This was done to keep it short and understandable. It can be discussed whether these acronyms are readable by future developers. Heavy use of comments should clarify each class’s purpose. We’ve used the domain and user case diagrams as inspiration to our methods and classes. See appendix for user case and Class diagram.

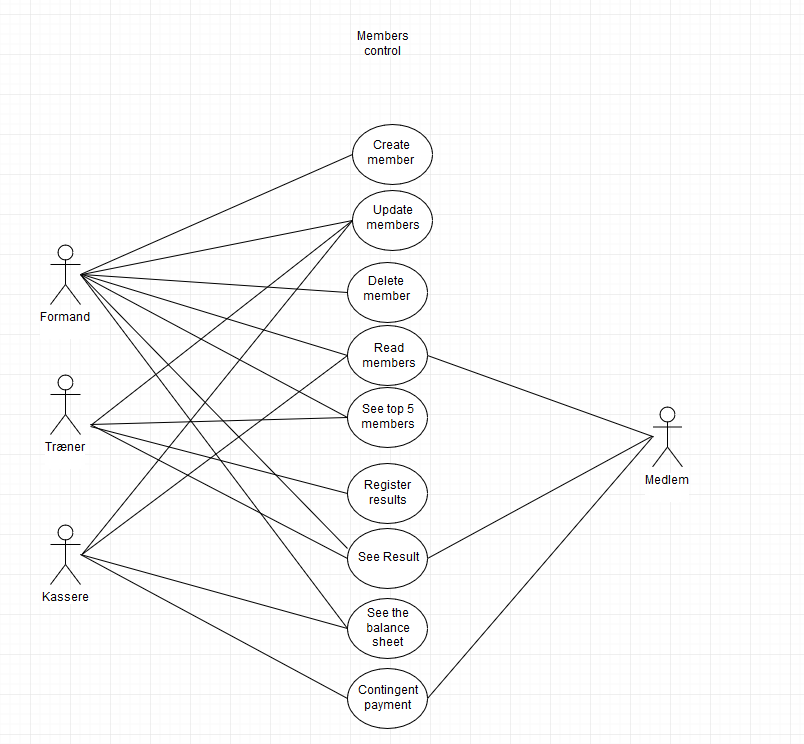
The work documentation can be seen through Github. Near the end of the work process, however, we used commit names which wouldn’t be understandable, if you weren’t part of the group. The commit name “.” means that same as last time. See appendix for a screenshot of the final Github commits.

Reusability of code plays an important role in the program. We have a class for each individual way of handling the logic of the tasks. Mostly the reuability of code is based around references and the way that we can use methods on instances of classes. We do, however, also have some static methods in one of the classes. The FilePrinter class is primarily used to communicate between a text file with a given path and the program itself. In the controller we use this communication, and a logical method of handling the information from the file to create and get objects to and from the given text file. We have compacted the code in the FilePrinter class as much as possible, so we ended up making the methods use a parameter given as a path to a file. This not only makes the code more reusable, but also easier to overcome, as that there are no longer several methods for communicating with each individual file, but instead there’s one method to communicate with files to a given path.

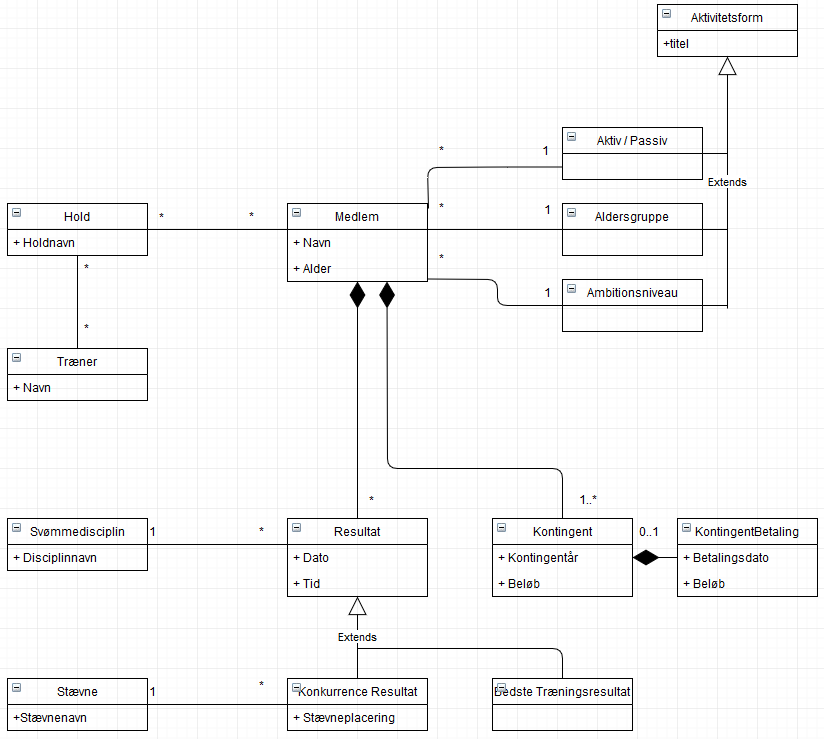
This also comes to show in the way that it has been implemented in the other classes as getters (in a way that we return the information in the text file in instances) such as the class handling results and showmembers.

**Appendix**:

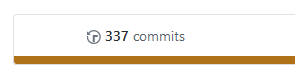
Use case diagram

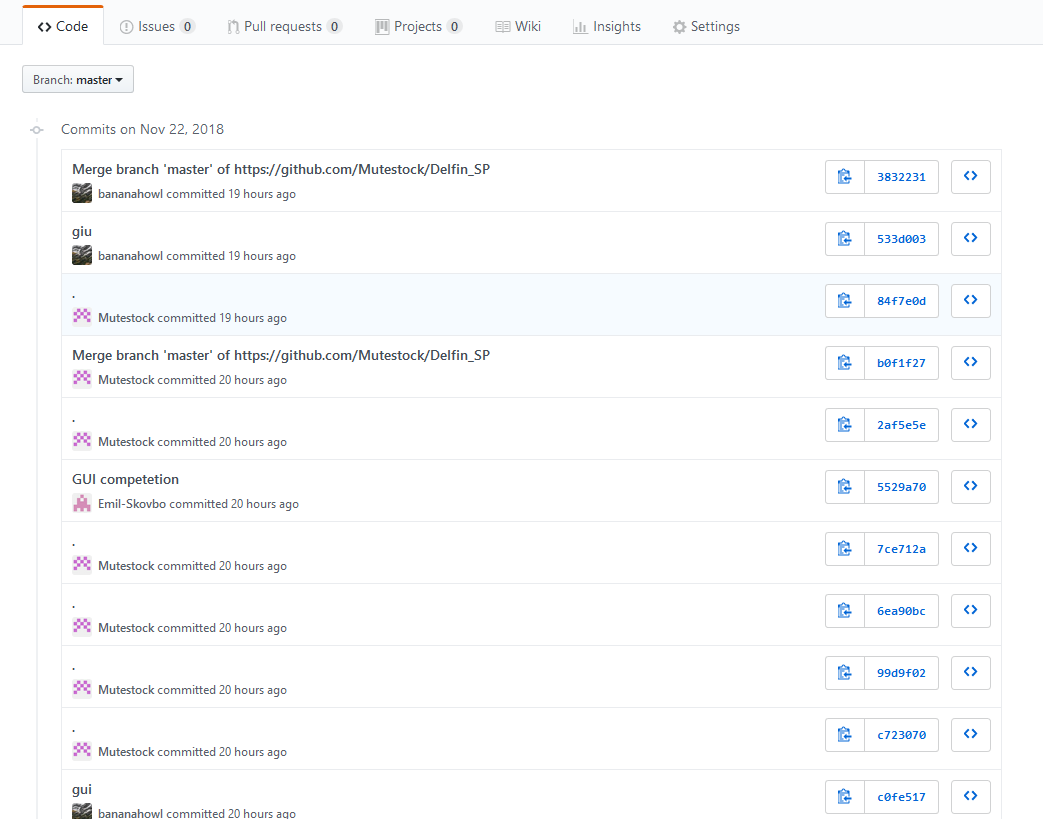


Domain / Class diagram.



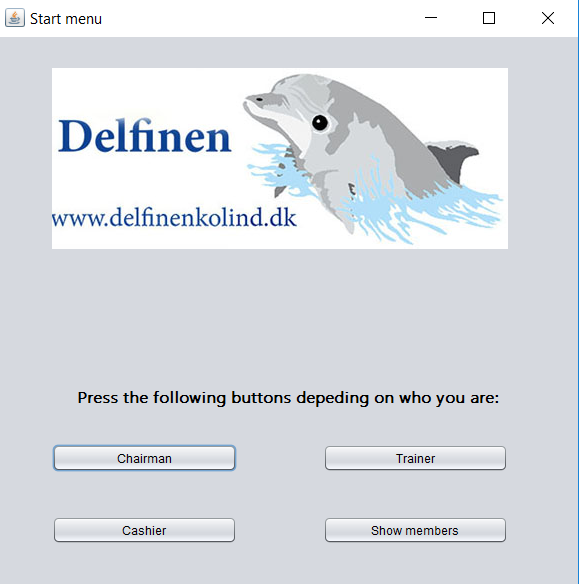
Final Github commits:

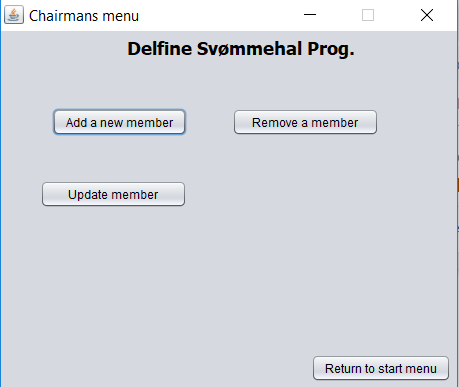




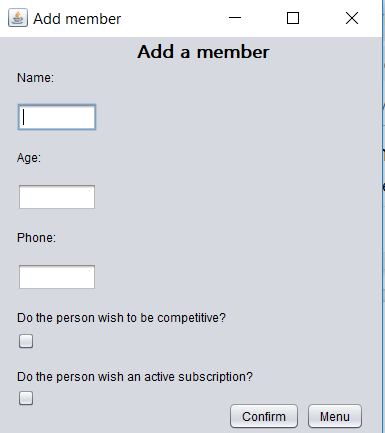
Screen shots of GUI:

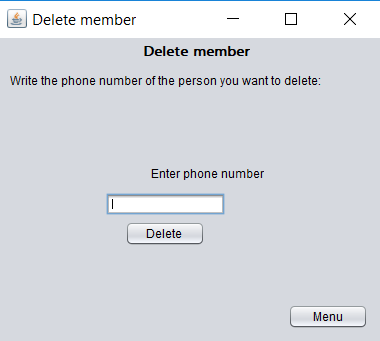
Start Menu:

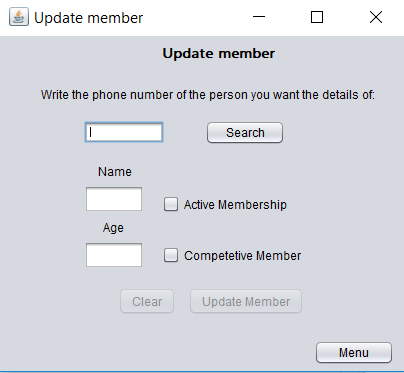
  
  
Chairman menu from startmenu



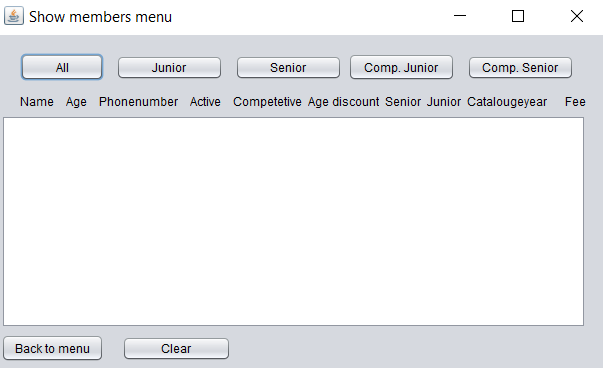
Chairman functions.



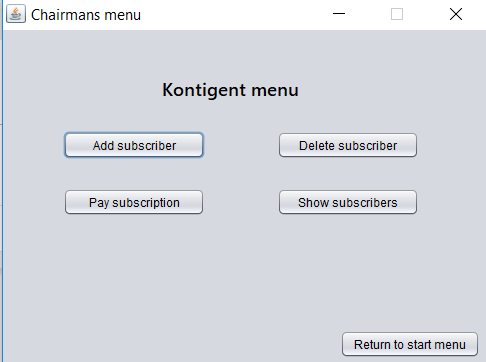




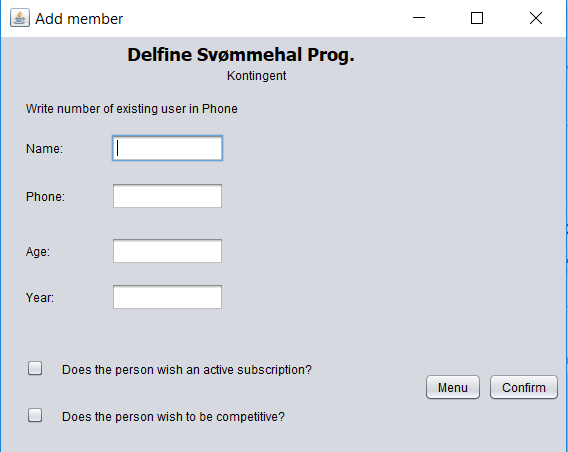
Show members GUI from Startmenu

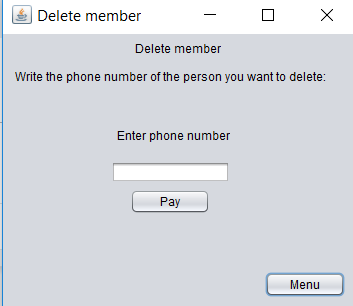


Kontingent menu from startmenu

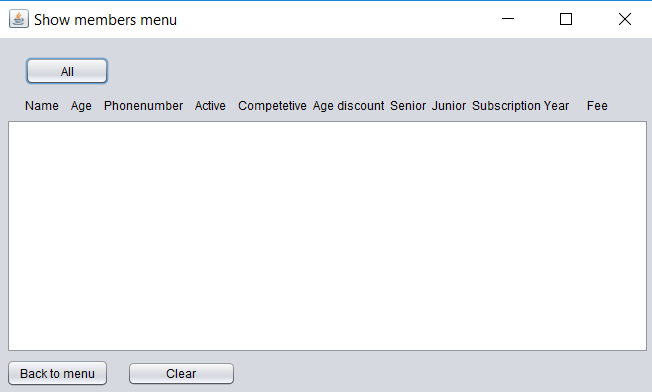


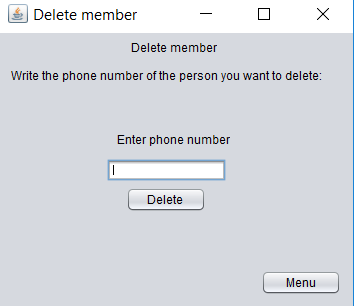
Kontingent functions





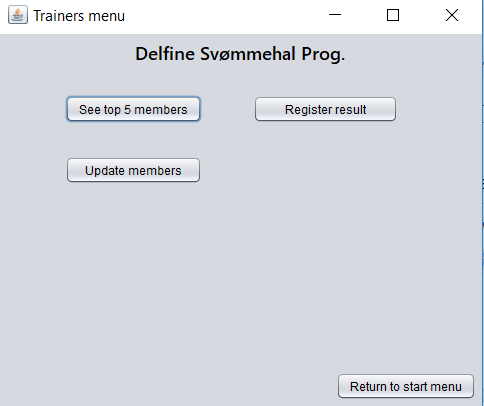
(edit: should be the written in the menu title and inside, that this is the Pay member menu.)





Trainer GUI:

Update members isn’t functional.



Trainer functions

