Bring ideas to life
VIA University College

ViaFit Fitness centre

SEP1 S18 – Software development

Mihai Draghiciu, 238971 

Eric Volmer, 273448 

Andrei Mungiu, 273473 

Simon Tirsgaard, 237434 

Ronalds Andris Kalnins, 260506 

Supervisor: Mona Wendel Andersen, Allan Henriksen

VIA University College

[28.706]

Software Engineering

1st semester, Spring 2018

08-06-2018

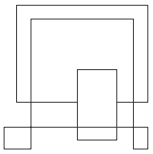
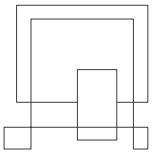


Table of content



Abstract

An abstract is a shortened version of the report and should contain all information necessary for the reader to determine:

1. *What are the aim and objectives of the project*
2. *What are the main technical choices*
3. *What are the results*

Frequently, readers of a report will only read the abstract, choosing to read at length those reports that are most interesting to them. For this reason, and because abstracts are frequently made available to engineers by various computer abstracting services, this section should be written carefully and succinctly to have the greatest impact in as few words as possible.

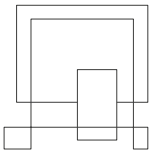
Although it appears as the first section in a paper, most report writers write the abstract section last.

Cf. (Dawson 2009, p.195).

The aim of this project is to enable the fitness center to keep track of all relevant data.

Project objectives include but are not limited to:

1. Storing all basic information about members, instructors, gym events and any data relevant to them in binary files.
2. Allow the user to filter, categorize and search through the available data by relevant criteria.
3. Allow the user to edit any details about any element while seamlessly restricting him from causing a system failure or data corruption.
4. Display in a user-friendly manner the application capabilities on screen and make core functionality easy to access.



1 Introduction

ViaFit is a fitness centre run by Bob Sixpack and his father. Until now the fitness centre was run in an old fashion way where every information required would be put on paper and the events would be written on a blackboard at the entrance, so members could see what was in store for the next month.

Bob Sixpack started to take up the responsibility and run the gym therefore some changes are required such as switching from the inefficiency of pen and paper to a more efficient solution which is a digital one.

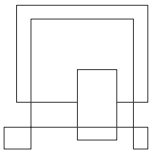
The reason the owner wishes to switch from storing the information on paper to storing it in a digital fashion is safety and efficiency. After a fire incident in the past where most of the information regarding the members was lost, the gym suffered greatly. The blackboard incident happened when a group of kids decided to erase events that were on the blackboard replacing them with drawings, this made the owner rewrite them by memory and it created problems like instructors being placed in the same day or being unavailable or straight up missing events altogether.

Therefore, after an interview, the owner requires a program that will run offline at the gym that can store information about customers, instructors and the overall gym management with the possibility of editing this information at any given time so issues of the past can stay in the past.

The main purpose is to enable the gym to manage information regarding members, instructors and events without hassle or difficulty.

Delimitations regarding the application are:

1. No login interfaces. (there will always be someone operating the computer so no login is required since the only person that can access that information will be a certified one)
2. The system is offline. (therefore there is no requirement for the project to have an active internet connection since the registration/editing of the members will be done live at the scene)
3. No implementation of a payment system. (that duty is fulfilled by a 3rd party so the application does not require a payment method)



2 Requirements

The requirements of the application itself are as follows:

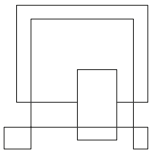
- Allow the user (instructor/owner/cashier) to add and edit member information. (Name, email, phone number and type of membership normal or premium)
- Allow the user to add/edit instructor contact information and what type of class the instructor can teach and if the instructor is available.
- Allow the user to create/edit a class that contains information about the type of class and when the said class will take place with an instructor.
- Allow the user to schedule and edit events. (set a time and date for the event as well as the number of members allowed to join)
- Allow the user to save this information and recall it whenever it is necessary.

The user will be able to manage the gym's daily and monthly routine through this application allowing for a more stress free environment.
The user will be able to create and set new events in order to place them on the blackboard (the owner wishes the blackboard to remain) only this time in case that information is erased it can easily be searched for and added within a matter of minutes.
Members can be easily added/removed to and from events, switched from a regular membership to a premium one and vice versa.
Information about the members will be kept private with only the eligible person being allowed to have access to it.

2.1 Functional Requirements

The staff member (instructor/owner) will have access at all times to the computer the runs the application

The functional requirements of the application will be the following:



- The application will allow the staff member will be able to register a new instructor with all the information it requires.

- The application will allow the staff member to edit a current instructor's information.

- The application will allow the staff member to search for an instructor by name, phone number, ID, email.

- The staff member will be able to register a new member.

- The application will allow the staff member to see all registered members.

- The application will allow the staff member to edit a member's information.

- The application will allow the staff member to search for a member by name, ID, phone number or email.

- The application will allow the staff member to set a type of membership (premium or regular) for the member.

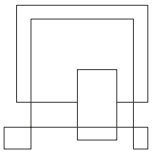
- The application will allow the staff member to add/remove events.

- The application will allow the staff member to edit an event.

- The application will allow the staff member to search for an event by name, type.

- The application will allow the staff member to edit the schedule.

- The application will allow the staff member to assign instructors to events.



2.2 Non-Functional Requirements

Non-functional requirements will be:

Security:

- Create, Read, Update and Delete levels.
- Access permissions for application data may only be changed by the system's administrator.
- Inactivity timeouts.
- System data backed every X hours and copies and stores it.

Audit:

- System must maintain full traceability of the actions.
- Audited objects are defined.
- Audited database fields, which data field requires audit info?
- File characteristics, size before, size after structure.
- User and transactional time stamps.

Capacity:

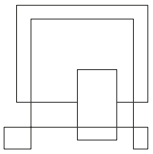
- Storage (memory/disk) volume of data the system will page / persist at run time through disk.
- Year-on-Year growth requirements (users, processing and storage).

Performance:

- Response times: application reload time, refresh time.
- Processing times: functions, calculations, imports, exports.

Availability:

- Hours of operation 24/7.



- Holidays, maintenance times.
- Location of operation: only available at the desk in the gym.

Reliability:

- The response of the application should be 0.5 seconds in 99% of the time.
- Mean time to recovery: in case of failure the system will require a restart that can take 1-3 minutes.

Usability:

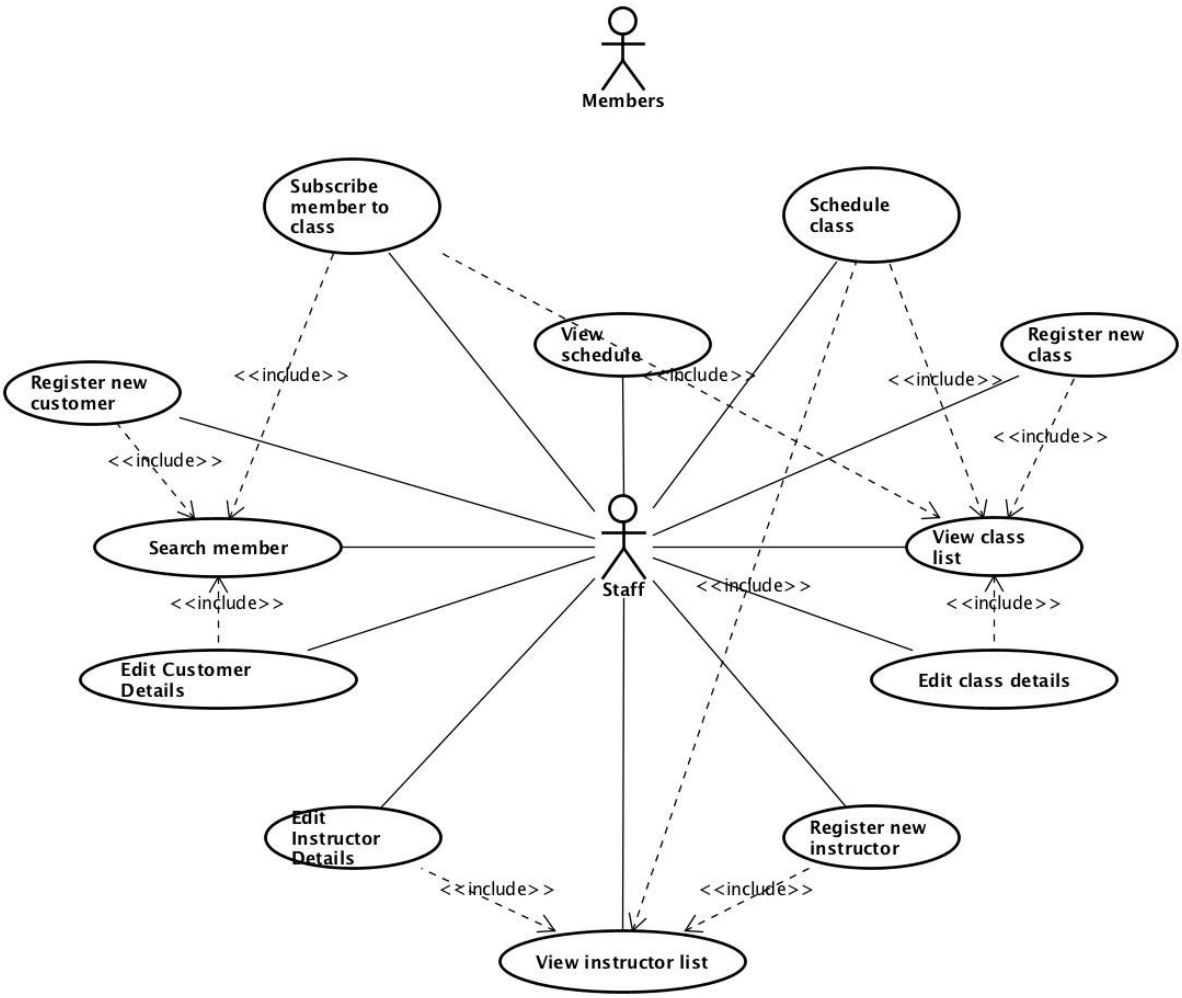
- Easy to use for the staff members.

Documentation:

- User Documentation.
- System Documentation.
- Training Material.

3 Analysis

3.1 Use Case Description Diagram.



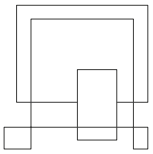
All diagrams can be found in Appendix A.

View Schedule: Shows the schedule.

Schedule Class: Add a reservation the scheduled class.

Register New Class: It creates a new class.

View Class List: Shows all the current classes.



Edit Class Details: Allows the editing of classes as type, number of members, instructors.

Register New Instructor: Creates a new instructor Id with the information required.

View Instructor List: Shows all the current registered instructors.

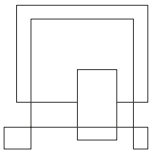
Edit Instructor Details: Allows the editing of instructor information (name, phone number, address, email)

Edit Customer Details: Allows the editing of customer details (name, phone number, email).

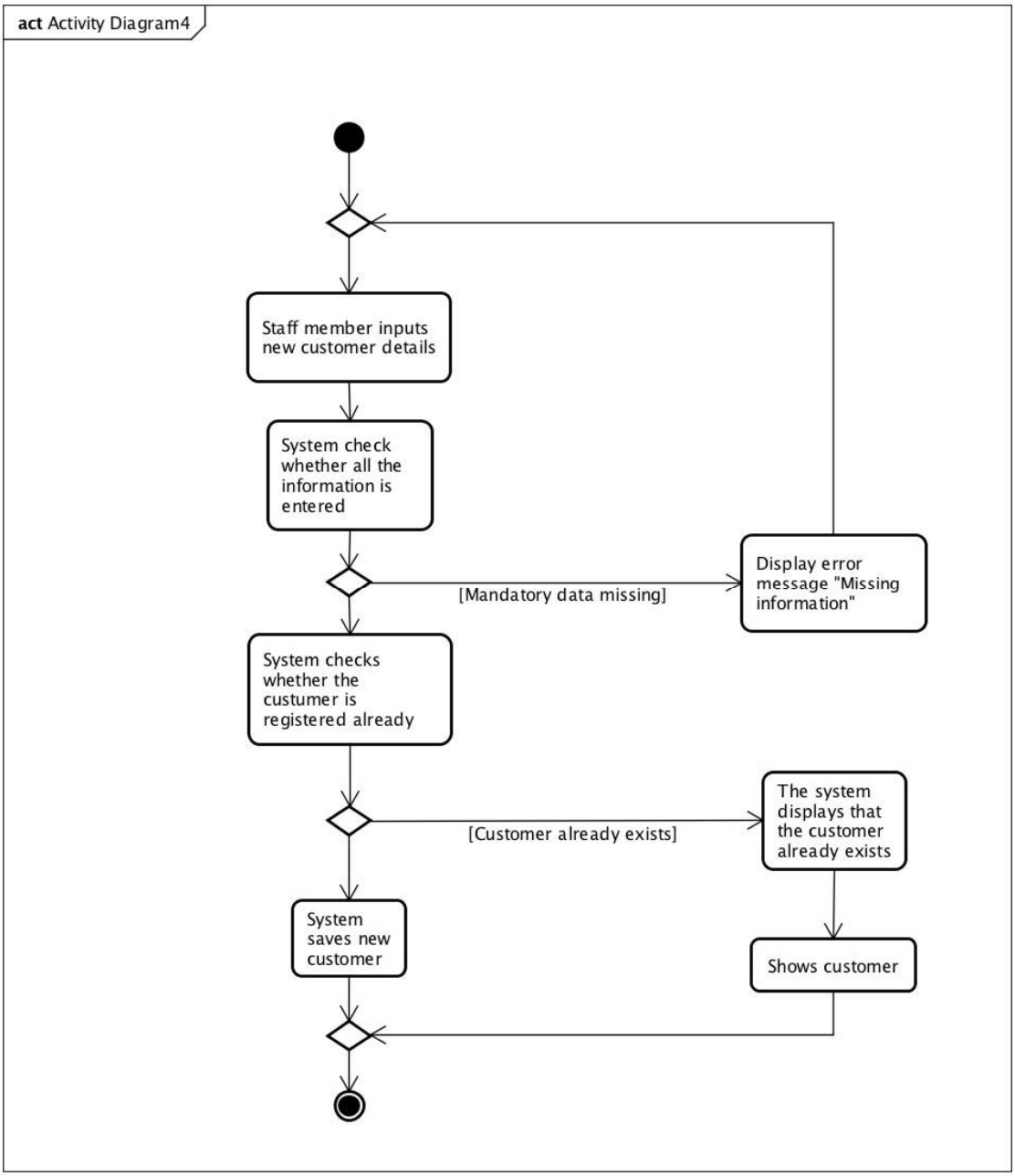
Search Member: Allows the possibility to search for a member based on name, ID, phone number, email, or part of the name, phone number, ID. (it will return all the members containing part of the input information)

Register New Customer: Creates a new member containing the information required. (name, phone number, email)

Subscribe Member to Class: Adds the member to the event.



3.2 Activity Diagram for registering a new customer.



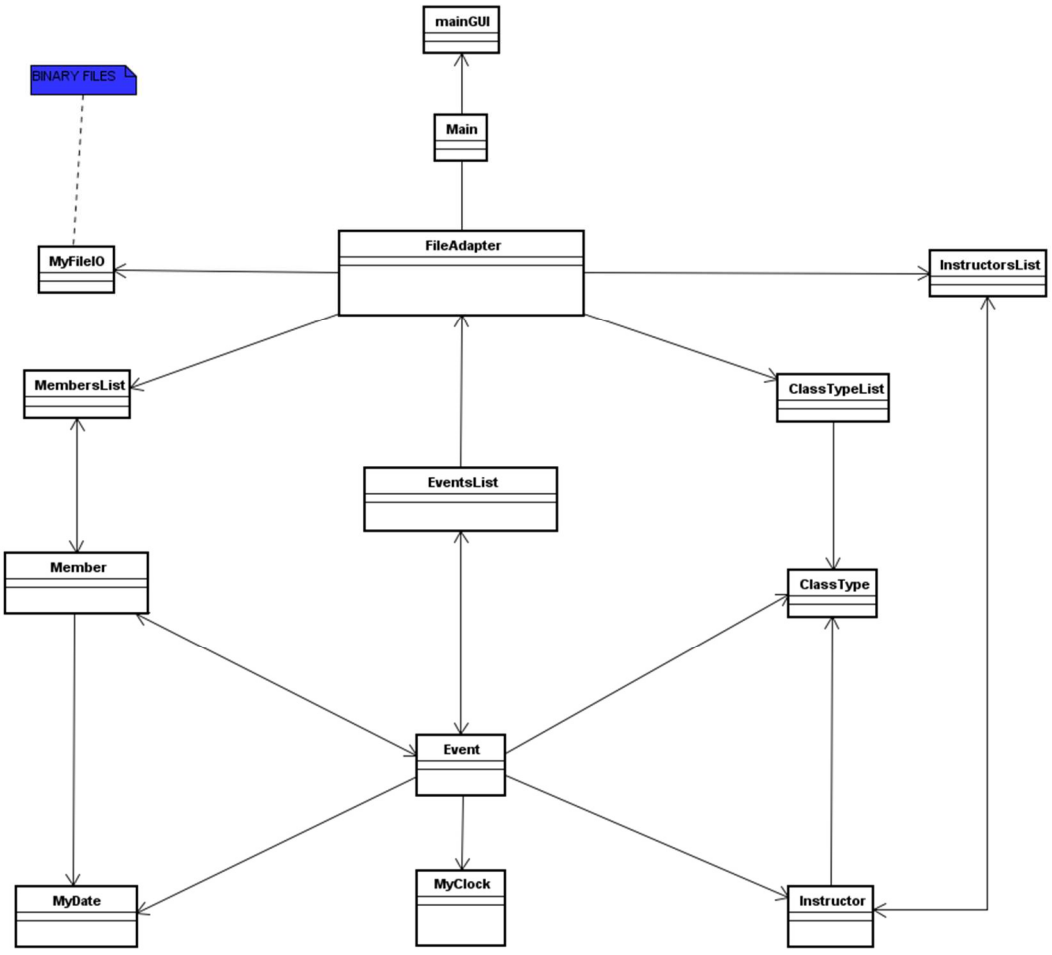
The activity diagram to register a new member is a very important one as it allows the staff member to register a new member and assign a type of membership. The role of this diagram is to show a step-by-step workflow of the registration component.

The staff member inputs the new customer's data, the system will display errors if there is information missing. The system will also check if the member has been

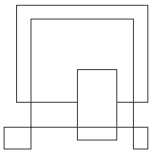
registered previously, if this is the case it proceeds to either show the customer is already registered. If the customer was not previously registered the system will save the new member and show it on the screen.

See all the activity diagrams in Appendix-A Activity Diagrams.

3.3 Model diagram class.



The diagram shows us the connection between the classes required by the program to run and fulfill the requirements. The classes have been separated in order to



follow a more object oriented approach and to make maintenance easier should something happen.

4 Design

4.1 ViaFit design.

The most important part of the project was the **software design**. The application that is being developed will interact with staff members therefore, the interface must be very simple, fast and easy to navigate through.

The program is based on two parts: **Java** (programming language) and **GUI** (Graphic User Interface). Additional software was used for planning (Astah professional) as well merging and putting the code together from all the group members(GitHub).

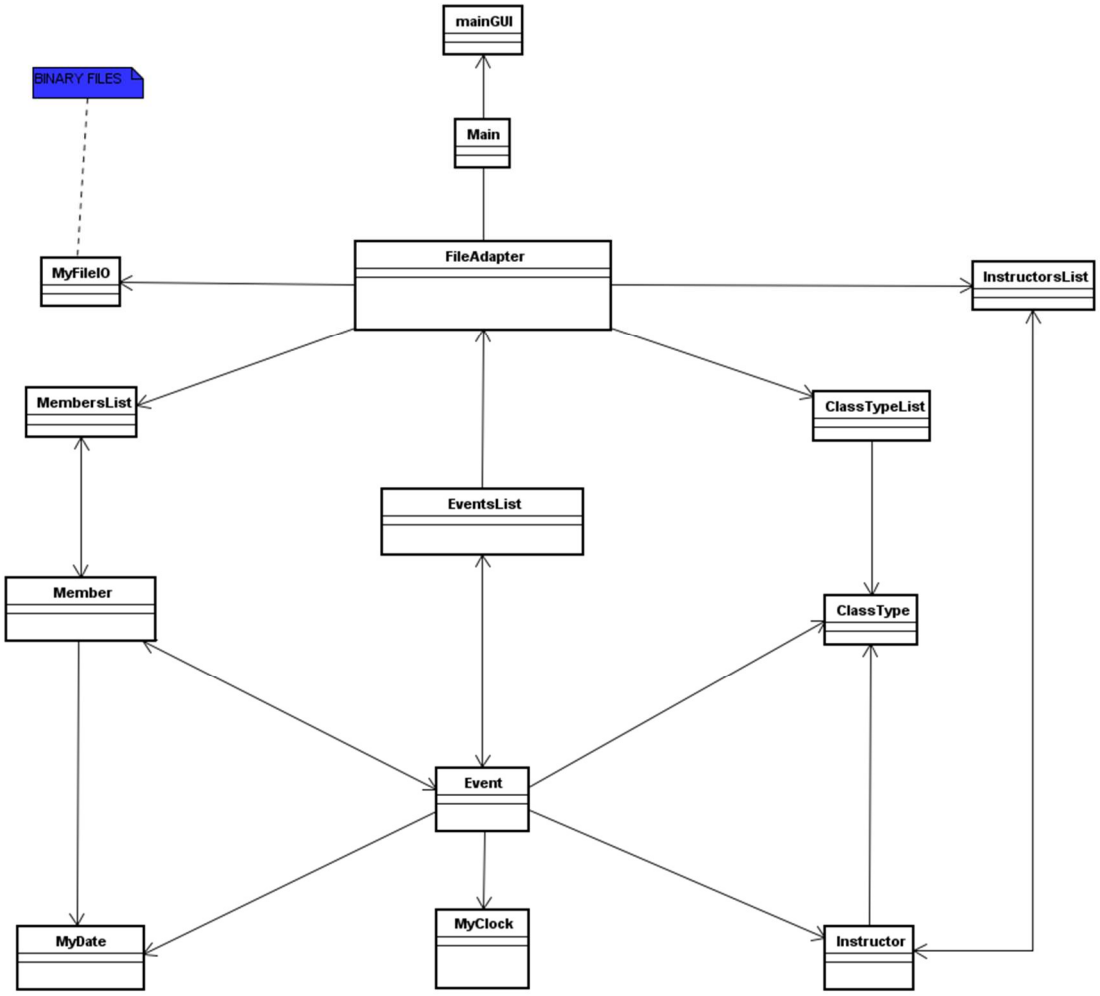
Java Classes.

With Java the application is based on classes that build up the whole interface and can assign the functionality to it, as well as create methods that display information, store information that will be created by the staff member in regards to the costumer.

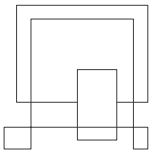
GUI (Guide User Interface)

The **GUI** is stored in separate classes, therefore, all the repetitive methods needed for implementing the buttons, tags, text fields and other graphical user interface related classes are not mixed with other classes so it's easier to program and maintain.

4.2 UML Class Diagram Design.



The UML Diagram above illustrates the structure of the system.



The event class contains all methods involving data manipulation about the members/instructors (name, phone number, email address) as well as adding or removing them to and from events. It also contains methods that return information about the classes (class types, date, time, maximum number of members attending).

Event class is required to store all the information about events like time and date that comes from MyClock and MyDate classes to save it in specific format.

Instructors/Members that are assigned to the event, as well as ClassType are classes that create separate instances of themselves in order to save and add to the desired event.

List Classes: MemberList, ClassTypeList, InstructorList are used to store class constructors into ArrayLists (in order to have the possibility to always expand). It allows for an easier upgrade, should the situation require it.

The FileAdapter class is storing the methods that save the information input by the staff member and is responsible for updating the events, information about the members, instructors daily.

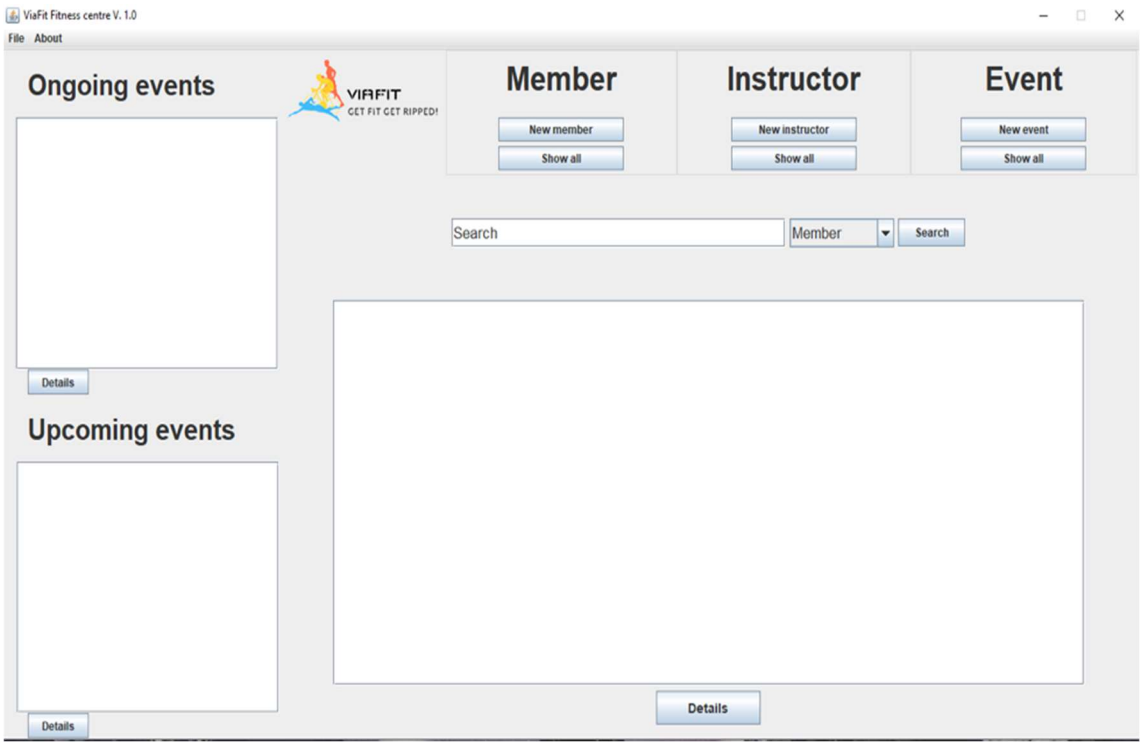
MyTextFileIo is the class responsible for writing the information to a binary file. It saves all the information input by the staff member regarding everything (events, members, instructors, schedule).

The MainGUI contains the life of the interface from the aspect of it like JPanels, JButtons, JLabels etc. and functionality and methods required to perform the needed actions.

The Main Class is the one that puts them all together and starts up the interface allowing the staff members to interact with the application. It is also responsible for updating the interface every 1 minute in order to correctly display upcoming/ongoing events.

See the complete UML Diagram in Appendix-C UML Class Diagram.

4.3 GUI Design.



ViaFit uses a simple and understandable design. It has a layout of buttons at the top of the screen that return the information requested in the white square be it information about instructors, members, events or the schedule.

Each button performs a specific function and works according to the Use Case Descriptions. The names of the buttons are similar to those of the Use Case Descriptions in order to be easily identifiable.

See all preliminary windows in Appendix-B Application Guide.

5 Implementation

During the during the development of the system we encountered many challenges. With the limited experience with developing GUI interfaces, but at the same time big ambitions for the GUI to look professional, the team gained a great deal of knowledge from supervisor Allan Henriksen and the web. Challenging functionality such as sorting ArrayLists based on specific criteria, automatically updating the interface based on a set time, also changing the interface when a window gained focus provided a great learning experience.

```
public void updateUpComingEventsList()
{
   updateEventsList();
   upComingEventsList.clear();
   int maxUpcomingEvents = 30;
   GregorianCalendar calendar = new GregorianCalendar();
   MyClock currentTime = new MyClock(calendar.get(GregorianCalendar.HOUR_OF_DAY), 0, 0);
   ArrayList<Event> temp = eventsList.getEventsList();
   MyDate today = MyDate.today();
   Event currentEvent;

   MyClock thisEventStartTime;
   MyClock thisEventEndTime;

   MyDate thisEventStartDate;
   MyDate thisEventEndDate;

   for (int i = 0; i < temp.size(); i++)
   {

      currentEvent = temp.get(i);
      thisEventStartTime = currentEvent.getStarTime();
      thisEventEndTime = currentEvent.getEndTime();
      thisEventStartDate = currentEvent.getStartDate();
      thisEventEndDate = currentEvent.getEndDate();

      boolean eventIsUpcomingYears = thisEventStartDate.getYear() > today.getYear();
      boolean eventIsUpcomingMonths = thisEventStartDate.getMonth() > today.getMonth();
      boolean eventIsUpcomingDays = thisEventStartDate.getDay() > today.getDay();
      boolean eventIsUpcomingHours = thisEventStartTime.getHour() > currentTime.getHour();
      boolean containsEvent = upComingEventsList.contains(currentEvent);
      // adding future events (30 max)
      if (upComingEventsList.size() < maxUpcomingEvents && !containsEvent)
      {
         if (eventIsUpcomingYears)
            upComingEventsList.add(currentEvent);
         else if (eventIsUpcomingMonths)
            upComingEventsList.add(currentEvent);
         else if (eventIsUpcomingDays)
            upComingEventsList.add(currentEvent);
         else if (eventIsUpcomingHours)
            upComingEventsList.add(currentEvent);
      }

      boolean eventWasBeforeThisYear = thisEventEndDate.getYear() < today.getYear();
      boolean eventWasBeforeThisMonth = thisEventEndDate.getMonth() < today.getMonth();
      boolean eventWasBeforeThisDay = thisEventEndDate.getDay() < today.getDay();
      boolean eventWasBeforeThisHour = thisEventEndTime.getHour() < currentTime.getHour();

      // removing old events
      if (upComingEventsList.contains(currentEvent))
      {
         if (eventWasBeforeThisYear)
            upComingEventsList.remove(currentEvent);
         else if (eventWasBeforeThisMonth)
            upComingEventsList.remove(currentEvent);
         else if (eventWasBeforeThisDay)
            upComingEventsList.remove(currentEvent);
         else if (eventWasBeforeThisHour)
            upComingEventsList.remove(currentEvent);
      }

   }

   // sorting events
   Collections.sort(upComingEventsList, new Comparator<Event>()
   {
      @Override
      public int compare(Event eventLowerIndex, Event eventUpperIndex)
      {
         return eventLowerIndex.compareTo(eventUpperIndex);
      }
```

Method for updating the upcoming events to only contain the nearest 30 events and sort them with earlies first.

```
// updating the bigInfoBox whenever mainGUI gains focus
addWindowFocusListener(new WindowAdapter()
{
    public void windowGainedFocus(WindowEvent e)
    {
        try
        {
            Thread.sleep(500);
        } catch (InterruptedException e1)
        {
            e1.printStackTrace();
        }
        updateBigInfoBox(searchOption.getSelectedItem().toString());
        updateOnGoingEventsArea();
        updateUpcomingEventsArea();
    }
});
```

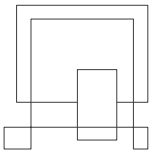
Method for updating the search field in the main screen whenever you open the app or switch to the app.

```
// sorting events
Collections.sort(upComingEventsList, new Comparator<Event>()
{
    @Override
    public int compare(Event eventLowerIndex, Event eventUpperIndex)
    {
        return eventLowerIndex.compareTo(eventUpperIndex);
    }
});
```

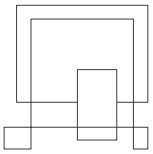
Up here we are sorting the ArrayList<Event> by first implementing "Comparable" to all involved Objects in the process of comparison, overriding the compareTo() methods inside each object and using those overridden compateTo() methods with the override compare() method. This is an example of how we tried to implement more than what was expected from us.

Libraries used:

- import java.io.EOFException;
- import java.io.FileNotFoundException;
- import java.io.IOException;
- import java.util.ArrayList;
- **import java.util.Collections;** - used for sorting Objects of custom type by applying the overridden method from Comparator on Classes that implement "Comparable<Type>"
- **import java.util.Comparator;** - used for creating a customized way to compare different objects such a an "Event" which must be compared according to some other objects inside it which also have overridden compareTo() methods.



- **import java.util.GregorianCalendar; -** used for pulling out details about the date in integers
- **import java.io.Serializable;** - used for saving objects to binary files



6 Test

The groups White Box test period started when the “On Going Events” and “Up Coming Events” functionality was completed. When testing, the group detected many problems which were less related to how the code has been written and more related to where the code was being called to take action.

During the White Box testing period the group has learned many valuable lesson about Object Oriented Programming, and even though there was not enough time to address all questions, the most important one have been solved by the method of trial and error combined with valuable insights from teachers.

It has been noted that Reference Types can behave in manners which we would only expect the primitive types to behave, according to what we have studied. Even though the software bugs caused by such behaviour have been addressed the team will take more interest in fully understanding why such behaviour occurred.

As the deadline approached and more group members finalized their core tasks, the team gradually shifted towards the black box testing, where team members that did not know the code behind a specific functionality where testing it and writing down their observations.

To a large degree, every testing activity has been coordinated with the help of the valuable notes from the requirements section, which helped the team use the already short amount of available time in an efficient and effective manner.

Therefore, Group 1 considers that all major software mistakes have either been solved or have been observed and noted.

6.1 Test Specifications

1 Introduction:

This section provides an overview of the entire testing procedure. It describes both the test plan and the test procedure.

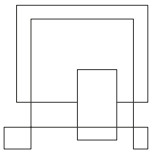
Goals and Objectives – To streamline as much as possible the process of software quality assurance by defining clear steps and expected outcomes.

Statement of Scope – The testing must be conducted on all aspects of software that are responsible for reading, writing and updating information in the User interface or inside the binary files.

Major constraints – The lack of in depth knowledge about certain Java libraries which have not been studied during the course but which the group was forced to use due to software requirements.

2 Test Plan:

The overall testing strategy is to test core functionality in the early White Box testing and gradually shift towards testing non-core elements in the black box test phase.



Unit Testing - Does the component restrict further progress in core elements which help create a core requirement? If Yes, start element core testing, else leave element for black box testing phase.

Integration testing

a. Finalize white box test phase of core elements independent functionality
b. Proceed to white box test phase of core elements functionality when integrated to other essential core elements of the application

Validation testing - Core functionality must be accessible and usable as intended from the user interface.

3 Test Procedure:

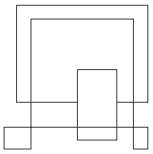
Software components to be tested:

- The upcoming events list.
- The ongoing events list.
- The ability to view detail about any instructor, member or event.
- The ability to create a new instructor, member or event.
- The ability to delete an instructor, member or event.
- The ability to edit and instructor, member or event.
- The ability to subscribe and unsubscribe members and instructors to an event.
- The ability to block a user from inputting wrong data into the required field when creating a new object or editing an existing object.
- The ability to read and write updated details about the instructors, members and events into binary file.

Testing procedure – Every components testing must begin in the white box test phase, transition towards the black box test phase and only finalize after the black box test phase has been fully completed.

Unit test cases

TEST ID	**PURPOSE**	**INPUT**	**Expected OUTPUT**
1	Test the behaviour of ArrayList<Member> when called before or after core software functions	Add member to array list before core function	Core function works with updated array list and gives an output that includes the core interaction with the added member



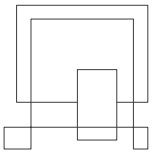
2	Test the behaviour of ArrayList<Instructor> when called before or after core software functions	Add instructor to array list before core function	Core function works with updated array list and gives an output that includes the core interaction with the added instructor
3	Test the behaviour of ArrayList<Event> when called before or after core software functions	Add event to array list before core function	Core function works with updated array list and gives an output that includes the core interaction with the added event

4 Validation testing

TEST ID	**PURPOSE**	**INPUT**	**Expected OUTPUT**
1	Test the ability to create a new member	Member details	Home screen with saved member details in binary files
2	Test the ability to create a new instructor	Instructor details	Home screen with saved instructor details in binary files
3	Test the ability to create a new event	Event details	Home screen with saved event details in binary files
4	Test the update of data	Time tick	Updated data
5	Test the sorting of data	Unsorted list of data	Sorted list of data
6	Test the ability to assign instructor or member to event	Event with un assigned member or instructor	Event with assigned member or instructor

5 Test records keeping and test log

White box test phase: test record keeping is only temporary for white box test phase as the developer will immediately use observation to address errors and thus save time.



Black box test phase: test records must be kept on paper as hand written notes and forwarded to the developer responsible for that functionality with any existing feedback and notes.

Test logs White Box Test Phase (Unit test cases):

Test ID 1 (Test the behaviour of ArrayList<Member> when called before or after core software functions):

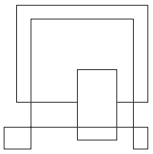
1. Inputs loaded
2. Core actions executed:
    - Create new member
    - Edit member
    - View member details
3. Errors:
    - Member list updates, however for unknown reasons the same lists object points to different lists in at different point in the execution timeline.
4. Solution: The array list must be called from inside the “FileAdapter” class always.
5. Reasoning behind solution: All update, read, write, and sort action are performed inside the file adapter, therefore the list with latest changes will always be inside the file adapter class.
6. Issues resulting from applied solution: Overriding the compareTo() and compare() methods causes the application to delete elements in one list and sort elements one a similar list with a different reference address.

NOTE: The same executed actions, errors, solutions and reasoning behind the solution can be applied for Test ID: 2 and 3 from the white box test phase.

Test logs Black Box Test Phase (Validation testing):

Test ID 1: Test the ability to create a new member.

1. Inputs loaded
2. Save button clicked
3. Errors:
    a. New member window not closing after hitting save button
4. Solution: added the missing functionality in the UI part of the application



5. Reasoning behind solution: since the core function worked as expected and the new member has been created, the only problem could be in the UI part of the application.
6. Issues resulting from applied solutions: N/A

NOTE: The same executed actions, errors, solutions and reasoning behind solution can be applied for Test ID: 2 and 3 from the black box test phase.

Test ID 4: Test the update of data

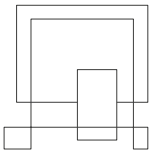
1. Inputs loaded
2. Update time tick passed
3. Errors:
    a. List not updated
    b. List not displayed
4. Solution: We changed the code so that the list is called from the File Adapter both for being updated and for being displayed.
5. Reasoning behind the solution: Even though the update method worked correctly it was called on a list which itself was not called from the File Adapter but from another class.
6. Issues resulting from applied solutions: N/A

NOTE: The same executed actions, errors, solutions and reasoning behind solution can be applied for Test ID: 5 from the black box test phase.

Test Id 6: Test the ability to assign instructor or member to event

1. Inputs loaded
2. Save button clicked
3. Errors:
    a. Instructor added but not displayed since UI not refreshed
4. Solution: clear and reload list inside the UI part of the application.
5. Reasoning behind the solution: Self explanatory

Issues resulting from applied solutions: N/A



7 Results and Discussion

When we started working on the project everything looked clear as in what classes we required and what functionality they should perform. After initial phases of implementation some problems started to appear as in dividing some classes into 2 smaller classes in order to improve functionality, maintenance and workload.

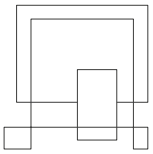
Some features of the system ended up overlapping or were causing conflicts between themselves. This called for a removal or editing of the methods employed. The same happened with interface when we found out some features are missing or should be added to improve the functionality.

Design was changed multiple times to find the perfect spot from our perspective how the interface should function and look like. Most of these problems were found trough testing phase. This phase was crucial for our project to determine whether all necessary functionalities are there and working, if any issues or errors were found the group had to brainstorm for solutions. All of our findings were priority and had to be solved straight away, so there are no complications further on.

8 Conclusions

The aim of the project was to to design and create a system that allows the company's staff to perform client/instructor management and storage in an efficient way to allow the gym to improve its assets. The end result of the project is the ViaFit application that fulfills the client's wishes.

From the design stage a number of problems were detected. The respective problems are as follows: the usage of the system had to be consistent with the desired actions, the correct data flow to and from the binary file, and the system had to fulfill the client's requests.



As a conclusion, the aim of the project was successfully reached. The current solution also has the possibility of improvement for the future should it require (a data base, more functionality involving the instructor/members).

9 Projects Future

Overall the whole project turned out better than first expected. However, the program could be improved if the group had had more time. One of the most important things is how we structured our code, for instance most of the functions were written directly when action was needed in the GUI class, therefore the code is harder to read. The ideal way would be to create all the functions separate methods and when the functionality is needed, it should call the method.
Another improvement could be the way methods are named to improve the speed of reading code and looking for suggestive methods.

Coming closer to the deadline of the project our group faced difficulties with time management, thus some functionality for the program is missing. Like stated earlier, the structure of the project could have been made in a more professional manner, but once again we faced time constraints. More time should have focused on separating the workload. Also, there should have been taken a more realistic approach on time consumption for aspects of the projects including planning, development and testing.

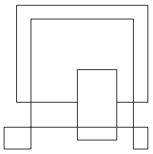
To improve our project, we should work on its structure, the code should be simplified and cleaned up, therefore making sure there are less places where we could have made mistakes that could cause errors. But mostly the program is complete and technically is ready for production. Looking further into future our program could be improved in some ways:

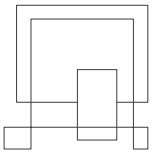
- Storing the information online or on LAN servers
(This means that the information could be accessed from multiple computers at the same time)

- The program could be online
(If we could manage to make our program online for instance on ViaFit website, would make it easier
for clients to see all events and register for them)

- Give users the possibility to make their own registrations for events

In conclusion the program has great potential, there are parts that could be improved about the project, but leaving aside these small mistakes, the core functionality of this application is in place and ready to be extended with more functionality.





10 Sources of information

Source of information

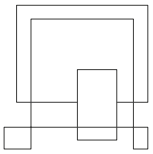
Tony Gaddis, 2015, *Starting Out with Java,* 5th ed. Clyde, North Carolina: Haywood Community College in assoc. With Pearson.

CodeAcademy, 2018. *Learn Javal.* [online]
Available at < https://www.codecademy.com/learn/learn-java>
[Acessed first 25 May 2018]

GitHub Incorporated, 2018. GitHub(1.2.3). [computer program] Published by GitHub Inc.
Available at <https://github.com/>
[Accessed 15 April 2018]

Change Vision incorporated, 2006-2017. Astah professional (7.2.0/1ff236). [computer program] Published by Change Vision Inc.
Available at <http://astah.net/>
[Acessed 5 February 2018]

Eclipse Foundation Incorporated, 2018. Eclipse IDE for Java Developers (4.7.3a). [computer program] Published by Eclipse Foundation Inc.
Available at https://www.eclipse.org/
[Accessed 5 February 2018]



11 Appendices

Activity-Diagram-Appendix-A.

Via-Fitness-Application-Guide-Appendix-B.

UML-Class-Diagram-Appendix-C.

Project-Description-SEP-Group1-Appendix-D.

JavaDocs-Appendix-F