Eksamens rapport Svømmehal Delfinen

To be able to write the code together, we tried something unorthodox and wrote most of the code in Google.docs. While we couldn’t compile the code, we were able to write the code together with everyone being able to keep track of what was happening. We then copied it to Visual Studio Code and corrected a few syntax errors to make it compile. This made everyone get more involved in writing the actual code for the program instead of a single person writing and then pushing it to Github. This has significantly increased the learning process for all members of the group.

We based our usecase/usecase diagram and noun/verblist on the information from the assignment given, and made a rough draft to how we thought our program was going to look like with different classes and which roles they were going to have.

*UseCase:*

*Active VerbList/sætninger:*

*Controlling members info such as subscription and swim results*

*Register misc basic information of members*

*Handling of subscriptions*

*Show a list of members who is in debt*

*Register the Competition/Elite swimmers*

*Register swim results, date, swim competitions, competition placement*

*Show the Delfinen 5 best elite swimmers.*

*Usecase:*

*Title: CRUD Members.*

*Actor: President*

*Scenario: Member register to the club with the firstName, lastName, age, activityStatus and Gets an Id number and a subscription type. Members are also categorized as senior/junior, active/passive and motionist or competitive swimmer.*

*Title: Display/Update subscribtions.*

*Actor: Cashier*

*Scenario: The cashier has to be able to display and update the price of the*

*Subscribtions type.*

*Title: ShowDebtList.*

*Actor: Cashier*

*Senario: Cashier can check if any members are in debt.*

*Title: Create/Display/Update swim results, date, location and Swim discipline for members.*

*Actor: Coach*

*Scenario: The coach can register results in 2 ways; training sessions and competitions.*

*The coach registers date, results, location and swim dicipline for members.*

*The coach should also be able to display the these results and in addition display a top 5 overall result List.*

*Noun List:*

*Members*

*Subscription*

*Swim Results*

*Infomation*

*Member List*

*Swimmer*

*Competition*

*Placement*

*Cashier*

*Debt List*

*Date*

*Location*

*Swim Discipline*

*Coach*

*Result*

*President*

Usecase diagrams see appendix

First version is 1a

Final version is 1b

We then proceeded to make a SSD Diagram showing the interactions within the different classes between actor and our system.

SSD diagram version 1:

Vers. 1 look at appendix 2a and 2b.

we then figured out, that we were missing the president as actor and added additional version.

Vers. 2 Look at appendix 2c

After the SSD diagram, we made a domain model to give us an idea of how the different classes were connected with multiplicities

See vers.1 at appendix 3a

See final version at appendix 3b

To have a more specific example of how our system would work, we made a diagram of how the “president” and the cashier interacted with each other when creating/Updating/deleting a member from the system.

To see the diagram see appendix 4a

In the end we made a class diagram, showing exactly how the code would look on paper with all the previous information put together.

To see version 1 see appendix 5a.

To see the final version see appendix 5b.

To conclude, writing in google docs made all of us able to write together, but when we had to compile the program and run it, we ran into a lot of errors.

Because we are 4 different persons with different thought process, there was some syntax errors mainly because we had different ideas of the layout of the project.

To resolve this issue for next time we write a code together, we would set up a coding template before we start writing to make our coding more coherent.

The template for our next program would look like this:

**// For hver class skal der være:**  
public class Class{

**// hvis nødvendigt almindelige attributter: dette er bare et eksempel der kan sagtens være flere**   
 private String attribute1;  
 private Int attribute2;  
  
// I hver class skal der være en Scanner attribute så vi kan bruge 'scan.nextLine()' i alle-  
// metoder  
 private Scanner scan = new Scanner(System.in);  
  
// Hvis der er ArrayLister skal der tilføjes 5 metoder til dem:  
 private ArrayList<Object> liste;

**// En Constructor: kun hvis der er almindelige attributter**  
public Class(String attribute1, int attribute2){  
this.attribute1 = attribute1;  
this.attribute2 = attribute2;  
  
// ArrayListen skal tilføjes i constructoren og sættes lig ny liste   
this.liste = new ArrayList<Object>();  
}

**// Metoder til almindelige attributter: settere og gettere**  
// setterne skal kun bruges, hvis vi skal være istand til opdatere vores ClassObject  
public void setAttribute1(){  
 attribute1 = scan.nextLine();  
}  
  
public String getAttribute1(){  
 return a1;  
}  
  
public void setAttribute2(){  
 attribute2 = scan.nextLine();  
}  
  
public String getAttribute2(){  
 return attribute2;

**// Metoder til ArrayLister: 5 metoder ialt, til hver liste:**   
// en getter  
// en addToList  
// en displayList  
// en getObjectfromList  
// en DeleteObjectFromList  
// husk!! de skal kun være der, hvis der er ArrayLister

public ArrayList<Object> getList(){  
 return liste;  
}  
  
// Til addToListe: skal det antal argumenter der i constructoren fra en object class tilføjes som // argumenter til addToListe metoden og som parametrer til Object kaldet   
public void addToListe(argumenter){  
 liste.add(new Object(parametrer)  
}  
  
public void displayListe(){  
 for (Object o : liste){  
 o.displayObject();  
 }  
}  
  
// for at finde et specifik object skal vi bruge parametrer i enten ental eller flertal   
// der kan være equal med objectets attributter i enten ental eller flertal   
public Object getObject(argumenter){  
 for (Object o : liste){  
 if (parametrer.equals(o.getAttributter()){  
 return o;  
 }  
 }  
 return null;  
}  
  
// I delete metoden skal vi bruge et Object parameter for at vide hvilket object vi skal slette fra // listen  
public void deleteObjectFromList(Object o){  
liste.remove(o);  
}

**// hvis der er almindelige Attributter i classen skal der laves en displayObject metode:**  
public void displayObject(){  
 System.out.println("tekst" + attribute1);  
 System.out.println("tekst" + attribute2);  
// hvis der er brug for det så skal displayList metoden også sættes ind her:  
 displayListe();   
}

**// hvis der er almindelige attributter i classen skal der være en toString**   
public String toString(){  
 return attribute1 + attribute2;