Projektopgave forår 2013 - Juni

02324 Videregående programmering.

CDIO del 1

Gruppe nr: 13. Afleveringsfrist: Mandag den 25/02 2013 Kl. 05:00

Denne rapport er afleveret via Campusnet (der skrives ikke under)

Denne rapport indeholder 15 sider incl. denne side.


s113730, Østergaard, Jesper 

 s123694, Celik ,Veli

s123849, Walbom, Oliver 

 s122966, Schmidt, Christopher

s123123, Kok, Mathias 

 s114211, Hviid, Nicklas

s123658 , Bengtsson , Martine 

# Videregående programmering - CDIO 1

**Time-regnskab**

| Dato | Deltager | Design | Impl. | Test | Dok. | Andet | Ialt |
|---|---|---|---|---|---|---|---|
| 11-02-2013 | Jesper | 1,5 | | | | | 1,5 |
| - | Oliver | 1,5 | | | | | 1,5 |
| - | Martine | 1,5 | | | | | 1,5 |
| | Veli | 1,5 | | | | | 1,5 |
| | Mathias | 1,5 | | | | | 1,5 |
| | Chrisopher | 1,5 | | | | | 1,5 |
| 18-02-2013 | | | | | | | 0 |
| | Jesper | | 2 | | | | 2 |
| | Oliver | | | 2 | | | 2 |
| | Martine | | 2 | | | | 2 |
| | Veli | | 2 | | | | 2 |
| | Mathias | | 2 | | | | 2 |
| | Chrisopher | | 2 | | | | 2 |
| | Nicklas | | 2 | | | | 2 |
| 19-02-2013 | | | | | | | 0 |
| | Jesper | | | | 2 | | 2 |
| | Oliver | | | | 2 | | 2 |
| | Martine | | | | 2 | | 2 |
| | Veli | | | | 2 | | 2 |
| | Mathias | | | | 2 | | 2 |
| | Chrisopher | | | | 2 | | 2 |
| | Nicklas | | | | 2 | | 2 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | 0 |
| 24-02-2013 | | | | | | | 0 |
| | Jesper | | | | 2 | | 2 |
| | Oliver | | | | 2 | | 2 |
| | Martine | | | | 2 | | 2 |
| | Veli | | | | 2 | | 2 |
| | Chrisopher | | | | 2 | | 2 |
| | Nicklas | | | | 2 | | 2 |
| | Martine | | | | 2 | | 2 |
| | | | | | | | 0 |
| | Sum | 9 | 12 | 2 | 28 | 0 | 51 |

# Indhold

Forord ..... 4
Indledning ..... 4
Problemformulering ..... 4
Usecase diagram ..... 5
Programmets opbygning ..... 6
Sekvensdiagrammer ..... 6
Nedarvning ..... 9
Klassediagram ..... 9
Implementering ..... 11
Test ..... 14
Afsluttende kommentar ..... 15

# Forord

Denne rapport er første del af et CDIO projekt på DTU, og er skrevet i faget 02324 - Videregående programmering. Rapporten forklare kort hvilke tanker gruppen har haft omkring opbygningen af første del af programmet, samt illustrerer det ved hjælp af UML diagrammer.

# Indledning

I CDIO del 1, har vi skulle lave et program, der kan benyttes til administration af operatører til et afvejningssystem. Afvejsningssystemet er dog kun blevet implementeret som en simpel password applikation her i CDIO 1. Senere hen er det meningen programmet skal snakke sammen med en rigtig vægt over netværket.

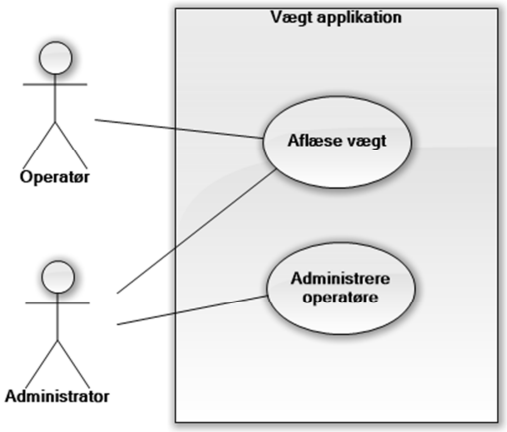
# Problemformulering

Programmet skal have en loign funktion, så det kun er muligt at tilgå vægt applikationen hvis man er oprettet som bruger. Brugerne skal selv kunne ændre deres password, de skal dog overholde samme regler mht. sværhedsgrad som anvendes for passwords på DTU's netværk.

Programmet skal have en administrator menu, hvilket vil sige hvis man er administrator kan man administrere brugere der har adgang til programmet. Når en administrator opretter en bruger skal den oprettede bruger automatisk tildeles et ID samt et autogenereret password som overholder reglerne. Der må kun eksistere brugere med ID fra 11-99.

Programmet skal have en simpel password beskyttet applikation der beregner brutto = Netto + Tara.

## Usecase diagram




For at bruge vægt programmet er det første skridt altid at logge på, hver skal brugernavn og password indtastes, hvorefter programmet giver brugeren yderligere muligheder alt efter om brugeren er en operator eller en administrator,

Hvis brugernavn og password ikke passer sammen, vil programmet informere om dette og give mulighed for at forsøge at logge på igen. Der i øjeblikket ingen krav om, at systemet skal låse hvis brugerinformation bliver tastet forkert tilstrækkeligt gange.

Når en bruger er logget på vil han/hun få stillet muligheder alt efter deres bruger type. En operator vil få mulighed for at logge ud, hvorefter programmet vil vende tilbage til log-in skærmen, vælge at bruge en vægt-application. Her skal et applications nummer indtastes, hvis nummeret matcher en eksisterende applikation, vil brugeren få adgang til applicationens funktioner. Hvis nummeret er forkert vil programmet informere om dette og bede om et korrekt nummer.

Der kan også vælges at ændre password. Her vil brugeren blive bedt om at indtaste sit nuværende password, samt det nye password to gange. Hvis det gamle password er rigtigt, og det nye overholder reglerne for password (DTUs password regler bruges) vil passwordet blive ændret, ellers vil brugeren få at vide, hvad der var forkert og få muligheden igen.

Alle administratorer har adgang til alle operator funktioner. De har dog også mulighed for at modificere operatorer. Disse muligheder indbefatter sletning af eksisterende brugere, oversigt over registrerede brugere, modificering at eksisterende brugere samt tilføjelse af nye brugere.

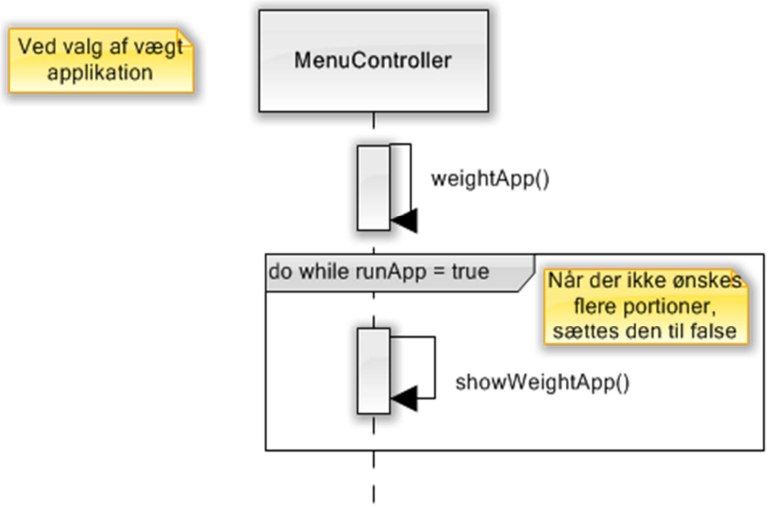
## Programmets opbygning

Programmet er bygget op efter 3 lags modellen, vi har dog indført en controller mellem grænsefladen og funktions laget. Dette har vi valgt at gøre, da det gør det nemmere for os at videreudvikle programmet f.eks. med GUI. Vi har lavet 2 controllere: 1 til at styre selve hovedmenuen(MenuController), dvs. der hvor den normale bruger har mulighed for at ændre password samt tilgå vægt applikationen. Den anden controller(AdminController) styrer de ekstra funktioner en administrator får dvs. oprette bruger, slette bruger og ændre bruger oplysninger. De 2 controllere modtager brugerinput fra grænseflade klassen, hvor de derefter sender informationen videre til funktionsklassen, som f.eks. laver tjekket på, om det nye password opfylder kravene, dvs. funktionaliteten i programmet. Alt data i programmet er gemt i klassen Data, og vil forsvinde når programmet lukker ned. Vi har valgt at gemme test data i en ArrayList, da den ikke skal have en fast længde, som f.eks. Arrays. Dette gør at vi nemt kan tilføje samt slette brugere.

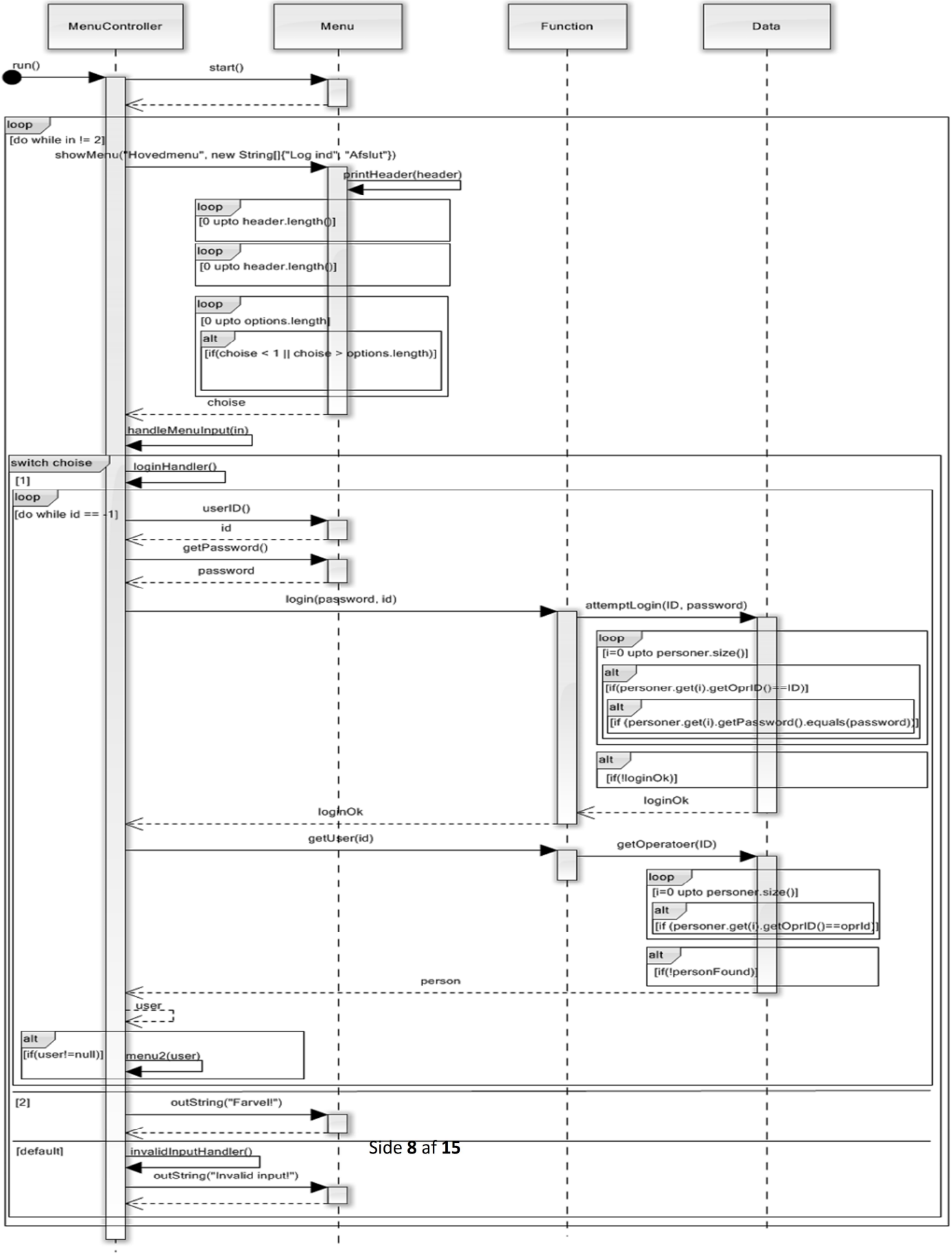
## Sekvensdiagrammer

Vi har valgt at lave to sekvensdiagrammer over vores program. Det første viser hovedmenuen, og i detaljer loginprocessen. Det andet diagram "starter" når man er logget ind og har valgt vægt applikationen. Diagrammet viser hvordan vægt applikationen fungerer.

På diagrammerne har vi valgt ikke at vise metodekald foretaget som dele af if-betingelser, input & output metoder (udskrivning til konsol og metoder fra Scanner-klassen), og catch-blokke.

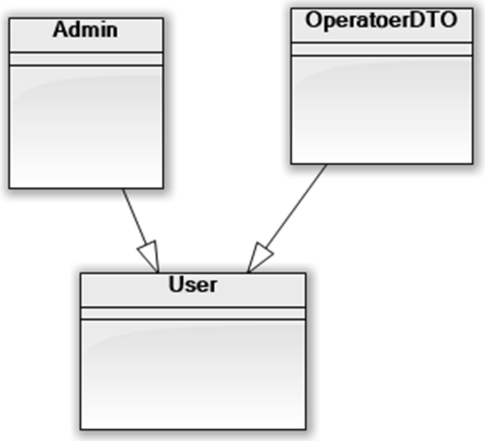


Dette sekvensdiagram viser situationen hvor at vægt applikationen er blevet valgt i menuen. MenuController kalder weightApplication på sig selv. Det første der sker, er at runApp bliver sat til true, dette bliver først sat til false når brugeren vælger ikke at afveje flere portioner. Der bliver så udskrevet en System.out.println der beder brugeren om at indtaste koden til applikationen. Hvis koden er skrevet rigtigt showWeightApp() kaldt, i denne metode er der mulighed for at sætte brutto, tara vægt og beregne vægten. Hvis koden ikke er rigtigt bliver det skrevet ud til brugeren, og brugeren sendes tilbage til menuen. Efter alt dette er gjort bliver brugeren spurgt om der skal afvejes endnu en portion. Hvis ikke bliver brugeren sendt tilbage til hovedmenuen, hvor en anden mulighed kan vælges.


MenuController
Menu
Function
Data
run()
start()
loop
[do while in != 2]
showMenu("Hovedmenu", new String[]{"Log ind", "Afslut"})
printHeader(header)
loop
[0 upto header.length()]
loop
[0 upto header.length()]
loop
[0 upto options.length]
alt
[if(choise < 1 || choise > options.length)]
choise
handleMenuInput(in)
switch choise
loginHandler()
[1]
loop
[do while id == -1]
userID()
id
getPassword()
password
login(password, id)
attemptLogin(ID, password)
loop
[i=0 upto personer.size()]
alt
[if(personer.get(i).getOprID()==ID)]
alt
[if (personer.get(i).getPassword().equals(password))]
alt
[if(!loginOk)]
loginOk
loginOk
getUser(id)
getOperatoer(ID)
loop
[i=0 upto personer.size()]
alt
[if (personer.get(i).getOprID()==oprId)]
alt
[if(!personFound)]
person
user
alt
[if(user!=null)]
menu2(user)
[2]
outString("Farvel!")
[default]
invalidInputHandler()
outString("Invalid input!")

## Nedarvning

I programmet har vi brugt nedarvning. Dette er grundet at vi har 2 typer brugere, nemlig en administrator samt en operator. Ideen med at lave nedarvning er at vi kan lave et tjek på hvilken type objekt brugeren er i stedet for at tjekke på et specifikt ID.



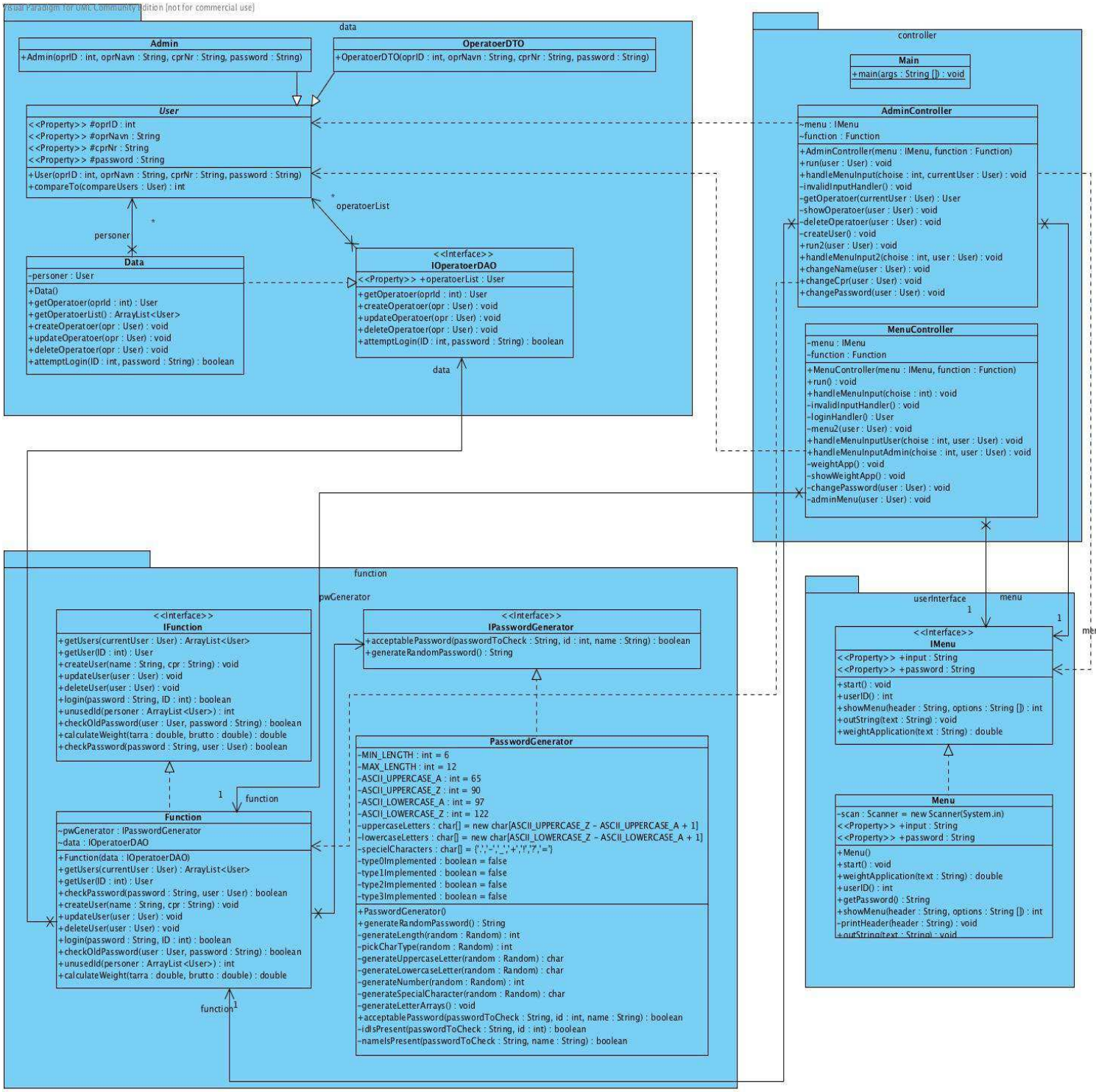

Tjekket bliver brugt i vores menu controller hvor den skal bestemme om der skal vises en admin menu eller en operator menu.

Forskellen på de 2 metoder er følgende:

- Med nedarvning: *if(user instanceof Admin)*
- Ud nedarvning: *if(user.getOprId()==10)*

Her ses det, at ved at bruge nedarvning får du en nemmere mulighed for at have flere administratorer. Hvis vi ikke havde brugt nedarvning og der var flere administratorer, så skulle deres ID være blevet tilføjet således: i*f(user.getOprId()==10 || user.getOprId()==11).* Hvis f.eks. både bruger med ID 10 og 11 er administratorer. Så det kan hurtigt bliver uoverskueligt hvis der skal være mere end en administrator.

## Klassediagram


Visual Paradigm for UML Community Edition (not for commercial use)
data
Admin
+Admin(oprID : int, oprNavn : String, cprNr : String, password : String)
OperatoerDTO
+OperatoerDTO(oprID : int, oprNavn : String, cprNr : String, password : String)
User
<<Property>> #oprID : int
<<Property>> #oprNavn : String
<<Property>> #cprNr : String
<<Property>> #password : String
+User(oprID : int, oprNavn : String, cprNr : String, password : String)
+compareTo(compareUsers : User) : int
operatoerList
personer
Data
-personer : User
+Data()
+getOperatoer(oprId : int) : User
+getOperatoerList() : ArrayList<User>
+createOperatoer(opr : User) : void
+updateOperatoer(opr : User) : void
+deleteOperatoer(opr : User) : void
+attemptLogin(ID : int, password : String) : boolean
<<Interface>>
IOperatoerDAO
<<Property>> +operatoerList : User
+getOperatoer(oprId : int) : User
+createOperatoer(opr : User) : void
+updateOperatoer(opr : User) : void
+deleteOperatoer(opr : User) : void
+attemptLogin(ID : int, password : String) : boolean
data
controller
Main
+main(args : String []) : void
AdminController
~menu : IMenu
~function : Function
+AdminController(menu : IMenu, function : Function)
+run(user : User) : void
+handleMenuInput(choise : int, currentUser : User) : void
-invalidInputHandler() : void
-getOperatoer(currentUser : User) : User
-showOperatoer(user : User) : void
-deleteOperatoer(user : User) : void
-createUser() : void
+run2(user : User) : void
+handleMenuInput2(choise : int, user : User) : void
+changeName(user : User) : void
+changeCpr(user : User) : void
+changePassword(user : User) : void
MenuController
-menu : IMenu
-function : Function
+MenuController(menu : IMenu, function : Function)
+run() : void
+handleMenuInput(choise : int) : void
-invalidInputHandler() : void
-loginHandler() : User
-menu2(user : User) : void
+handleMenuInputUser(choise : int, user : User) : void
+handleMenuInputAdmin(choise : int, user : User) : void
-weightApp() : void
-showWeightApp() : void
-changePassword(user : User) : void
-adminMenu(user : User) : void
function
pwGenerator
<<Interface>>
IFunction
+getUsers(currentUser : User) : ArrayList<User>
+getUser(ID : int) : User
+createUser(name : String, cpr : String) : void
+updateUser(user : User) : void
+deleteUser(user : User) : void
+login(password : String, ID : int) : boolean
+unusedId(personer : ArrayList<User>) : int
+checkOldPassword(user : User, password : String) : boolean
+calculateWeight(tarra : double, brutto : double) : double
+checkPassword(password : String, user : User) : boolean
<<Interface>>
IPasswordGenerator
+acceptablePassword(passwordToCheck : String, id : int, name : String) : boolean
+generateRandomPassword() : String
PasswordGenerator
-MIN_LENGTH : int = 6
-MAX_LENGTH : int = 12
-ASCII_UPPERCASE_A : int = 65
-ASCII_UPPERCASE_Z : int = 90
-ASCII_LOWERCASE_A : int = 97
-ASCII_LOWERCASE_Z : int = 122
-uppercaseLetters : char[] = new char[ASCII_UPPERCASE_Z - ASCII_UPPERCASE_A + 1]
-lowercaseLetters : char[] = new char[ASCII_LOWERCASE_Z - ASCII_LOWERCASE_A + 1]
-specielCharacters : char[] = {'.','-','_','+','!','?','='}
-type0Implemented : boolean = false
-type1Implemented : boolean = false
-type2Implemented : boolean = false
-type3Implemented : boolean = false
+PasswordGenerator()
+generateRandomPassword() : String
-generateLength(random : Random) : int
-pickCharType(random : Random) : int
-generateUppercaseLetter(random : Random) : char
-generateLowercaseLetter(random : Random) : char
-generateNumber(random : Random) : int
-generateSpecialCharacter(random : Random) : char
-generateLetterArrays() : void
+acceptablePassword(passwordToCheck : String, id : int, name : String) : boolean
-idIsPresent(passwordToCheck : String, id : int) : boolean
-nameIsPresent(passwordToCheck : String, name : String) : boolean
1
function
Function
~pwGenerator : IPasswordGenerator
~data : IOperatoerDAO
+Function(data : IOperatoerDAO)
+getUsers(currentUser : User) : ArrayList<User>
+getUser(ID : int) : User
+checkPassword(password : String, user : User) : boolean
+createUser(name : String, cpr : String) : void
+updateUser(user : User) : void
+deleteUser(user : User) : void
+login(password : String, ID : int) : boolean
+checkOldPassword(user : User, password : String) : boolean
+unusedId(personer : ArrayList<User>) : int
+calculateWeight(tarra : double, brutto : double) : double
function
1
userInterface
menu
1
1
menu
<<Interface>>
IMenu
<<Property>> +input : String
<<Property>> +password : String
+start() : void
+userID() : int
+showMenu(header : String, options : String []) : int
+outString(text : String) : void
+weightApplication(text : String) : double
Menu
-scan : Scanner = new Scanner(System.in)
<<Property>> +input : String
<<Property>> +password : String
+Menu()
+start() : void
+weightApplication(text : String) : double
+userID() : int
+getPassword() : String
+showMenu(header : String, options : String []) : int
-printHeader(header : String) : void
+outString(text : String) : void

# Implementering

PasswordGenerator()

PasswordGenerator klassen har to public metoder, generateRandomPassword og acceptablePassword. Begge bliver understøttet af flere private metoder i klassen.

GenerateRandomPassword genererer et tilfældigt password på mellem 6 og 12 tegn, som opfylder password kravene på DTU.

```
int passwordLength = generateLength(random);
String randomPassword = "";

//Sets the 4 first categories
randomPassword = randomPassword + generateLowercaseLetter(random);
randomPassword = randomPassword + generateNumber(random);
randomPassword = randomPassword + generateUppercaseLetter(random);
randomPassword = randomPassword + generateSpecialCharacter(random);
```

af det tilfældige password foregår ved, at der først bliver genereret en længde på det tilfældige password, det gøres ved brug af den private metode generateLength som retunerer en int i intervallet 6-12.

```
private char generateUppercaseLetter(Random random) {
        int arrayPlacement = random.nextInt(uppercaseLetters.length);
        return uppercaseLetters[arrayPlacement];
}
```

Dernæst tilføjes et tilfældigt tegn fra hver de fire lovlige kategorier, ved brug af fire private metoder en fra hver kategori af lovlige tegn Dette gøres ved at alle lovlige tegn er initialiseret i et array for hver sin kategori, den private metode returnerer et tilfældigt tegn i kategorien, som udvælges ved at bruge et tilfældigt tal genereret ud fra arrayets længde.

```
for(int i = 4; i < passwordLength; i++) {
        int type = pickCharType(random);

        switch(type) {
        case 0:
                randomPassword = randomPassword + generateUppercaseLetter(random);
                break;
        case 1:
                randomPassword = randomPassword + generateLowercaseLetter(random);
                break;
        case 2:
                randomPassword = randomPassword + generateNumber(random);
                break;
        case 3:
                randomPassword = randomPassword + generateSpecialCharacter(random);
                break;
        }
}
return randomPassword;
```

Da alle regler for indhold af kategorier nu er overholdt. Resten af passwordet udfyldes ved brug af et "for loop", der tilføjes tilføjes tilfældige tegn fra alle lovlige kategorier indtil passwordet opfylder den tilfældigt genereret længde.

Metoden acceptablePassword checker om et password overholder de givne regler som kræves på DTU.

```
boolean acceptable = false;
char[] passwordChars = new char[passwordToCheck.length()];
int spaceCounter = 0;

if(passwordToCheck.length() >= MIN_LENGTH &&
                passwordToCheck.length() <= MAX_LENGTH) {

        passwordChars = passwordToCheck.toCharArray();
```

Først oprettes et char array med passwordets længde, derefter checkes om passwordet overholder de givne regler omkring længden. Hvis checket består omdannes passwordet til et char array til videre brug.

```
for(int i = 0; i < passwordChars.length; i++) {

        if(passwordChars[i] == ' ') {
                return false;
        }

        for(int j = 0; j <= 9; j++) {
                if(passwordChars[i] == Character.forDigit(j, 10)) { //TODO check virker
                        spaceCounter++;
                        type0Implemented = true;
                        break;
                }
        }
        for(int j = 0; j < uppercaseLetters.length; j++) {
                if(passwordChars[i] == uppercaseLetters[j]) {
                        spaceCounter++;
                        type1Implemented = true;
                        break;
                }
        }
        for(int j = 0; j < lowercaseLetters.length; j++) {
                if(passwordChars[i] == lowercaseLetters[j]) {
                        spaceCounter++;
                        type2Implemented = true;
                        break;
                }
        }
        for(int j = 0; j < specielCharacters.length; j++) {
                if(passwordChars[i] == specielCharacters[j]) {
                        spaceCounter++;
                        type3Implemented = true;
                        break;
                }
        }
}
```

Nu bliver hvert enkelt tegn i passwordet checket, for at se om det opfylder kravene, der returneres "false" hvis der bliver fundet et mellemrum da det ikke er tilladt. Når der findes et af de lovlige tegn, tælles der op på en counter og den pågældende kategoris "typeImplemented" boolean sættes til true.

```
int typeCounter = 0;

if(type0Implemented) {
        typeCounter++;
}
if(type1Implemented) {
        typeCounter++;
}
if(type2Implemented) {
        typeCounter++;
}
if(type3Implemented) {
        typeCounter++;
}
```

Der tælles op hvor mange af de påkrævne typer som er indeholdt, da det ikke er et krav at alle er indeholdt.

```
            if(spaceCounter == passwordChars.length && typeCounter >= 3) {
                    if(!idIsPresent(passwordToCheck, id) &&
                                    !nameIsPresent(passwordToCheck, name)) {
                            acceptable = true;
                    }
            }
            type0Implemented = false;
            type1Implemented = false;
            type2Implemented = false;
            type3Implemented = false;
    }
    return acceptable;
```

Hvis der er blevet optalt, lige så mange lovlige tegn som passwordet er langt og der mindst er inkluderet 3 ud af de 4 påkrævne kategorier, så udføres de sidste check, hvor der bruges to private metoder.

```
private boolean idIsPresent(String passwordToCheck, int id) {
        boolean output;
        String idString = "" + id;

        if(passwordToCheck.indexOf(idString) == -1) {
                output = false;
        }
        else {
                output = true;
        }
        return output;
}

private boolean nameIsPresent(String passwordToCheck, String name) {
        boolean output = false;
        String[] nameStringArray = name.split(" ");

        for(int i = 0; i < nameStringArray.length; i++) {
                if(passwordToCheck.indexOf(nameStringArray[i]) != -1) {
                        return true;
                }
        }
        return output;
}
```

Først checkes om passwordet indeholder brugerID, ved brug af indexOf i den private metode idIsPresent. Til sidst checkes om passwordet indeholder brugerens navn eller efternavn som også gøres ved brug af indexOf.
Hvis passwordet ikke indeholder nogle af disse, returneres at passwordet er acceptabelt.

# Test

For at teste vores program lavede vi først en brugertest, hvor vi tjekkede, hvordan programmet reagerer på forskellige ugyldige/gyldige input.

I vægt applikationen fandt vi bl.a. følgende fejl:

- En negativ tarra og/eller brutto vægt blev accepteret.
- En brutto vægt mindre end tarra vægten blev accepteret.
- Kommatal accepteres ikke som vægt.

Vi fandt forøvrigt ud af, at meget høje tal blev vurderet som ugyldigt input. Det sker som følge af, at vi bruger datatypen int, men er som sådan ikke en fejl, da det ikke bør være muligt at indtaste så mange antal kilo på en medicin vægt. Egentlig burde vi have brugt float eller double og have øvre grænser for antal kilo samt antal decimaler efter kommaet, eftersom den fysiske vægt har sådanne begrænsninger.

I operator administration fandt vi disse fejl:

- En administrator har mulighed for at slette sig selv, (men kan bruge vægten indtil, han logger ud).
- Når en administrator sletter en anden bruger, slettes han også selv, (men igen kan han bruge vægten indtil, han logger ud).

I skift password fandt vi følgende fejl:

- Når man forsøger at ændre sit password, gemmes det nye password ikke (selvom det burde være et gyldigt password).
- Det er muligt at bruge sit ID eller operatørnavn som en del af sin adgangskode.

Ovenstående tests gav os en mistanke om, at der måske var mere galt med funktionen, som tjekker, om et password er gyldigt eller ej, end vi allerede havde set. Derfor lavede vi en unit test af metoden. Her fandt vi følgende fejl:

- Et ugyldigt specialtegn accepteres, så længe ID/operatørnavn ikke er en del af passwordet og passwordet er på minimum 6 tegn samt indeholder mindst tre af de fra DTU's side bestemte tegn kategorier, (A-Z, a-z, 0-9 og (nogle) specialtegn). Konkret testede vi passwordet: aA%08o, hvor % er et ugyldigt specialtegn.

- Det accepteres, at et password indeholder færre end tre af tegn kategorierne, så længe der ikke benyttes ugyldige specialtegn, ID/operatørnavn ikke er en del af passwordet og passwordet er på minimum 6 tegn. Konkret testede vi passwordet: 00-080, som kun indeholder tegn kategorierne 0-9 og specialtegn.
- Æ, Ø, Å og æ, ø, å tælles med som henholdsvis tegn kategorien A-Z og tegn kategorien a-z.

Alle fundne fejl beskrevet ovenfor er nu rettet. Vi har ikke fundet andre fejl i programmet, men vi har et par forbedringsforslag, som vi muligvis vil arbejde videre med i fremtidige delopgaver:

- Tjek af, om det CPR-nr, der indtastes til en ny bruger, er et gyldigt CPR-nr.
- Data om systemets brugere burde gemmes i en database (eller tekstfil).
- Mulighed for, at nye brugere kan få at vide, at de er oprettet som brugere, samt hvad deres ID og password er.

## Afsluttende kommentar

Vi har produceret et program med en login funktion så kun registrerede brugere kan bruge programmet. Inde i programmet er det muligt for brugeren at ændre deres eget password. Dette password bliver tjekket af vores program så det overholder de gældende regler for passwords på DTU's netværk.

Vi har gjort det muligt at programmet kan have flere administratorer. En administrator har mulighed for at administrere alle andre brugere.

Programmet tjekker om der er et ledigt ID mellem 11-99, når der bliver oprettet en ny bruger.

Programmet har en simpel password beskyttet vægt applikation, med koden "appCode". Denne vægt applikation udregner brutto = netto+tara.

Programmet er bygget op på sådan en måde at den nemt kan udvides med flere funktioner samt grænseflade.