CDIO del 1

02324 Advanced programming

Deltagere:

Christin Holt,	s073653	Emil Eriksen	s120561
Jens Nielsen	s123115	Khan Noori	s122997
Kim Rylund	s123667	Magnus Vestergaard	s113727
Thomas Mortensen	s110795	Mathias Larsen	s113734

Lærer:
Mads Nyborg

CDIO del-1							
Time-regnskab	Ver. 2008-09-03						
Dato	Deltager	Design	Impl.	Test	Dok.	Andet	Ialt
11-02-2013	Jens					3	3
-	Kim					3	3
-	Christin					3	3
-	Emil					3	3
-	Khan					3	3
-	Magnus					3	3
-	Thomas					3	3
-	Mathias					3	3
19-02-2013	Jens	1	1				2
21-02-2013	Jens	0	2	0	0	0	2
-	Kim				2		2
-	Khan				2		2
-	Magnus				2		2
-	Emil		2				2
-	Thomas				2		2
24-02-2013	Christin				1		1
-	Emil				1		1

1. Indledning

I denne første del arbejder vi med at oprette operatør konti. For at kunne få adgang til systemet skal der være en operatør konto. Der er en række krav til hvilke oplysninger denne konto skal indeholde. Det ses i afsnit to om krav.

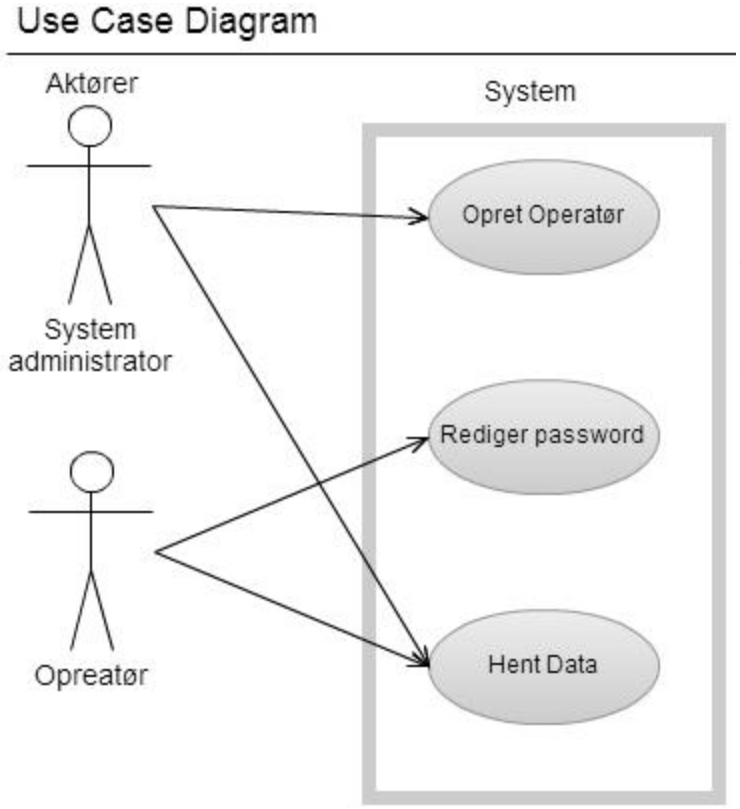
I tredje afsnit gennemgås opbygning og design af programmet, via en use-case og et klassediagram. Implementeringen opbygges via 3-lags princippet, hvor de forskellige klasser har et dertilhørende interface. Det gennemgås i detaljer i afsnit fire. Til sidst beskrives de tests vi gennemfører. Herunder generering af password, ændring af password og oprettelse af operatører.

2. Krav

- Oplysningerne om dem der bruger printeren(operatørerne) skal indeholde et bruger ID, operatørens navn, hans Cpr-nummer, og hans password i ukrypteret version.
- Det password som en bruger får tildelt skal indeholde mindst 6 tegn, og skal indeholde tegn fra minimum 3 ud af 4 af kategorierne, små bogstaver, store bogstaver, tegn, og specialtegn.
- Programmet skal være skrevet efter princippet i 3-lags modellen.
- Det skal være muligt for brugeren at ændre sit password.

3. Design

3.1. Use-case

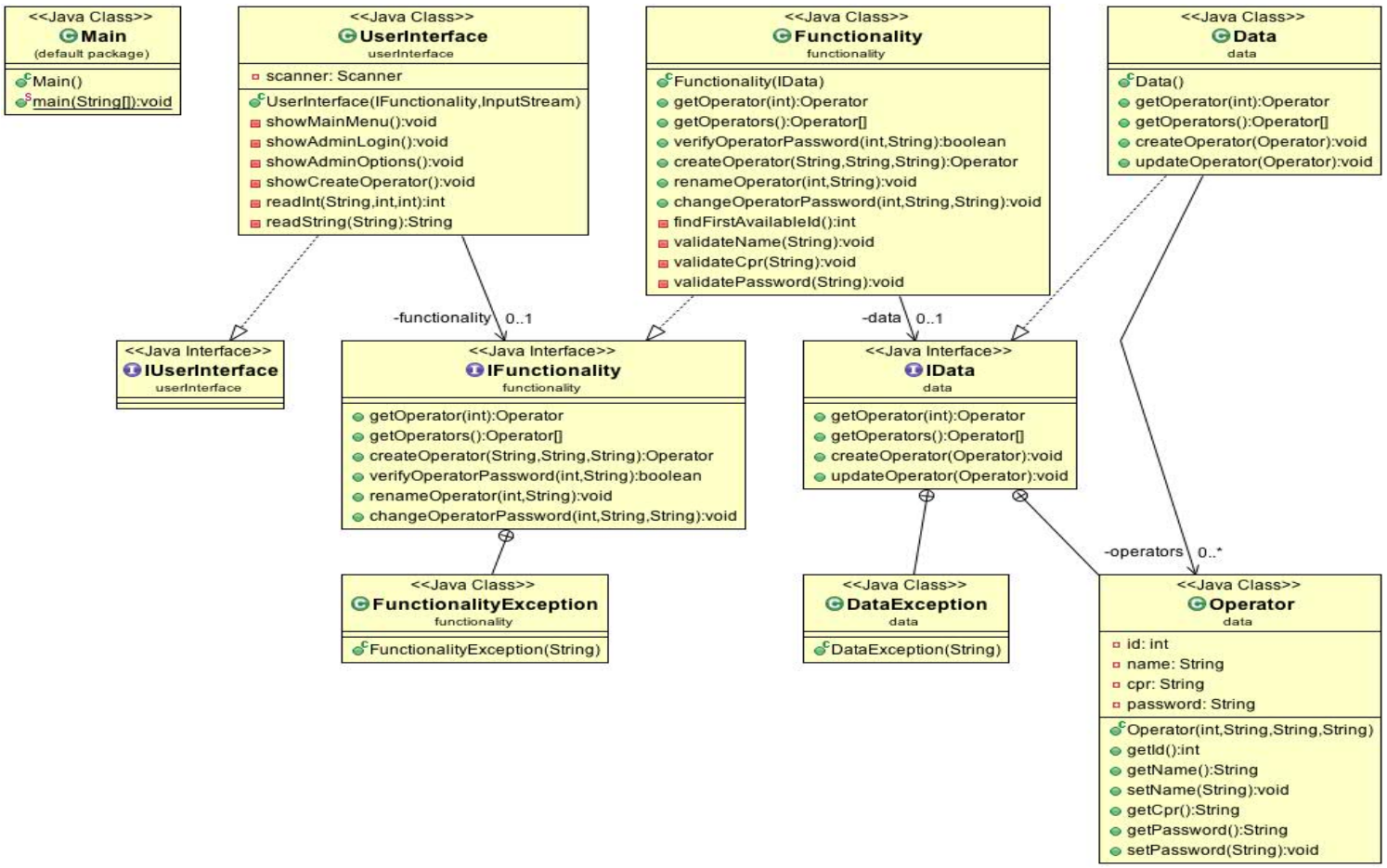


Use Case UC1: Opret operatør
Scope: CDIO1
Primær aktør: System administrator
Stakeholders and interests:
Administratoren vil gerne oprette en operatør
En operatør vil gerne oprettes for at kunne få adgang til vægten.
Preconditions: Administratoren er logget ind.
Succes guarantee(Post conditions): En operatør bliver oprettet og klar til at kunne bruge programmet.
Main succes Scenario/Basic Flow:
1. Adminstratoren logger ind.
2. Adminstratoren vælger opret operatør i

adminstratorens dialog.
3. Adminstratoren indtaster operatørens brugere id og password.
4. Adminstratoren gemmer operatøren
Extentions/ Alternative Flows.
3a. Der eksisterer allerede en operatør med det indtastede bruger id.
    1. Systemet beder administratoren indtaste et nyt bruger id og password
    2. går tilbage til punkt 4
4a. Der er ikke plads til flere operatører.
    1. Systemet fortæller administratoren at der ikke er plads til flere operatører.
    2. går tilbage til punkt 2

Use Case UC2: Hent data
Scope: CDIO1
Primær aktør: Operatør
Stakeholders and interests:
Operatøren vil gerne veje noget.
Preconditions: Operatøren er logget ind.
Succes guarantee(Post conditions): Målingen fra vægten bliver gemt og vist til operatøren.
Main success Scenario/Basic Flow:
1. Operatøren logger ind.
2. Operatøren vælger vej i operatørens dialog.
4. Vægten gemmer den målte værdi og sender den til operatøren

3.2. Klasse-diagram



3.2. Klasse-diagrams beskrivelse.

Som det kan ses, viser vores design klasse-diagram koblingen mellem vores klasser og hvordan de kommunikerer med hinanden. Programmet starter fra Main i Main klassen, denne klasse vil spørge brugeren, om de vil oprettere operatøren, ændre password, teste applikationer eller afslutte programmet.

Vi har så vidt muligt forholdt os til 3-lags modellen, hvor vi deler vores program til tre forskellige lage, nemlig Datalaget, Funktionalitetaslaget og Grænsefladelaget. Vi benyttet os af 3-lags modellen, fordi 3-lags modellen viser mere overskuelighed, svag kobling og bedre mulighed for genbrug og større vedligeholdelsesvenlighed.
Datalaget i dette tilfælde behandler alt, der direkte er knyttet til det at hente informationer om en operatør. Vores Funktionalitetslaget har den nødvendige viden om, hvordan bruger-id , cpr-nummer og password konstrueres, er placeret og dette lag ved, hvilken kombination af bogstaver (kryptering) der skal benyttes til at generere et bruger-id, password og cpr-nummer.

Grænsefladelaget i dette tilfælde sætter brugeren i stand til at oprette en operatør, ændre sin password, teste applikationer og afslutte programmet.

4. Implementering

Programmet er opdelt i tre dele efter tre lags modellen. Lagene er UserInterface, Functionality og Data. Hver af dem har et interface der beskriver deres funktionalitet.

Her ses vores main metode der binder de tre lag sammen.

```
public static void main(String[] args) {
    new UserInterface(new Functionality(new Data()), System.in);
}
```

4.1. UserInterface

UserInterface laget har ansvaret for at håndtere bruger input samt at formidle informationer til brugeren.

4.1.1. IUserInterface

IUserInterface interfacet er i øjeklikket tomt, da der ikke er nogle generelle funktioner i vores UserInterface klasse.

4.1.2. UserInterface

UserInterface klassen benytter en stream til at læse bruger input samt at udskrive informationer til. I vores program bruger vi System.in.

Her ses funktionen der køres når en operatør skal logge ind. Alle de andre menuer er opbygget på samme måde som denne.

```
private void showOperatorLogin() {
    String response;
    Operator operator;
    while (true) {
        System.out.println();
        response = readString("Operatør navn");
        if (response.equals(" ")) {
            break;
        }
        try {
            operator = functionality.getOperator(response);
```

```
            } catch (FunctionalityException e) {
                System.out.println();
                System.out.println(" Fejl: " + e.getMessage());
                continue;
            }
            response = readString("Operatør password");
            if (response.equals(" ")) {
                break;
            }
            try {
                functionality.verifyOperatorpassword(operator.getId(), response);
                showTestApplication();
                break;
            } catch (FunctionalityException e) {
                System.out.println();
                System.out.println(" Fejl: " + e.getMessage());
            }
        }
}
```

Som det kan ses i koden, vil brugeren blive anmodet om at indtaste et operatør navn med tilhørende password, for at fortsætte til næste menu (showTestApplication). Er navnet eller passwordet ikke gyldig vil brugeren blive spurgt igen, indtil der indtastes gyldige data eller et mellemrum for at vende tilbage til forrige menu.

4.2. Functionality

Functionality laget har ansvaret for at udføre alle de nødvendige operationer i programmet. Det tilgås kun af UserInterface laget gennem interfacet IFunctionality.

4.2.1. IFunctionality

IFunctionality interfacet beskriver hvilke operationer vores funktionalitets lag understøtter samt vores FunctionalityException klasse. Funktionalitets laget kan returnere operatør baseret på hans id eller navn, returnere en liste over alle operatører, oprette en operatør, tjekke en operatørs password, omdøbe en operatør og ændre en operatørs password.

Her ses metode listen.

```
Operator getOperator(int id) throws FunctionalityException;
Operator getOperator(String name) throws FunctionalityException;
Operator[] getOperators();
Operator createOperator(String name, String cpr) throws FunctionalityException;
void verifyOperatorpassword(int id, String password) throws FunctionalityException;
void renameOperator(int id, String newName) throws FunctionalityException;
void changeOperatorpassword(int id, String oldpassword, String newpassword) throws
FunctionalityException;
```

Her ses vores FunctionalityException klasse er en exception der bliver kastet hvis noget går galt.

```
public class FunctionalityException extends Exception {
    public FunctionalityException(String message) {
        super(message);
    }
}
```

4.2.2. Functionality

Her ses den funktion der anvendes til at generere en tilfældig password.

```
private static final Random random = new Random();
private static final String passwordChars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789.-_+!?=";
private String generatepassword() {
    String password = "";
    int index;
    while (true) {
        for (int i = 0; i < 6; i++) {
            index = random.nextInt(passwordChars.length());
            password += passwordChars.substring(index, index + 1);
        }
        try {
            validatepassword(password);
        } catch (FunctionalityException e) {
            password = "";
            continue;
        }
        return password;
    }
}
```

Som det kan ses i koden udvælges der seks tilfældige tegn fra stringen passwordChars, og til at godkende om passwordet er tilstrækkelig varieret bruges funktionen validatepassword som kan ses herunder. Hvis ikke passwordet er godt nok, gentages proceduren.

```
private void validatepassword(String password) throws FunctionalityException {
    int categories = 0;
    if (password.matches(".*[A-Z].*")) categories++;
    if (password.matches(".*[a-z].*")) categories++;
    if (password.matches(".*[0-9].*")) categories++;
    if (password.matches(".*[\\.\\-_+!?=].*")) categories++;
    if (categories < 3 || !password.matches("^.{6,}$")) {
        throw new FunctionalityException("password is invalid.");
    }
}
```

Denne kode tester om password matcher de forskellige kriterier der er opstillet. Den skal være på mindst seks tegn og skal indeholde mindst tre af følgende kategorier: Store bogstaver, små bogstaver, tal og special tegn (som kan være “.”, “-”, “_”, “+”, “!”, “?” og “=”). Til at tjekke der er mindst er tre af de

fire kategorier er der opstillet en regex til hver, og antallet af matches skal så være mindst 3. Der er også en regex til at teste om passwordet er langt nok.

4.3. Data

Data laget har ansvaret for at opbevare programmets data. Det tilgås kun af Functionality laget gennem interfacet IData.

4.3.1. IData

IData interfacet beskriver hvilke operationer vores data lag understøtter, vores Operator klasse samt vores DataException klasse. Data laget kan returnere en operatør baseret på hans id, returnere en liste over alle operatører, oprette en operatør og opdatere en eksisterende operatør.

Her ses metode listen.

```
Operator getOperator(int id) throws DataException;
Operator[] getOperators();
void createOperator(Operator operator);
void updateOperator(Operator operator);
```

Vores Operator klasse beskriver en operatør med et id, navn, CPR og en password.

```
public class Operator {
    private int id;            // Indenfor intervallet 11-99.
    private String name;       // Maksimalt 20 karakterer.
    private String cpr;
    private String password; // Ukrypteret.
}
```

Vores DataException klasse er en exception der bliver kastet hvis en operatør ikke kan findes ud fra hans id.

```
public class DataException extends Exception {
    public DataException(String message) {
        super(message);
    }
}
```

4.3.2. Data

Data klassen bruger en ArrayList til at opbevare vores operatører.

```
private ArrayList<Operator> operators = new ArrayList<Operator>();
@Override
public Operator getOperator(int id) throws DataException {
```

```
        for (Operator operator : operators) {
            if (operator.getId() == id) {
                return operator;
            }
        }
        throw new DataException("Cannot find operator with id " + id + ".");
    }
    @Override
    public Operator[] getOperators() {
        return operators.toArray(new Operator[operators.size()]);
    }
    @Override
    public void createOperator(Operator operator) {
        operators.add(operator);
    }
```

Data laget har ikke så meget at lave på nuværende tidspunkt.

5. Test

Programmet er udelukkende testet ved hjælp af black-box testing. Vores exceptions skulle gerne fange de fleste fejl der kan opstå, og ellers bruges black-box testing til at finde eventuelle fejl i logikken. Vi kunne også have oprettet en speciel testklasse, som kunne teste alle vores metoder, for at se om output er det samme som det forventede.

5.1 password generator

Herunder ses en test af den automatisk generede password, som nye operatører får tildelt. Koden skal opfylde kravene, som er er svarende til passwordkriterierne på:
www.campusnet.dtu.dk

```
Operatør navn: Test
Operatør CPR: 123456-7891

Operatøren blev oprettet med id 11 og adgangskoden cq!qV9.
```

Koden er altså genereret korrekt efter kriterierne og id'et som hver operatør får ligger som vist mellem 11 og 99.

5.2 ændring af operatørs eget password

En anden funktionalitet som er testet og en operatørs mulighed for at ændre hans egen password.

```
1: Operatør administration.
2: Ændring af adgangskode.
3: Test applikation.
4: Afslut.

2

Operatør navn: Test
Operatør adgangskode: 3aQ0W7
Ny adgangskode: rrer333E
Gentag ny adgangskode: rrer333E
```

En operatør har altså mulighed for at ændre sin egen password, til et selvvalgt et, så længe det overholder de givne regler. Hvis operatøren ikke kan sin gamle password, kan han selvfølgelig ikke få lov til at oprette et nyt.

5.3 opret operatør

En administrator skal være i stand til at oprette en ny operatør til vægten

```
1: Operatør administration.
2: Ændring af adgangskode.
3: Test applikation.
4: Afslut.

1

Administrator adgangskode: >sdfsdfsdf

Fejl: Password is incorrect.

Administrator adgangskode: >02324it!<

1: Opret operatør.
2: Tilbage til forrige menu.

1

Operatør navn: test
Operatør CPR: 123456-7891

Operatøren blev oprettet med id 11 og adgangskoden 5cXQ5v.
```

Som vist på skærmbilledet ovenfor, skal man bruge et administrator login for at kunne oprette en ny operatør. Man skal derefter give et navn og cpr-nummer, hvorefter systemet opretter et id og password. Dette virker efter hensigten og hvis man indtaster forkerte eller ugyldige værdier vil de bliver fanget af vores exceptions.

6. Konklusion

Operator klassen findes, og den indeholder alle de data som der stod i kravene at den skulle, den er en inner class i data-lagets interface.
Brugerne får tildelt et automatisk genereret kodeord når de bliver oprettet, og kodeordet bliver tjekket igennem, for at se om det indeholder de rigtige typer tegn.
Vores program indeholder et data lag, et userinterface lag, og et funkionalitetslag, som alle kun kender til hinanden gennem interfaces. Dvs. programmet er kodet efter princippet i 3-lags modellen.
Det er muligt for en bruger at ændre sit kodeord.