Delprojekt 1 IT

TEMPERATURMÅLER MED MATH.RANDOM

Zakir H. Shahoo(s.194054), Andreas Bach Berg Nielsen(s.205869) og Troels Engsted Kiib(s.205492) | Informationsteknologi |28-10-2020

Abstract

This project aims to examine how Information Technology can ease the workload experienced by the health worker in a health center. The Unified Modeling language (UML) is being used to visualize the design of a system, meanwhile the programing language "Java" is to generate the developed communication system in the mentioned heath center. The system measuring a patient's body temperature which is one of the four vital signs. Furthermore, the system informs on-call doctor in case of urgent.

Indhold

Formål

Vores Formål er at gøre Jakob glad og stolt samt at udvikle os som programmører og mennesker. Formålet med systemet er at monitoren løbende overvåger sensoren, og i tilfælde af at temperaturen kommer uden for normalområdet skal der sendes en "notifikation" til grænseflademodulet. Hvis temperaturen afviger mere end hvad der er angivet med *setUrgent ()* skal der sendes en meddelelse til lægen via grænseflademodulet til det trådløse tilkaldesystem.

Unified Procces

USE CASE

Kravspecifikationer:

Overvågningsenhed (monitor).

Skal indeholde en metode kaldet *run ()*, der kører en uendelig løkke der henter en værdi fire gange i minuttet, og undersøger om den skal reagere. Man skal kunne sætte en nedre og øvre grænse for normalniveauet på den aktuelle patient, den øvre og nedre grænse skal kunne sættes med to metoder *setmin() setmax()*.

Yderligere skal overvågningsenheden kunne returnere den målte værdi efter konvertering af temperaturen til celsius.

Man skal kunne sætte tolerancen med metoden *seturgent()* og aflæse den med *geturgent()*. Den målte værdi skal gemmes som en værdi.

Sensor.

Sensoren kommer til at bestå af en *math.random()* der skal genere den målte værdi, som bliver et tal mellem 150-200, når vi laver en måling.

Grænseflade modul. (grafisk modul)

Skal være vores grafiske brugerflade(konsol) mellem programmet, altså monitoren, og personalet. Det er blandt andet gennem grænsefladen bruger indstiller *setmin()* og *setmax()*.

Grænsefladen skal opdateres med den sidst foretagne måling 2 gange i minuttet, ved at kalde den relevante metode på monitoren. Det skal også have en aktiv proces, der løbende viser den aktuelle øvre og nedre grænse, samt den sidst målte værdi i celsius i konsollen.

Alarm System

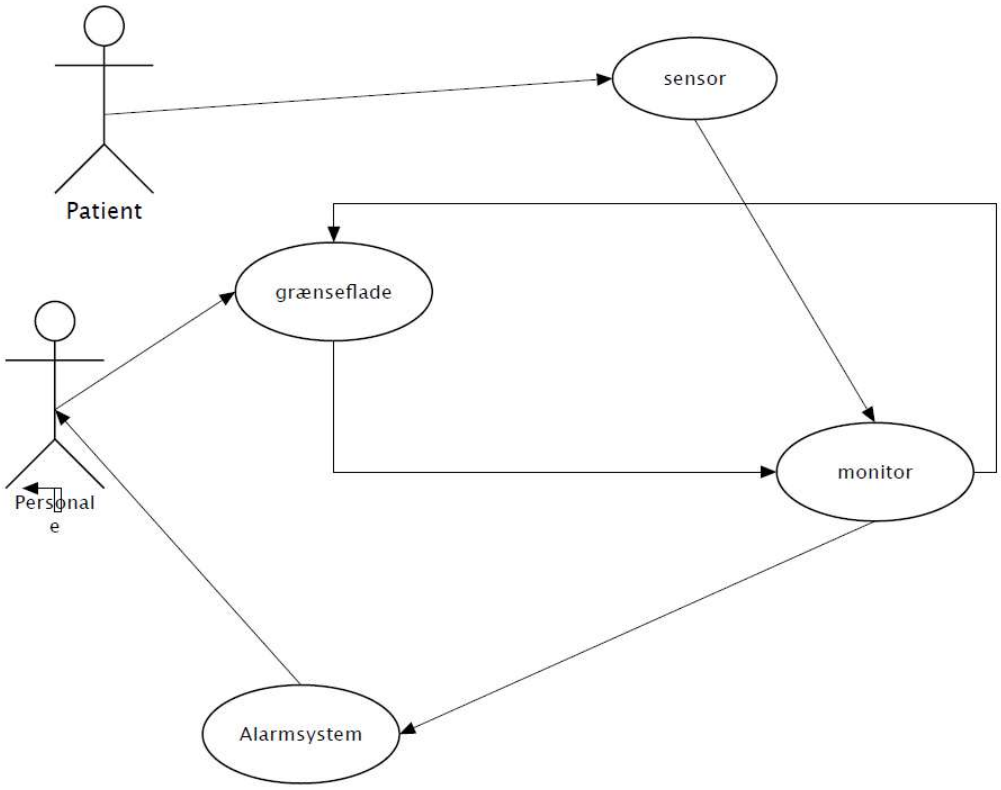
Alarmsystemet er vores trådløse tilkaldesystem, som kan tilkalde en vagthavende læge Dette modul skal implementere en *send ()* metode der har den meddelelse der ønskes sendt som parameter.

Ønsker

Der ønskes et program der løbende overvåget en patient opsat til en sensorisk temperaturmåler. Programmet skal overvåge patientens indre temperatur, og skal opfange hvis temperaturen afviger standardtemperaturen intervallet mellem 36 og 40 grader celsius.

ANALYSE

Kollaborations diagram og aktivitets diagram.

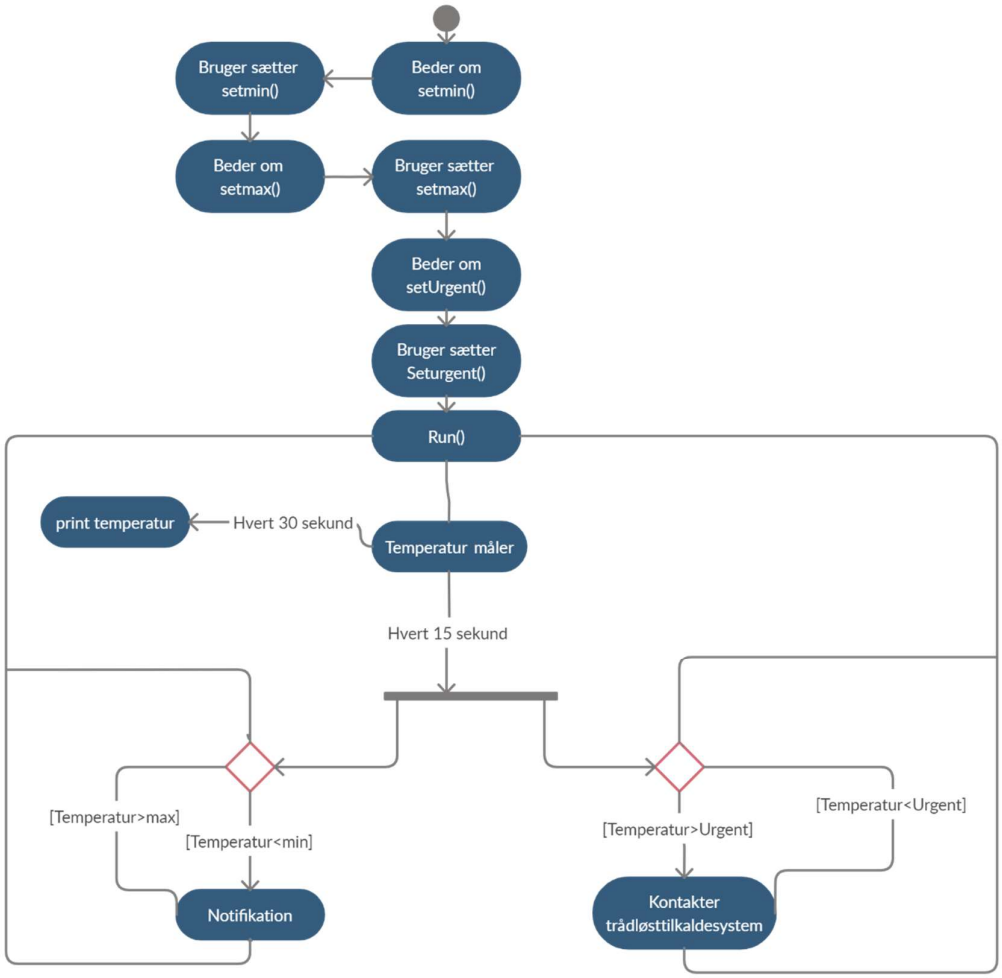


Figur 1: Kollaborations-diagram[1]

Med dette "kollaborations diagram" illustreres samarbejder mellem komponenter.

Først kan vi se personalet interagerer med grænsefladen. Herefter kan vi at monitoren kommunikerer med grænsefladen, vi kan desuden se at monitoren for input fra sensoren, som får input fra patient. Desuden viser illustrationen at det er monitoren der sender til alarmsystemet, som derefter kommunikerer til personalet.

[1] S.71 UML Explained



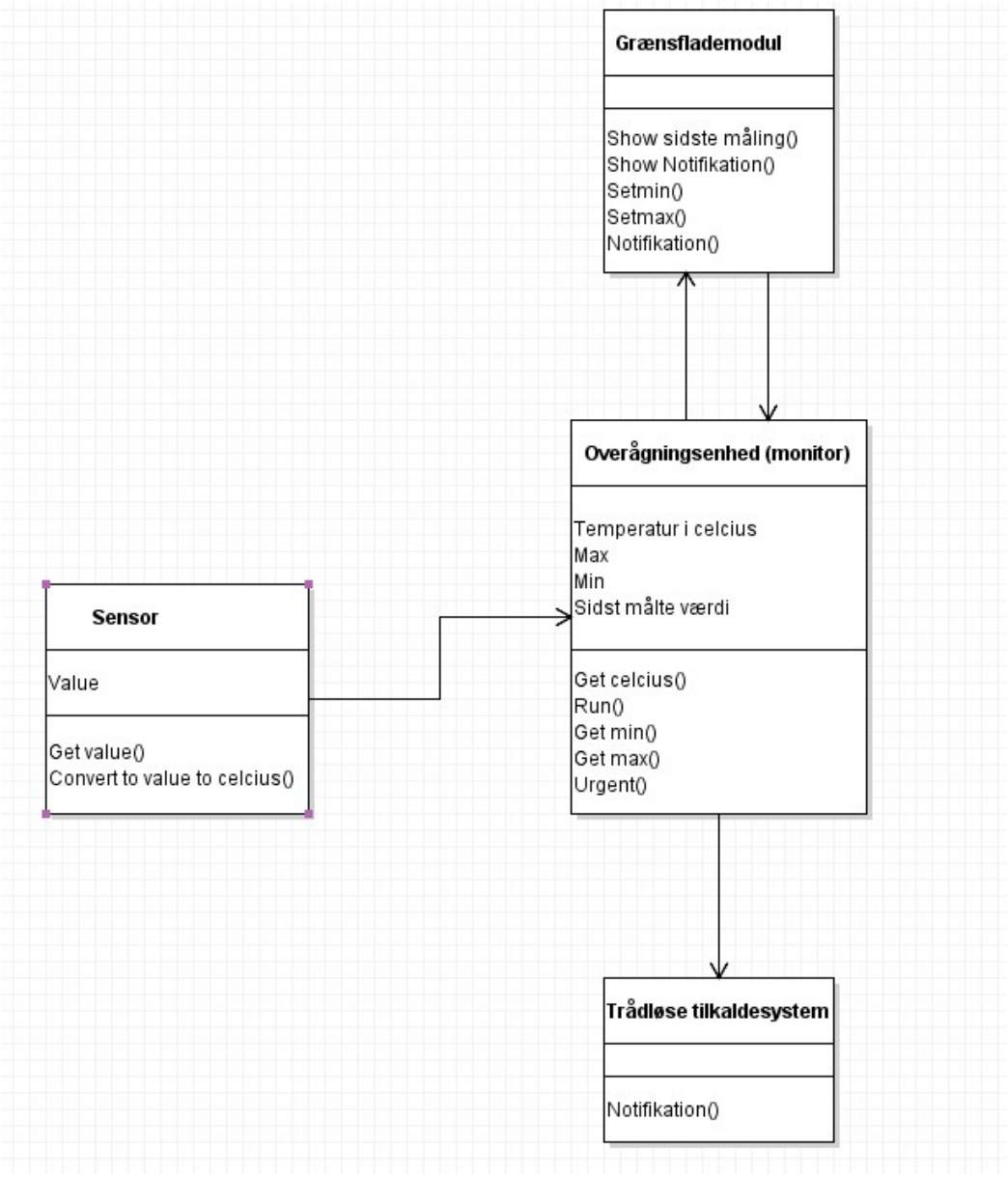
Figur 2: Aktivitets diagram[2]

Aktivitetsdiagrammet har en "Startstand", men ingen "Slutstand" da programmet skal køre uendeligt indtil manuelt stoppet. Programmet starter med at bede brugeren om at sætte max, min og urgent værdierne og fører derefter videre til run. Run kører uendeligt og fører ned til en temperatur måler som hvert 30 sekund printer dets værdi, samt 2 if sætninger der vurderer hvordan der skal reageres, hvor der til slut bliver returneret tilbage til run.

DESIGN

Design diagram.

[2] S.91 UML Explained



Figur 3: Design/Klasse-diagram[3]

Dette design diagram fungerer således, at man kan se de forskellige klasser, deres attributter og deres metoder.

Overvågnings enheden er vores centrale klasse. Denne indeholder de brugbare værdier i attributter, samt kører den centrale *run()* metode. Modulet indeholde en vigtige metode kaldet *urgent()*, der overvåger om sensorens temperatur måling overstiger den normale standard temperatur. Derved kalder på hjælp fra det trådløse tilkaldesystem, og kontakter grænseflademodullet.

[3] S.34 UML Explained

Bemærker man grænseflademodullet, kan man observere at klassen er en attributløs klasse, det indeholder metoder der gemmer værdier i monitorens attributter. Modulet indeholder samtidig metoder der kan vise temperaturen og en eventuel notifikation, hvis *urgent ()* metoden bliver kaldt på overvågningsenheden.

Sensor er vores komponent der sidder på patienten, og opfanger en værdi der gemmes i dets attribut Value. Sensoren indeholder en metode der konverterer den arbitrære værdi til en læsbar enhed kaldet temperatur celcius.

Det trådløse tilkaldesystem indeholder kun metoden *notifikation ()*, og dette moduls formål er, at kalde efter en hjælp via det trådløse netværk.

IMPLEMENTERING

Monitor Attributer

```
public class Monitor {
    static double min, max;
    static double temperatur;
    static double urgent_alarm;
    static double urgent_min, urgent_max;
```

I dette styk kode opretter vi nogle atributter i klassen monitor. De er alle static da vi gerne vil kunne bruge dem som globale variabler og alle doubles, de vi gerne vil give brugeren mulighed for at bruge decimaler, såsom (35,60) og diverse tal.

Monitor Objekter

```
    static Sensor sensorobj_1 = new Sensor();
    static Brugergraenseflade Brugergraeseflade = new Brugergraenseflade();
    static Traadloesnetvaerksystem alarm_server = new Traadloesnetvaerksystem();
```

Her opretter vi objekter således at vi kan bruge de andre klassers metoder. Igen er de *static* således at vi kan bruge dem globalt.

Monitor Metoder

```
    public static void main(String[] args) throws InterruptedException {
        //Kald af get metoderne, til indtasting af værdier
        Getmax();
        Getmin();
        Geturgent();

        Run();
    }
```

Vores *main* metode indeholder kun de fire andre metoder, samt en ”*exception*”

```
    static void Run() throws InterruptedException {
        //kald af intial sensor værdi
        int counter = 0;
        while (true) {
            temperatur = sensorobj_1.konverttemperatur();
            Thread.sleep( millis: 15_000);

            //Print af temp hver 30. sec
            if ((counter % 2) == 0) {
                System.out.println(String.format("%.2f", temperatur) + " celcius.");
            }
            counter++;

            // kontrol af urgent / alarm
            if (temperatur > urgent_max || temperatur < urgent_min) {
                //kalder metoden urgent og slår alarm
                alarm_server.Urgent();
            }

            //kontrol om temperaturer overskrider min eller max
            if (temperatur > max || temperatur < min) {
                // print pop up til sygeplejersken
                Brugergraeseflade.notifikation(temperatur);
            }

        }
    }
```

Vores *run* metode starter med at definere en *integer* kaldet *"counter"*, dernæst kommer en uendelig løkke. *(While loop)*

I løkken sætter vi en temperatur fra vores "konverttemperatur" metode på den globale variable temperatur. Dernæst holder vi en 15 sekunder pause, med *"Thread.sleep"* metoden. Da vi gerne vil printe en værdi hvert 30 sekund, har vi en *if* sætning der tjekker om værdien af *counter* er et lige tal, hvis dette er rigtigt printer vi temperaturen. Efter dette tælles *"counter"* op med 1.

Efter dette har vi to *if* sætninger der vurderer om temperaturen er gået over eller under, det vi kalder *"urgent_max, urgent_min,* max og min" som vi definerer i andre metoder og reagerer henholdsvis herefter med metoder defineret andre steder.

```
//methoder til intialisering max og min værdier
static void Getmax() {
    max = Brugergraeseflade.setmax();
    System.out.println("Maksimal temperatur i celcius :" + max);
}

static void Getmin() {
    min = Brugergraeseflade.setmin();
    System.out.println("Minimal temperatur i celcius :" + min);
```

Her defineres *"Getmin"* og *"Getmax "*metoderne, der sætter værdierne fra *"setmin"* og *"setmax"* metoderne i vores globale variabler. Samt printer en kontrol sætning for hvad værdierne er.

```
    static void Geturgent() {
        urgent_alarm = Brugergraeseflade.SetUrgent();
        urgent_min = min / 100 * (100 - urgent_alarm);
        urgent_max = max / 100 * (urgent_alarm + 100);
        System.out.println("tollerancen er " + urgent_alarm + "%");
        System.out.println("Minimums afvigelse " + String.format("%.2f", urgent_min) + "    " +
                "maximum afviglese " + String.format("%.2f", urgent_max));
    }
}
```

Her defineres *"Geturgent"* metoderne, der sætter værdien fra *"SetUrgent"* metoderne i den globale variable *"urgent_alarm"*. Vi bruger derefter værdien i *"urgent_alarm"* til at udregne *"urgent_min"* og *"urgent_max"* ved at trække eller lægge den procentdel *"urgent_alarm"* er til eller fra min og max værdierne.

Til slut printer vi to kontrol sætninger for hvad værdierne er, hvor vi afrunder ned til to decimaler.

Brugergrænseflade metoder

```
//set min/max methoder, med kald af java swing og konvertering til double
public double setmax() {
    String str = javax.swing.JOptionPane.showInputDialog("indtast en maximalstemperatur");
    return Double.parseDouble(str);
}

public double setmin() {
    String str = javax.swing.JOptionPane.showInputDialog("indtast en minimumstemperatur");
    return Double.parseDouble(str);
}
```

Her opretter vi *"setmax og setmi*n" ved at vi bruger *"javax.swing"* metoden til at prompte en dialogboks, hvori man indtaster en værdi og returnere dette som en double. Hvis man skriver et bogstav her i stedet for et tal, får man en fejl og løsningen på dette problem bliver beskrevet under afprøvning.

```
// bruger interaktion
static void notifikation(double temperatur) {
    javax.swing.JOptionPane.showConfirmDialog( parentComponent: null,  message: "ALARM patienten er i fare," +
            "temperaturen er " + String.format("%.2f", temperatur) + " celcius");
}
```

Her opretter vi *"notifikation"* metoden, som tager en double som input. Ved at bruge *"javax.swing"* metoden til at promote en svarboks, hvori den globale variabel "temperatur" afrundet til to decimaler er delagtigt.

```java
// tilsvarende set methode for urgent
public double SetUrgent() {
    String str = javax.swing.JOptionPane.showInputDialog("indtast en alarm tolerance");
    return Double.parseDouble(str);
}
```

Her opretter vi "*SetUrgent*" metoden ved, at bruge "*javax.swing*" metoden til at prompte en dialogboks, hvori man skriver tolerancen man ønsker programmet skal indeholde, dette returneres i en double. Hvis man skriver et bogstav her i stedet for et tal, får man en fejl og løsningen på dette problem bliver beskrevet under afprøvning.

Sensor klassen

```java
public class Sensor {
    double vaerdi;

    // psuedo sensor som bruger random og omregner til celsius
    public double konverttemperatur() {
        vaerdi = Math.random() * 90 + 130;
        return vaerdi * 4 / 50 + 24;
    }
}
```

I sensor klassen opretter vi en attribut, kaldet "vaerdi", dette gør vi fordi sensorens målte værdi egentlig er pseudo kode. Havde man brugt en reel sensor, ville man nemlig få et vilkårligt tal og dette prøver vi at simulere ved at, vi genere en tilfældig værdi og indfører den på "vaerdi" attributten.

I vores sensor findes metoden "konverttemperatur", som genererer en værdi ved hjælp af "*Math.random*" metoden i intervallet mellem (0 til 90 +130) også skrevet 130 til 220. Dernæst konvertere vi dette til en temperatur grader celcius, mellem 34,4 og 41,6 grader celcius. Vi vælger dette interval da vi gerne vil have at temperatur, der kan overskride normaltemperaturen for mennesker som ligger mellem 36-40 grader. Men stadig ikke gør intervallet for bredt således, at chancen for at tallet bliver generet over standard temperaturen for høj.

Til slut returnerer vi den konverterede værdi.

Trådløst tilkaldesystem klassen

```java
public class Traadloesnetvaerksystem {
    static void Urgent() {
        // Slår alarm til server / alarm notifikation "psuede alarm"
        System.out.print("alarm besked bliver sendt til server    :     ");
        System.out.println("kontakter server...");
    }
}
```

Her oprettes "*Urgent*" metoden, der igen er et styk pseudo kode, der skal simulere at der bliver kontaktet en ekstern server, hvor der skal kontaktes en læge.

AFPRØVNING

Black Box: Vi har lavet afprøvning ved at indsætte en række forskellige tal, bogstaver og symboler. Vi har her observeret at programmet crasher og brænder hvis der ikke bliver inputtet et tal som *min, max* og *urgent* værdi.
Dog har vi ikke implementeret en løsning for dette, da vi mener at omfanget af opgaven skal begrænses.

White Box: Vi har løbene lavet white box teststing ved at implementere print statements i vores kode så vi kunne identificere lokationen af problemer, og om programmet blev kørt som det skulle.
Desuden har vi brugt en række af intellejs indbyggede hjælpeværktøjer til fejlfinding af funktionalitets problemer samt til at finde ubrugte variabler og andre småfejl.
Sidst men ikke mindst har vi løbende talt designet og funktionaliteten igennem på gruppen.

IDRIFTTAGNING

Programmet er ikke blevet taget i drift. Dog hvis det vi skulle lave en betaversion, ville vi idiot sikre input til programmet og lave en form for dataindsamling, og interviewe brugere, altså læger og sygeplejersker.

Resultat

Her er en link til vores programfil, komprimeret til en .rar

https://drive.google.com/file/d/1YgN_4q7ZUOBtsQoyUIy6SnoTiNmW_TIC/view?usp=sharing

Konklusion

Evaluering af metoder

Evauering af metoder samt forslag til andre metoder/værktøjer

Vi har brugt UML metoden til at lave diagrammer over programmets funktionalitet og struktur. Vi kan efterfølgende konkludere at den tid vi brugte på at diskutere struktur og funktionalitet i de forskellige UML-stadier/ (forståelses niveauer)? Har gjort det nemmere at programmere, og forstå hinandens kode og intention med elementer af programmet.

Vi har skrevet programmet i java som er et objektorienteret sprog, og brugt værktøjet IntelleJ til at lave og køre vores program. Det har været gavnligt at bruge Java da vi har kunne bruge allerede eksisterende metoder i vores program, såsom math.random og diverse java.swing paneler.

Evaluering af resultat

Vi kan konkludere, at resultatet af opgaven er det ønskede resultat, da vi under implementering havde en informativ diskussion med vores vejleder, hvori vi blev forklaret at vi lige manglede at implementere en tidsfunktion i vores kode, men ellers var resten egentlig som han ønskede. Derved har vi implementere en tidsfunktion i vores kode, og vil nu mene at programmet, kan det, det som det bliver anvist fra opgaven, at det skal kunne.

Litteraturliste

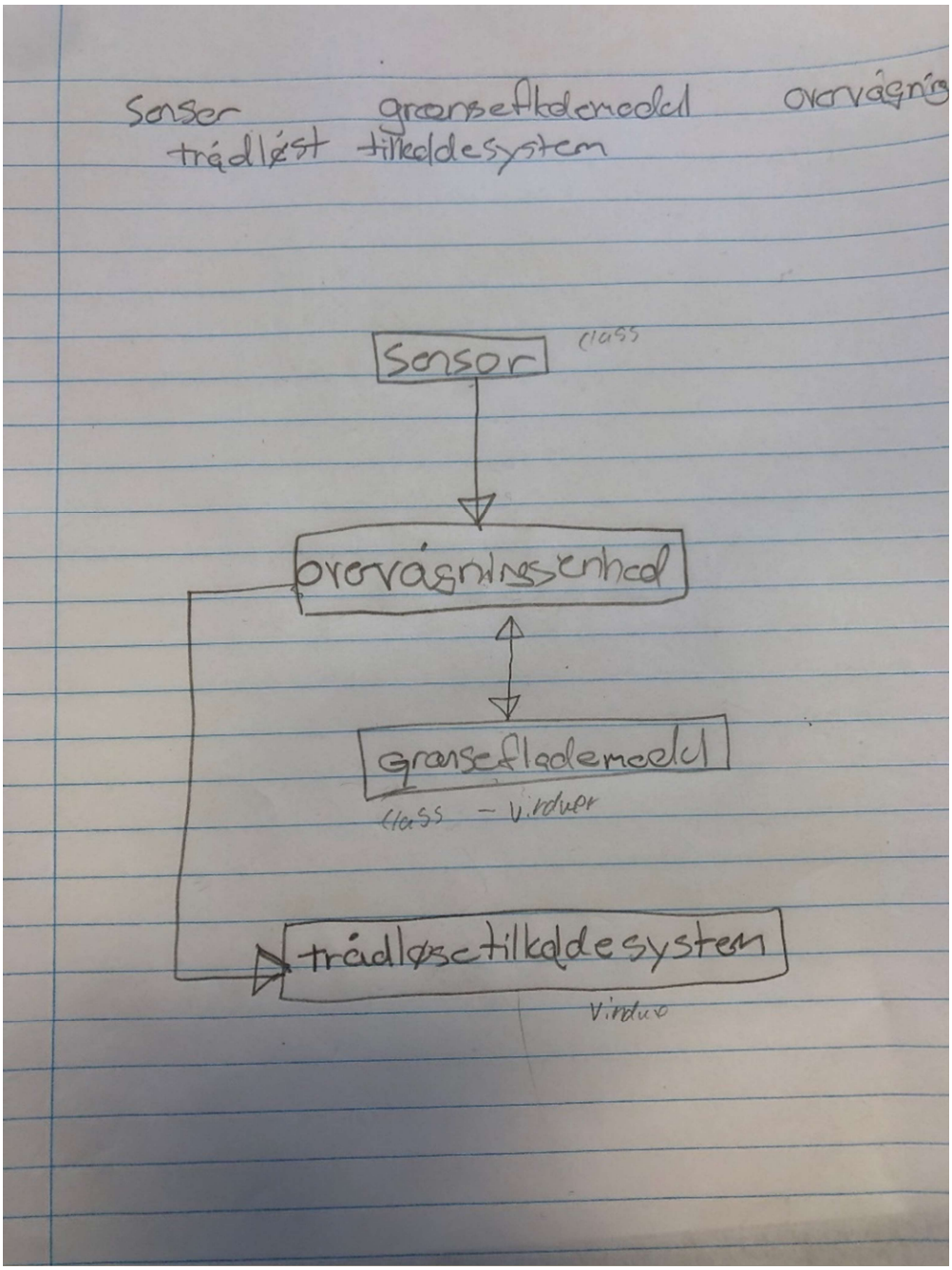
En Bog: Titel, Forfattere, Forlag, Udgivelsesår, ISBN.

Objektorienteret programmering i Java, 6. udgave - dækker Java 11, Globe, 2019, 978-87-425-1034-6.

UML Explained, Kendall Scott, Addison-Wesley, 0-201-72182-1.

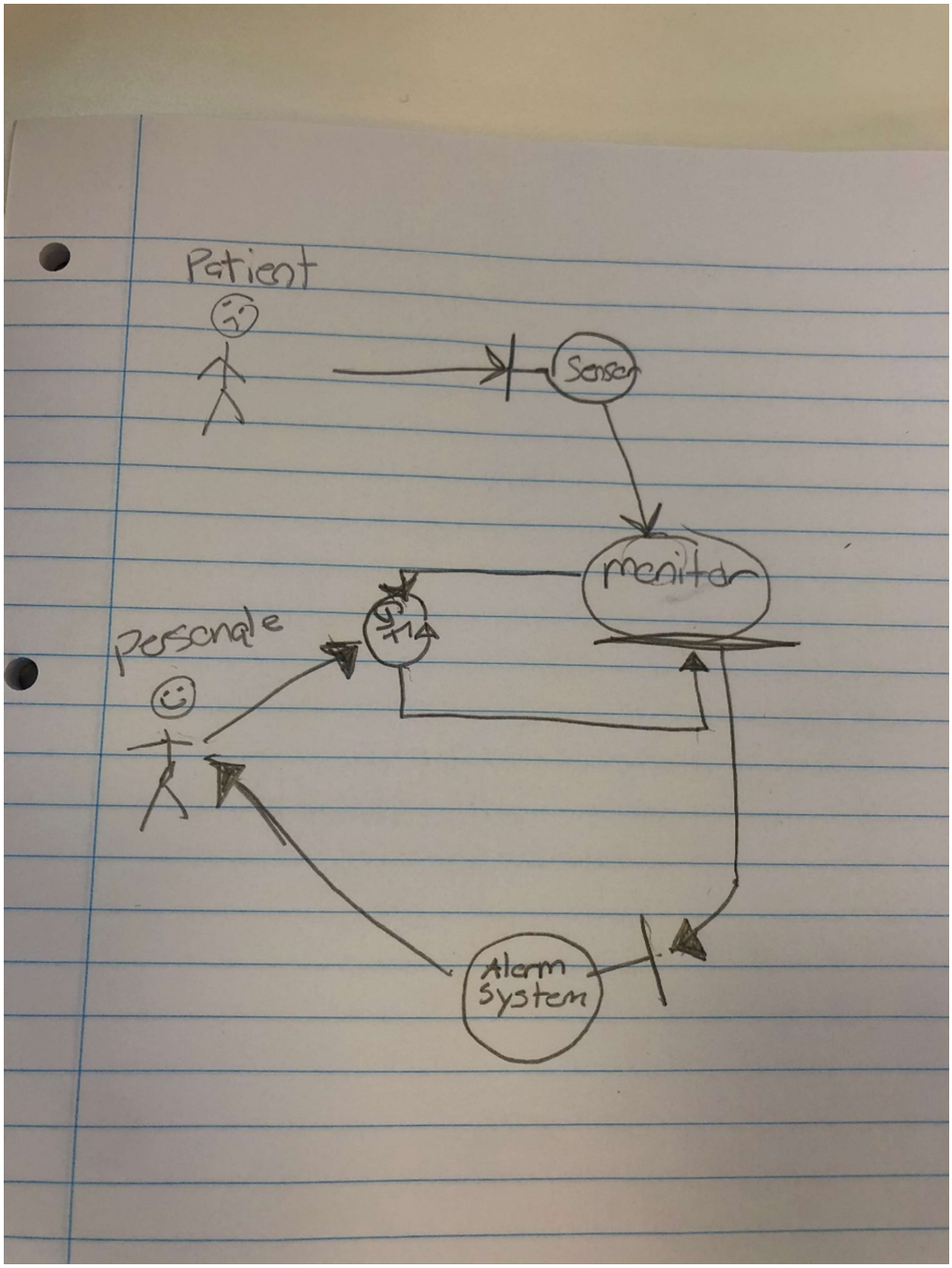
Bilag

Bilag: 1



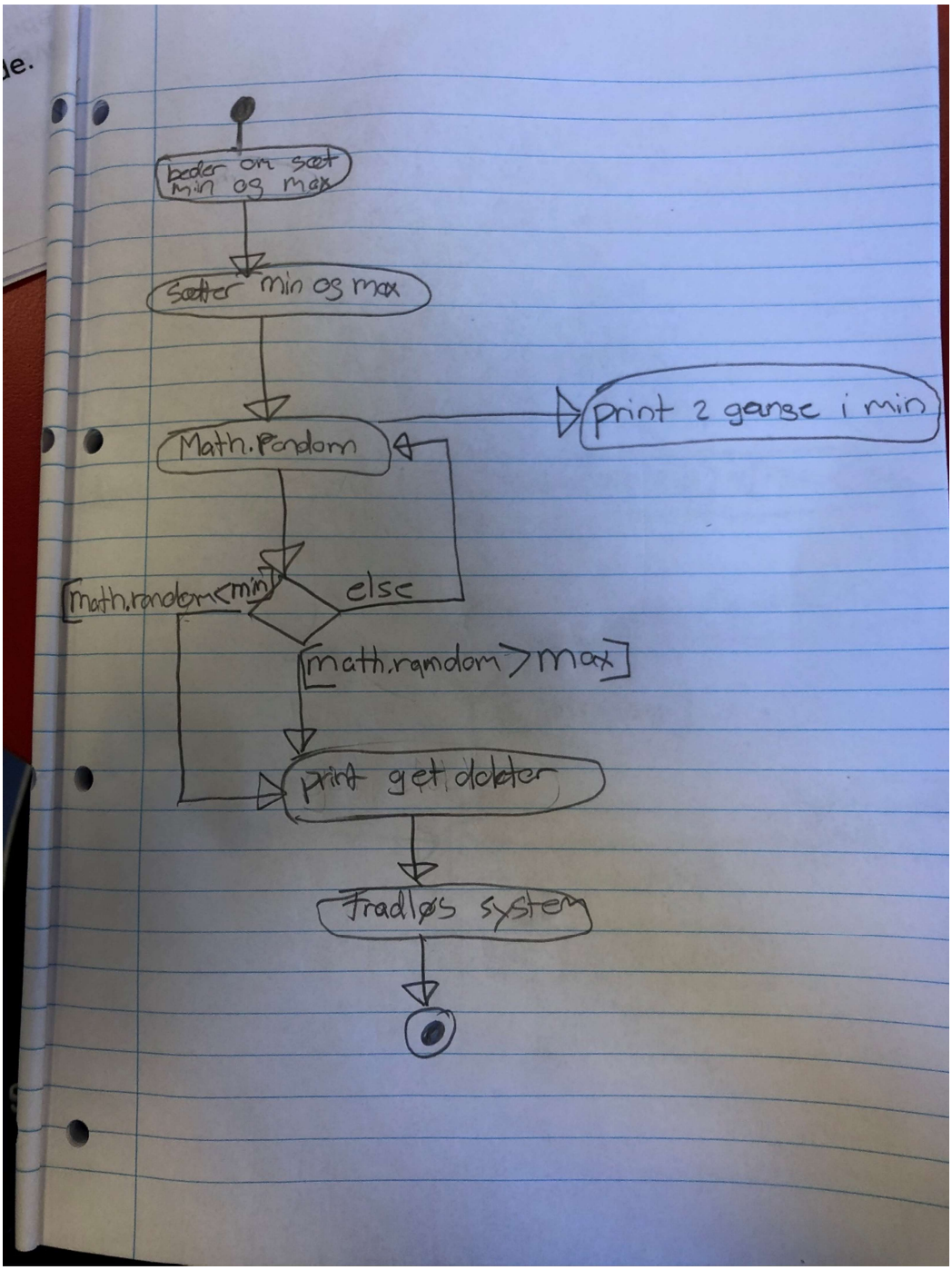
Figur 4 : Kollaborations-diagram (første iteration).

Bilag: 2



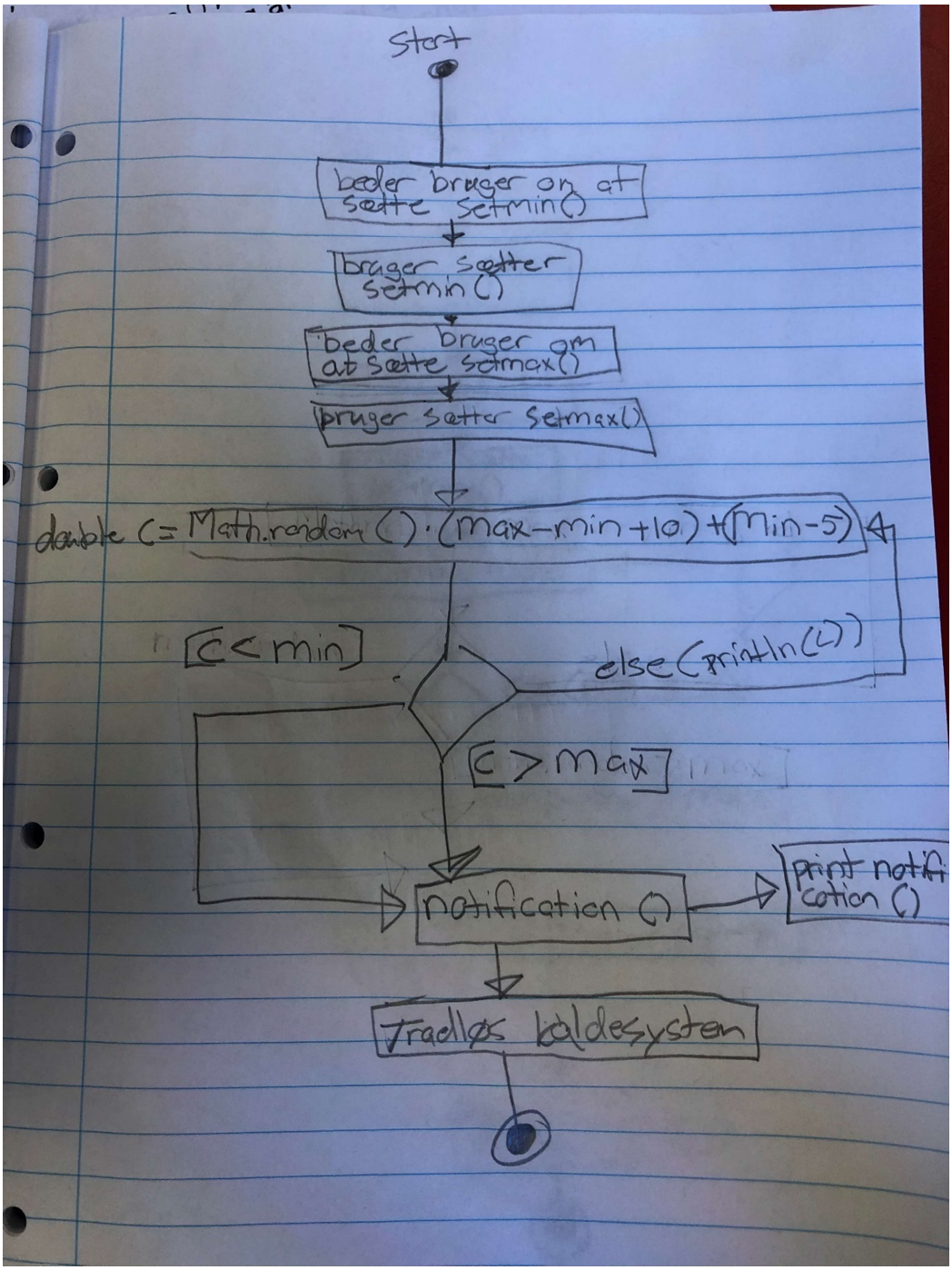
Figur 5: Kollaborations-diagram (andet iteration).

Bilag: 3



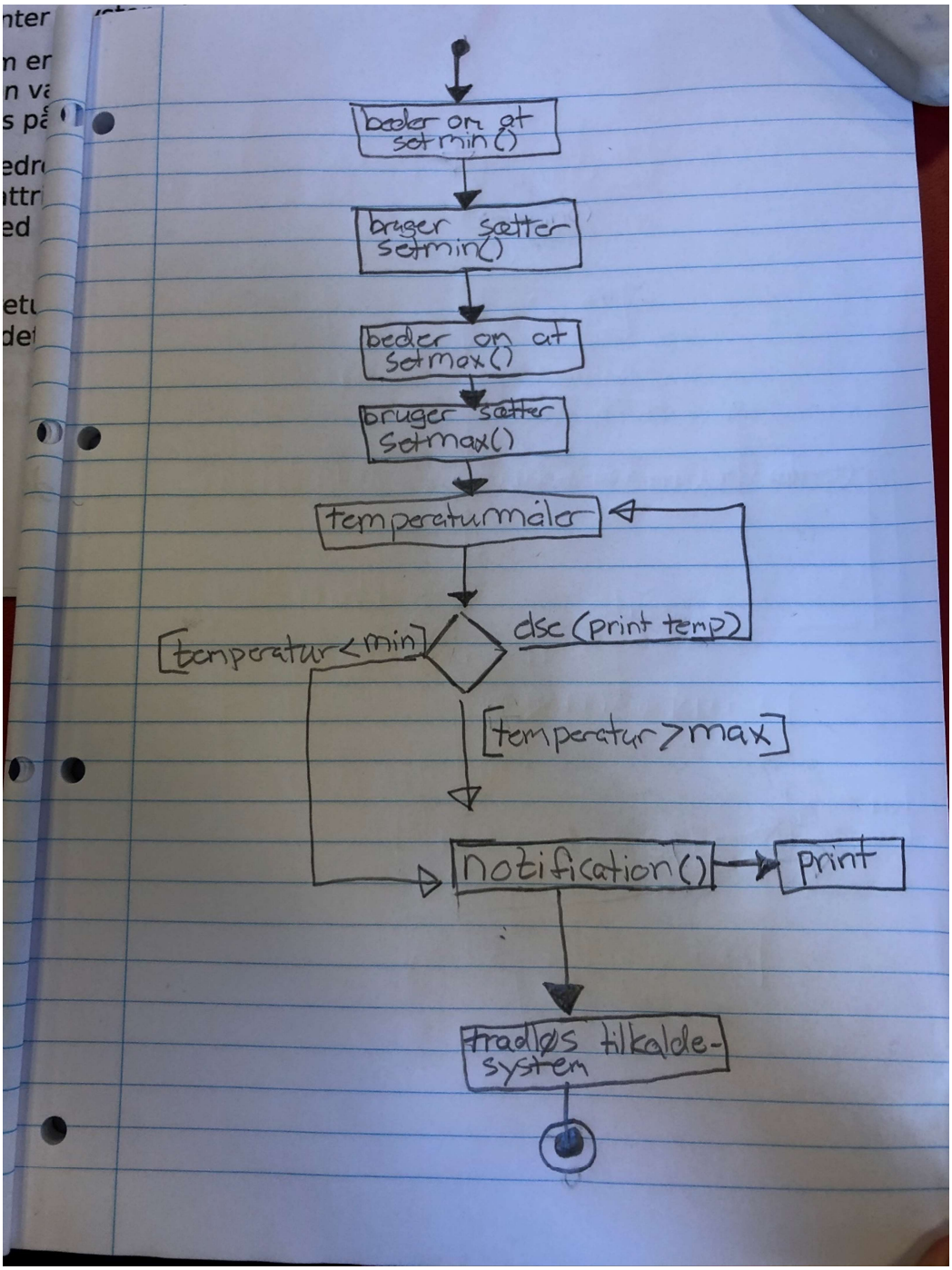
Figur 6: Aktivitets-diagram (første iteration)

Bilag: 4



Figur 7: Aktivitets-diagram (andet iteration)

Bilag: 5



Figur 8: Aktivitets-diagram (tredje iteration)