Ohjaus- ja analysointiohjelmiston kehitys maalämpöpumpulle

Mikko-Pekka Räisänen

Opinnäytetyö

Ammattikorkeakoulututkinto

Savonia
ammattikorkeakoulu

SAVONIA-AMMATTIKORKEAKOULU **OPINNÄYTETYÖ**
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Mikko-Pekka Räisänen	
Työn nimi Ohjaus- ja analysointiohjelmiston kehitys maalämpöpumpulle	
Päiväys 21.3.2012	Sivumäärä/Liitteet 46
Ohjaaja(t) Markku Kosunen	
Toimeksiantaja/Yhteistyökumppani(t) Itä-Suomen yliopisto / Jouni Huhtinen	

Tiivistelmä

Opinnäytetyö on tehty Itä-Suomen yliopistolle. Opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa IVT:n valmistamalle maalämpöpumpulle ohjaus- ja analysointiohjelmisto Java-kielellä.

Ohjaus- ja analysointiohjelmisto rakentuu kahdesta osasta, palvelin ja asiakasohjelmasta. Palvelin osan tarkoituksena on lukea maalämpöpumpusta CAN-väylän kautta tulevaa dataa ja ohjata lämpötilanpudotusta asiakasohjelmalta saatujen käskyjen mukaan. Datasta tunnistetaan eri anturien lähettämät lämpötilat ja muunnetaan Celsius-asteikkoon. Lämpötiloja voidaan näyttää näytöllä. Asiakasohjelma toimii graafisena käyttöliittymänä, josta voidaan nähdä yhdellä silmäyksellä eri lämpötila-arvoja. Käyttöliittymästä voidaan ohjata lämpötilan pudotusta.

Opinnäytetyössä käsitellään lämpöpumpun toimintaa ja rakennetta, käytettyjä laitteita ja ohjelmia, Java-ohjelmointikieltä ja historiaa. Javan ominaisuuksia ja sen rakenteita käsitellään ohjelmoinnin näkökulmasta. Käyttöliittymäsuunnittelun teoriaa käydään läpi lyhyesti. Pääasiallisena tarkastelun kohteena on Javalla toteutettu ohjaus- ja analysointiohjelmisto. Tarkastelussa perehdytään tarkemmin ohjelmiston toteutukseen, rakenteeseen ja ratkaisuihin.

Työn lopputuloksena on toimiva ohjelmisto, jolla voidaan seurata maalämpöpumpun toimintoja.

Avainsanat

Java, maalämpöpumppu, ohjelmistosuunnittelu, käyttöliittymäsuunnittelu, käyttöliittymä

SAVONIA UNIVERSITY OF APPLIED SCIENCES **THESIS**
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Mikko-Pekka Räisänen			
Title of Thesis The Development of Analysing and Control Software for a Ground Heat Pump			
Date	21.3.2012	Pages/Appendices	46
Supervisor(s) Markku Kosunen			
Project/Partners University of Eastern Finland / Jouni Huhtinen			

Abstract

The purpose of this thesis was to develop control and analysing software for a ground heat pump manufactured by IVT using JAVA programming language. The software is developed for University of Eastern Finland.

The software consists of two parts, server and client program. The meaning of server program is to read data from the ground heat pump's CAN-bus and to control temperature drop with instructions coming from the client. The software recognizes temperature values coming from different sensors and convert them into Celsius scale. It is possible to show temperature values at server's monitor. Client software works as a GUI (graphical user interface), where the user can see all the temperature readings at a glance. GUI also has a button to control temperature drop on and off.

The thesis covers the operating principles and structure of ground heat pump, JAVA-programming language and its history, JAVA's features and structures from a view of coder, the hardware and software used in thesis and theory of user interface development. The main part of the thesis is the development of control and analysing software for the ground heat pump.

The final result of thesis is functioning software.

Keywords

Java, ground heat pump, software development, user interface

SISÄLTÖ

KÄSITTEET

KÄSITTEET

DDR

Double Data Rate on muisteissa käytettävä tiedonsiirtotekniikka, jossa tietoa siirretään kellojakson molemmilla puoliskoilla.

FTP

File Transfer Protocol on suojaamaton tiedonsiirtoprotokolla, jota käytetään tiedon siirtämiseen kahden laitteen välillä tcp-yhteyden kautta. FTP käyttää tiedonsiirtoon porttia 21.

IDE

Integrated Development Environment tarkoittaa ohjelmointiympäristöä joka koostuu yleensä käyttöliittymästä ja kääntäjästä.

JDK

Java Development Kit laajempi versio Javasta joka sisältää kääntäjän ja tarvittavat ohjelmisto kirjastot Java-pohjaisten sovellusten kehittämiseen.

JVM

Java Virtual Machine tarjoaa virtuaaliympäristön jossa varsinainen Java-koodi ajetaan.

PHP

PHP: Hypertext Preprocessor on yleisesti web-palvelimissa käytettävä ohjelmointikieli jolla luodaan dynaamisia web-sivuja.

POE

Power Over Ethernet on tekniikka jolla voidaan syöttää käyttöjännite esim. lähiverkkokytkimelle kierrettyä parikaapelia pitkin.

SFTP

SSH File Transfer Protocol on verkkoprotokolla joka mahdollistaa tiedonsiirron suojattua yhteyttä pitkin kahden laitteen välillä.

SSH

Secure Shell tai SSH on verkkoprotokolla, joka mahdollistaa tiedonsiirron turvallista reittiä pitkin kahden laitteen välillä. SSH:n on alun perin kehittänyt Tatu Ylönen vuonna 1995. SSH:ta käytetään yleisimmin Unix/Linux pohjaisissa tietokoneissa yhteyden muodostamiseen. SSH:n tarkoituksena on salata turvattomien tietoverkkojen esim. internetin yli tapahtuva tiedonsiirto. Yleisin SSH:n käyttämä tiedonsiirtoportti on 22. (Barret D. J., Silverman R. 2001, 8-9)

1 JOHDANTO

Erilaisten vaihtoehtoisten lämmitysjärjestelmien määrä kasvaa Suomessa jatkuvasti. Näistä lämmitysjärjestelmistä ilma- ja maalämpöpumput ovat yksi osa-alue. Lämmityksessä yleensä käytettävä sähköenergian hinnan nouseminen on saanut ihmiset liikkeelle etsimään vaihtoehtoisia sähköä säästäviä lämmitysenergian lähteitä.

Useimmilta lämpöpumppujen valmistajilta itseltään ei löydy ohjelmistoa, jolla voitaisiin seurata pumpun toimintaa internetin kautta ja mahdollisesti ohjata pumpun toimintaa. Ohjelmistolle, jonka avulla voidaan seurata lämpöpumpun toimintaa, on kysyntää. Ohjelmiston tarpeesta johtuen päädyin Itä-Suomen ylipoistolle kesätyönä kehittämään ohjaus- ja analysointiohjelmistoa. Ohjelmistoa hyödynnetään maalämpöpumpun toiminnan tarkkailussa ja ohjauksessa internetin välityksellä. Ohjelmistoa on mahdollista käyttää myös lämpöpumpun yhteyteen liitetyssä PC:ssä paikallisesti, jolloin internet yhteyttä ei tarvita. Ohjelmistolla on mahdollista ohjata lämpötilanpudotusta, jolloin voidaan säästä energiaa esimerkiksi vapaa-ajan asunnon lämmityskustannuksissa. Lämpötilanpudotuksen etäohjaus mahdollistaa lämpötilan nostamisen normaaliin huonelämpötilaan ennen vapaa-ajan asunnolle saapumista lisäten asumismukavuutta. Ohjelmiston kehitys toimi samalla opinnäytetyönä.

Opinnäytetyössä käsitellään pääasiallisesti ohjaus- ja analysointiohjelmiston rakennetta ja toteutusta IVT:n maalämpöpumpulle. Opinnäytetyössä käydään läpi Java-ohjelmointikielen rakenteita ja historia läpi. Muita opinnäytetyössä käsiteltäviä asioita ovat käyttöliittymä suunnittelun teoria, työssä käytetyt laitteet ja ohjelmistot sekä taustatietoja Itä-Suomen yliopistosta.

2 ITÄ-SUOMEN YLIOPISTO

2.1 Yleistä

Itä-suomen yliopisto on julkisoikeudellinen laitos, joka aloitti toimintansa 1.1.2010. Se syntyi Kuopion ja Joensuun yliopistojen yhdistyessä. Yliopistossa opiskelee noin 14000 opiskelijaa ja se tarjoaa koulutusta monilla eri aloilla. Yliopisto toimii kahdella pääkampuksella, Joensuussa ja Kuopiossa. Lisäksi yliopistolla on sivukampus Savonlinnassa. Yliopisto työllistää noin 3000 henkilöä, joista noin 20 % työskentelee hallinnon ja toimistotehtävien parissa. Suurin osa muusta henkilöstöstä työskentelee tutkimus- ja opetustehtävissä. Hallinnosta vastaavat rehtorit, hallitus, dekaanit ja tiedekuntaneuvostot. Käytännön hallintotehtäviä varten on hallintokeskus ja tiedekuntien hallintopalvelukeskukset. Tiedekuntia on neljä: filosofinen, luonnontieteiden ja metsätieteiden, terveystieteiden, yhteiskuntatieteiden ja kauppatieteiden tiedekunta. (Itä-Suomen yliopisto, 2010)

2.2 Vahvuusalat

Itä-Suomen yliopisto on johtavassa asemassa suomalaisessa metsäalan tutkimuksessa. Tutkimuskohteina ovat metsät, ilmasto, muut luonnonvarat ja ihmisen elinympäristö. Tutkimuksissa tähdätään luonnonvarojen kestävään käyttöön. Kansansairauksien perusmekanismeja selvitetään molekyylilääketieteeseen perustuvien tutkimusten avulla. Tautien ennaltaehkäisyyn ja diagnostiikkaan luodaan uusia keinoja väestöpohjaisella ja kliinisellä tutkimuksella. Tutkimuskohteina ovat myös liikunnan, ravitsemuksen ja elämäntapojen terveydelliset merkitykset hyvinvoinnille. Keskeisiä tavoitteita ovat rajan yli tapahtuva yhteistyö ja monipuolinen Venäjä-osaaminen. Venäjä-osaamisella pyritään Euroopan ja Venäjän reuna-alueiden ja rajaseutujen tutkimuksessa kansainväliseen kärkeen. Tutkimuksen tarkoituksena on vahvistaa Venäjän elinkeinoelämään, kieleen ja kulttuuriin kohdistuvaa koulutusta ja tutkimusta.

Itä-Suomen yliopisto tarjoaa myös monipuolista opettajakoulutusta. Koulutuksessa tähdätään erityisopetuksen kehittämiseen ja luonnontieteiden aineenopettajien koulutuksen parantamiseen. Opettajakoulutuksen yksi tavoitteista on oppimisympäristöjen ja –teknologioiden käyttöönotto ja tutkimus.

Itä-Suomen yliopistolla on yhteistyösopimuksia lähes 70 eri ulkomaalaisen yliopiston kanssa ja on mukana lukuisissa vaihto-oppilas-ohjelmissa. Yliopisto hallinnoi myös pohjoismaisten yliopistojen Intia-verkostoa ja sillä on Nanjingin yliopiston kanssa yhteinen ympäristötutkimuskeskus The Sino-Finnish Environmental Research Centre (SFERC). SFERC keskittyy energia, ympäristö ja metsäalan tutkimukseen. (Itä-Suomen yliopisto, 2010)

3 MAALÄMPÖPUMPPU

3.1 Yleistä

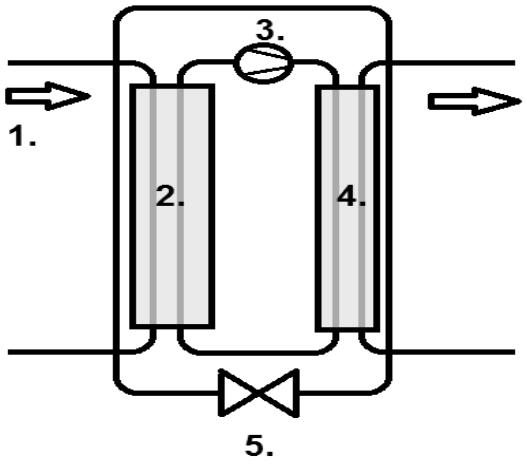
Maalämpöpumpulla hyödynnetään maaperään kesällä auringosta varastoitunutta tai veteen sitoutunutta lämpöenergiaa. Maasta saadulla lämpöenergialla lämmitetään käyttövettä ja/tai keskuslämmityksessä käytettävää vettä.

3.2 Lämmön talteenotto maasta

Lämpöenergia kerätään talteen maahan, kallioon tai veteen upotetulla putkistolla. Lämmönkeruuputkistossa kiertää jäätymätön liuos. Liuoksena on yleensä etanoli, jonka pakkasenkesto on -15°C. Lämmönkeruuputkisto asennetaan maaperään vaakasuoraan n. 0.7-1.2 metrin syvyyteen. Tarvittavan putkimäärän laskemisessa voidaan käyttää laskennassa 1-2 putkimetriä jokaista rakennuksessa olevaa lämmitettävää kuutiometriä kohden. Putkistolle tarvitaan tonttimaata noin 1,5m^3 jokaista putkimetriä kohti. (SULPU)

3.3 Lämpöpumpun toiminta

Putkistossa(1.) kiertävä neste siirtyy höyrystimeen(2.). Höyrystimessä nolla-asteinen neste kohtaa kylmäaineen. Nesteen kohdatessa kylmäaineen alkaa kylmäaine kiehua. Kiehuessaan kylmäaine muodostaa höyryä. Höyry johdetaan kompressoriin(3.). Kylmäaineen lämpötila nousee nollasta noin sataan asteeseen kompressorissa. Kompressorista kaasu johdetaan lauhduttimeen(4.). Lauhduttimessa lämpö siirretään talon lämmitysjärjestelmään. Höyry palautuu takaisin nestemäiseen olomuotoon. Kylmäaine johdetaan paisuntaventtiiliin(5.) jossa sen paine on edelleenkin suuri. Kylmäaineen paine laskee paisuntaventtiilissä. Kylmäaineen lämpötila laskee noin -10°C lämpötilaan. Kylmäaine palautuu takaisin höyrystimeen. (IVT, 6-8)



Kuva 1: Lämpöpumpun toiminta.

3.4 Säätökeskus

Lämpöpumpussa on säätökeskus huolehtimassa energian säästöstä ja lämpöpumpun kestävyydestä. Säätökeskus ohjaa kompressorin pyörimisnopeutta käyttö- ja lämmitysveden tarpeen mukaan. Lämmöntuottoa ohjataan ulkolämpötila anturilla tai ulko- ja sisälämpöanturien yhdistelmällä. (IVT, 9)

3.5 Käyttöveden tuotannon priorisointi

Talossa, jossa on vesikiertoinen lämmitysjärjestelmä, erotetaan käyttövesi ja lämmitysvesi toisistaan. Lämmitysvedellä lämmitetään lattialämmitystä tai lämpöpattereita. Käyttövesi on suihkuja ja hanoja varten. Käyttöveden tuotantoa varten voidaan lämpöpumppuun liittää ulkoinen lämminvesivaraaja. Käyttöveden lämpötila tunnistetaan säiliössä sijaitsevalla anturilla. Lämpötilan ollessa liian alhainen kierrättää lämpöpumppu kuumaa lämmitysvettä säiliön ulkovaipassa lämmittääkseen säiliön sisällä olevan veden riittävän lämpimäksi. (IVT, 9)

3.6 Ohjaus ulkolämpötila-anturilla

Ulkolämpötila-anturi asennetaan kylmimmälle ja varjoisimmalle ulkoseinälle. Säätökeskus ohjaa talon lämpötilaa anturin antamien arvojen mukaan. Lämmitysjärjestelmän lämpötilaa säädetään suhteessa ulkolämpötilaan säätökäyrän avulla. Säätökäyrää muokkaamalla on mahdollista saavuttaa huomattavia säästöjä energiankulutuksessa. (IVT, 9)

3.7 Ohjaus ulkolämpötila-anturilla ja huoneanturilla

Lämpöpumppuun liitetty sisälämpötila anturi kertoo säätökeskukselle hetkellisen huonelämpötilan. Sisälämpötila-anturin arvot vaikuttavat säätökäyrään. Huonelämpötila-anturia käytetään etenkin silloin jos huonelämpötilaan vaikuttaa muutkin tekijät kuin ulkolämpötila. Muita huonelämpötilaan vaikuttavia tekijöitä ovat esimerkiksi talossa oleva takka, talon alttius tuulelle tai suoralle auringonpaisteelle. (IVT, 9)

4 JAVA

4.1 Historiaa

Sun Microsystemin työntekijöitä James Goslingin johdolla päättävät aloittaa ohjelmointikielen kehittelyn kuluttajaelektroniikan tarpeisiin. Tarkoituksena oli kehittää ohjelmisto, joka toimisi erilaisissa tietokoneissa ja muissa elektronisissa laitteissa. Vuonna 1991 saatiin valmiiksi alustariippumaton ohjelmisto, jolle annettiin nimeksi Oak. Myöhemmin ohjelmisto jouduttiin nimeämään Javaksi patenttiongelmien vuoksi. Vuonna 1995 julkaistiin ensimmäinen versio Javasta, Java 1.0.
Javan kehitykseen on lainattu ominaisuuksia C, C++, ja Smalltalkista. Kehitysideana oli, että Java-ohjelmia voidaan kehittää ja ajaa millä tahansa alustalla, jossa on JVM. Java toimii useissa eri käyttöjärjestelmissä kuten Linuxissa, Windowsissa ja Solariksessa. (Kinnunen 2000, 2; Oracle)

4.2 Yleistä Javasta

Javan syntaksi muistuttaa C-kieltä. Javasta löytyy ohjelmointikielien perusrakenteet kuten ehtolauseet ja silmukat. Lauseet päättyvät puolipisteeseen. Aaltosulkeilla lauseet kootaan lohkoiksi. Lohkon sisällä olevat muuttujat eivät näy lohkon ulkopuolelle. Java on olio-ohjelmointi kieli, jossa voidaan käyttää luokkia, moduuleita, ja luokkien periyttämistä. (Kinnunen 2000, 25)
Java-ohjelmat ajetaan virtuaalikoneessa, jossa virtuaalikone tulkkaa Java-koodia lennosta tietokoneen prosessorin ymmärtämiksi käskyiksi. Virtuaalikone toimii hiekkalaatikkona, joka estää Java-ohjelmia vaikuttamasta suoraan muihin prosesseihin. Kaikki ohjelman käyttämät käskyt kulkevat virtuaalikoneen kautta. Java-ohjelmien ajo virtuaalikoneessa tekee niistä konekieliohjelmia turvallisempia, mutta samalla hiukan hitaampia. (Kinnunen 2000, 4)

4.3 Kommentit Java koodissa

Java koodiin on mahdollista lisätä kommentteja. Kommentit ohitetaan ohjelman kääntämisen aikana, joten ne eivät hidasta varsinaista ohjelman suoritusta. Yksinkertainen kommentti aloitetaan kahdella / -merkillä. Tällainen kommentti on voimassa koodirivin loppuun asti. Useampia rivejä ohjelmakoodia tai tekstiä kerrallaan voidaan asettaa kommentiksi sijoittamalla se /* ja */ merkkien väliin. (Kinnunen 2000, 12)

```
int d=0; // Muuttujan esittely

/*
     Useampi
     rivi
     kommentoituna
     c=5;
*/
```

Listaus 1: Koodin kommentointi

4.4 Muuttujat Javassa

Javassa on monia eri muuttujatyyppejä, joista yleisimmin käytettyjä ovat *int*, *double*, *boolean* ja *string*. Int on kokonaislukumuuttuja. Double-muuttujaa käytetään desimaaliluvuille. Boolean on totuusarvotyyppinen muuttuja, jonka arvoksi voidaan asettaa true (=tosi) tai false (=epätosi) . String-muuttujaan voidaan tallentaa merkkijonoja. Muuttujat täytyy esitellä ennen niiden käyttöä. Java koodissa muuttujat esitellään seuraavasti: <muuttujan tyyppi> <muuttujan nimi>; esimerkiksi `int tulos;`. Muuttujille on esiteltäessä hyvä asettaa oletusarvo. Oletusarvo voidaan laittaa esiteltäessä esimerkiksi `int tulos=0;`. Oletusarvojen asettamisella voidaan vähentää virheiden esiintymistä. (Kinnunen J. 2000, 13-15)

Taulukko1: Muuttujien minimi ja maksimi arvoja (Burd B 2005, 95; Deitel H. M. 2006, 55)

Muuttuja	Minimiarvo	Maksimiarvo
int	-2^{31}	$2^{31}-1$
double	$-1{,}8*10^{308}$	$1{,}8*10^{308}$

4.4.1 Taulukot

Muuttujista voidaan Javassa tehdä taulukoita, jotka voivat olla yksi tai moniulotteisia. Taulukkomuuttuja esitellään kuten tavallinen muuttuja, mutta lisäksi käytetään määrittelyssä [ ] merkkejä. Taulukkomuuttujat esitellään seuraavasti: <muuttujan tyyppi>[ ] <muuttujan nimi> = new <muuttujan tyyppi>[alkioiden määrä]. Esimerkiksi 6-alkioinen kokonaislukutaulukko esitellään seuraavasti: `int[ ] taulukko = new int[6];`. Taulukosta voidaan lukea arvoja tai kirjoittaa siihen arvoja viittaamalla taulukkomuuttujan indeksiin. On kuitenkin huomioitava, että taulukon ensimmäisen alkion indeksi on 0. Taulukon ensimmäiseen alkioon voidaan kirjoittaa numero 1 esimerkiksi seuraavasti: `taulukko[0]=1`. Indeksi numeron paikalla voidaan myös käyttää numeron sisältävää kokonaislukumuuttujaa. Moniulotteisen taulukon määrittelyssä käytetään useampia [ ] merkkejä ja alkioiden määrät erotellaan pilkulla toisistaan. Esimerkiksi kolmiulotteinen 3x3x3 double-tyyppinen taulukko määritellään `double [ ][ ] [ ] = new double[3,3,3]`. (Burd B 2005, 277-286)

4.5 Aritmeettiset operaatiot

Muuttujille voidaan tehdä myös erilaisia aritmeettisia operaatioita. Käytettävissä ovat yleisimmät aritmeettiset operaatiot kuten yhteen-, vähennnys-, jako-, ja kertolasku. Käytettävissä on myös jakojäännös. Muuttujan arvon kasvattamiseen yhdellä on olemassa kaksi tapaa. Ensimmäinen tapa on lisätä muuttujan arvoon 1 ja sijoittaa se muuttujaan itseensä esimerkiksi `a=a+1`. Toinen tapa arvon kasvattamiseen yhdellä on a++. Muuttujan arvoa voidaan myös näillä tavoilla pienentää käyttämällä + merkin sijasta – merkkiä.

Taulukko 2: Aritmeettiset operaatiot. (Deitel H. M. 2006, 55)

operaattori	operaatio	esimerkki	c:n arvo
+	yhteenlasku	c=1+1;	2
-	vähennys	c=1-2;	-1
/	jako	c=4/2;	2
*	kerto	c=2*2;	4
%	jakojäännös	c=11%5;	1

4.6 Vertailu- ja relaatio-operaattorit

Vertailu- ja relaatio-operaattoreilla voidaan tutkia muuttujien arvojen välisiä suhteita. Näitä käytetään etenkin ehtolauseiden ehdoissa.

Taulukko 3: Vertailuoperaattorit (Kinnunen J. 2000, 17)

<	pienempi kuin
<=	pienempi tai yhtä pieni kuin
>	suurempi kuin
>=	suurempi tai yhtä suuri kuin
==	yhtä suuri
!=	erisuuri

Taulukko 4: Relaatio-operaattorit (H. M. Deitel 2006, 210-230)

&&	ja
||	tai
^	poissulkeva tai
!	ei

4.7 Ehtolauseet

Ehtolause mahdollistaa eri koodin suorittamisen eri tilanteissa. Ehtolause alkaa if-sanalla, jonka jälkeen tulee itse ehto suluissa. Tämän jälkeen tulee aaltosuluissa koodi joka suoritetaan ehtolauseen ehdon täyttyessä. Ehtolause voi myös sisältää else osion, jonka jälkeen oleva koodi suoritetaan, mikäli ehto ei täyttynyt. Else-osio ei ole pakollinen. (Burd B 2005, 143-175)

```
int a = 2;
int b = 4;

if ( a < b )
     { System.out.println("Oli pienempi"); }
else
     { System.out.println("Oli suurempi"); }
```

Listaus 2: Ehtolauseen rakenne.

Ehtolauseessa voidaan käyttää useampia ehtoja. Useampia ehtoja käytettäessä se toteutetaan else if –rakenteella. Rakenteessa else if- sanojen jälkeen tulee uusi ehto suluissa ja seuraavalla rivillä suoritettava ohjelmakoodi aaltosuluissa. Else if-rakenteeseen voidaan myös sisällyttää else-osio, joskaan tämä ei ole pakollinen. (Burd B 2005, 143-175)

```
int a = 5;
int b = 4;

if ( a < b )
{ System.out.println("Oli pienempi"); }
else if ( a > b)
{ System.out.println("Oli suurempi"); }
else
{ System.out.println("Ei ollut kumpikaan"); }
```

Listaus 3: Else if-rakenne.

Ehtolauseiden ehdoissa voidaan myös yhdistää useampia ehtoja relaatio-operaattoreilla. Kuvassa esimerkiksi on ehtolause, jonka ehtona on, että a:n arvo täytyy olla pienempi kuin b:n arvo ja x:n arvo ei saa olla 5. (Burd B 2005, 143-175)

```
int a = 1;
int b = 3;
int x = 6;

if ( a < b && x != 5)
{ System.out.println("Ehto täyttyi"); }
else
{ System.out.println("Ehto ei täyttynyt"); }
```

Listaus 4: Ehtojen yhdistäminen.

4.7.1 Switch-lause

Switch-lauseeella voidaan toteuttaa monivalinta helposti. Switch-sanan jälkeen tulee sulkujen sisälle muuttuja, jonka arvon perusteella valinta tehdään. Muuttuja voi olla tyyppiä char, byte, short tai int. Case-sanan jälkeen tulee valintavakio, joka voi esiintyä vain kerran. Case-kohdan jälkeen aaltosuluissa tulee suoritettava ohjelmakoodi. Viimeisenä ohjelmakoodina lohkossa on `break;`. Break päättää switch-lauseessa olevan ohjelmakoodin suorittamisen. Mikäli break jätetään pois, suoritetaan eri case-kohtien ohjelmakoodia break-sanaan asti huolimatta valinta-arvoista. Switch-lause voi sisältää default valinta-arvon, jonka lohkossa oleva ohjelmakoodi suoritetaan, mikäli yksikään valinta-arvo ei täyttynyt. (Burd B 2005, 181-190; Kinnunen J. 2006)

```
int sijoitus=1;

switch(sijoitus)
{
	case 1:
	{
		System.out.println("Kultaa tuli");
		break;
	}
	case 2:
	case 3:
	{
		System.out.println("Ei tullut kultaa");
		break;
	}
	default:
	{
		System.out.println("Ei tullu mitallia  :/");
		break;
	}
}
```

Listaus 5: Switch-lause.

4.8 Silmukat

Silmukoita on kolmea eri tyyppiä. Näitä ovat While, Do-While ja For-silmukat. Silmukoiden avulla voidaan suorittaa yhtä tai useampaa koodiriviä haluttu määrä, jolloin vältytään samojen koodirivien kirjoittamisesta useampaan kertaan. Silmukkaa käytetään esimerkiksi luettaessa tiedostosta tekstiä rivi kerrallaan. Do ja Do-while -silmukuiden käyttö vaatii tiettyä huolellisuutta, jottei ohjelmoida niin sanottua ikuista silmukkaa. Ikuinen silmukka ei ole varsinainen ohjelmointivirhe, mutta ohjelman suoritus pysähtyy muilta osin ohjelman jäädessä suorittamaan silmukan sisältöä loputtomasti.

4.8.1 While –silmukka

While -silmukassa on alkuehto. Alkuehdolla määritellään kuinka monta kertaa silmukka ajetaan. On myös mahdollista, että while-silmukan alkuehto täyttyy jo heti alussa, jolloin koko silmukkaa ei suoriteta. (Kinnunen J. 2000, 27)

```
int aa=1;
while (aa<6)
{
	System.out.println("Kierros: " + aa);
	aa++;
}
```

Listaus 6: While-silmukka.

4.8.2 Do-while –silmukka

Do-while –silmukassa on loppuehto. Do-while –silmukan sisällä oleva ohjelmakoodi ajetaan ainakin kerran. (Kinnunen J. 2000, 30)

```
int bb=6;
do
{
	System.out.println("Kierros: " + bb);
	bb++;
}
while (bb<6);
```

Listaus 7: Do-while-silmukka.

4.8.3 For –toisto

For –toiston alkuehto koostuu kolmesta osasta, jotka ovat alkuasetus, jatkamisehto ja eteneminen. Alkuasetus on muuttuja joka sisältää aloitusarvon. Muuttuja voidaan myös esitellä toistolauseen ehto-osiossa. Jatkamisehdossa alkuasetuksessa annetun muuttujan arvoa verrataan vertailuoperaattorilla haluttuun arvoon. Eteneminen määrittelee sen kuinka muuttujan arvo kasvaa tai pienenee. Ehdot eritellään toisistaan puolipisteellä. For –toistosta on myös mahdollista tehdä ikuinen laittamalla ehto sulkujen sisälle kaksi puolipistettä. (Kinnunen J. 2000, 29)

```
for (int i=1;i<6;i++)
{
	System.out.println("Kierros: " + i);
}
```

Listaus 8: For-toisto.

4.9 Säikeistäminen

Ohjelmassa käytetään säikeistystä käyttöliittymän palvelimelle ja lämpötilojen tulostukselle näytölle. Sarjaportista tiedon lukemiseen käytetään omaa säiettä. Säikeistämällä saadaan eri toiminnalliset kokonaisuudet toimiman samanaikaisesti samassa ohjelmassa niiden häiritsemättä toisiaan. Liian monen säikeen käyttäminen voi johtaa ohjelmiston hidastumiseen. Säikeistystä käytettäessä tulee huolehtia että useampi säie ei yritä kirjoittaa arvoja samaan muuttujaan samanaikaisesti, mikä voi johtaa tiedon tallentumiseen virheellisesti. Mikäli tarvitaan saman muuttujan arvo lukea tai tallentaa eri säikeistä, voidaan Javassa käyttää synkronointi määritettä. Esimerkiksi luokassa määritellään muuttuja johon halutaan tallentaa tietoa. Muuttujan arvo voidaan lukea tai asettaa turvallisesti käyttämällä luokassa synkronoituja metodeja listauksen 9 esimerkin mukaisesti. (Kinnunen J. 2000, 61-63)

```
private int i = 0;

public synchronized int Get_i()
     {
          return this.i;
     }
public synchronized int Set_i(int x)
     {
          this.i=x;
     }
```

Listaus 9: Muuttuja arvon turvallinen lukeminen ja asettaminen synkronoituja metodeja käyttäen.

4.10 Poikkeukset

Try/Catch rakenteella hoidetaan Javassa virheet ja odottamattomat tilanteet. Tällä voidaan estää ohjelman kaatuminen hallitsemattomasti. Try-lohkon sisään sijoitetaan suoritettava koodi. Virhetilanteet käsitellään catch-lohkolla. Catch-lohkoja voi olla useampia samalle try-lohkolle, jos halutaan käsitellä eri virheitä eri tavalla. Catch-lohkot käsitellään niiden kirjoitusjärjestyksessä. Try/catch-rakennetta käyttäen voidaan estää klassinen nollalla jakovirheen ilmentyminen. (Kinnunen J. 2000, 34-35)

```
int a=5;
int b=0;
int vastaus=0;

try
{
     vastaus=a/b;
     System.out.println("Vastaus on " +vastaus);
}
catch(Exception e)
{
     System.out.println("Oho, meinasit jakaa nollalla");
}
```

Listaus 10: Try-catch-rakenne.

5 KÄYTTÖLIITTYMÄSUUNNITTELUN TEORIAA

5.1 Käyttöliittymäsuunnittelun ongelmakohtia

Käyttöliittymäsuunnittelussa aiheutuu ongelmia useista eri tekijöistä. Kehittymättömät työkalut ja suunnittelumenetelmät aiheuttavat ongelmia sovellusten käydessä yhä monimutkaisemmiksi. Jatkuvasti muuttuvat asiakkaan tarpeet voivat aiheuttaa ongelmia. Eri käyttäjillä ja käyttäjäryhmillä on erilaisia tarpeita. (Kinnunen, 7)

5.2 Käytettävyyssuunnittelu

Käytettävyyssuunnittelun avulla on mahdollista vaikuttaa käyttöominaisuuksiin. Hyvällä suunnittelulla saadaan tietoa käyttäjien työtavoista, tarpeista ja työympäristöstä. Käytettävyyssuunnittelun avulla on mahdollista saada tietoa työskentelyn aikana ilmenevistä ongelmista. (Kinnunen, 13-14)

5.3 Suunnittelulla säästöjä

Hyvällä suunnittelulla on mahdollista saavuttaa säästöjä. Säästöjä saadaan esimerkiksi työvaiheiden suorittamiseen kuluvan ajan pienenemisellä, ohjelmiston käyttöön tarvittavan koulutustarpeen supistumisella ja tukihenkilöiden työmäärän vähenemisestä. (Kinnunen, 15-16)

5.4 Laaduntekijät

Ohjelmiston laatuun vaikuttavat erilaiset tekijät esimerkiksi suorituskyky, ominaisuudet, luotettavuus ja esteettisyys. Suorituskyvyllä on iso vaikutus ohjelmiston käytettävyyteen ja laatuun. Hitaasti toimiva ohjelma ei anna laadukasta kuvaa itsestään ja sen käytettävyys on huono. Liialliset ominaisuudet voivat tehdä ohjelmistosta käytettävyydeltään hankalan. Liian suppeat ominaisuudet tai huonosti toteutettu ulkoasu puolestaan voivat antaa ohjelmistosta keskeneräisen kuvan. Luotettavuus on suoraan verrattavissa ohjelmiston laatuun. (Kinnunen, 17)

5.5 Käytettävyystavoitteet

Tavallisesti toiminnallisuus määritellään käytettävyyttä paremmin. Tavoitteet voidaan usein asettaa käyttäjän kanssa. Tietolähteinä tavoitteiden asettamisessa voidaan käyttää lopullisia käyttäjiä. Tavoitteiden asettamisen pohjana voidaan käyttää toimiviksi koettuja ominaisuuksia ja tunnettuja käytettävyysongelmia. (Kinnunen, 18-21)

5.6 Suunnitteluprosessi

Suunnitteluprosessi voidaan toteuttaa käyttöliittymäpainotteisena tai toimintopainotteisena. Käyttöliittymäpainotteisen suunnittelun mahdollisena ongelmana ovat mahdollisesti puutteelliseksi tai kokonaan pois jäävät toiminnot. Toimintopainotteisessa

suunnittelussa toteutetaan ensi toiminnot, jonka jälkeen toteutetaan käyttöliittymän suunnittelu. Yksityiskohtaisella suunnittelulla saadaan pääosa toiminnallisuudesta selville. Suunnittelussa voidaan käyttää apuna erilaisia työkaluja, esimerkiksi Microsoftin Visiota ja Exceliä. Työkaluilla voidaan luoda esimerkiksi erilaisia kaavioita kuvaamaan ohjelman toimintoja, graafisia piirroksia ja tekstimuotoisia kuvauksia. (Kinnunen, 25-27)

5.7 Käytettävyystestaus

Käytettävyydeltään hyvää sovellusta on hankala toteuttaa ilman käytettävyystestausta. Testaus voidaan suorittaa monella eri tavalla. Peruskäyttäjien käyttäminen testauksessa on paras tapa. Ohjelmoijan itse tekemä testaus toimii lähinnä toiminnallisuuden testauksena. Testauksessa on hyvä käyttää useampaa testaajaa, jotta pienemmät käytettävyysongelmat löytyvät paremmin. Testauksessa voidaan käyttää myös maksullisen ammattilaisen apua. (Kinnunen, 104)

5.8 Testattavat asiat

Testauksessa testataan monia asioita. Testauksella selvitetään asetettujen tavoitteiden ja tarpeiden toteutumista. Tositilanteessa tapahtuvalla käytön nopeudella voidaan selvittä muun muassa mahdolliset käytettävyyden pullonkaulat. Toistuvien, kriittisten ja satunnaisten virheiden määrät voidaan selvittää huolellisella testauksella. (Kinnunen, 105)

5.9 Korjaukset

Testaamisen jälkeen havaittuihin ongelmiin voidaan kehittää korjauksia. Korjausten toteuttaminen voi tuoda mukanaan uusia ongelmia. Ongelmia korjatessa tulee varmistaa, että muuta kuin virheen aiheuttajaa ei muuteta. Korjaukset tulisi tehdä nopeasti testauksen jälkeen. Korjausten jälkeen testataan uudelleen mahdollisien korjauksessa tulleiden virheiden varalta. (Kinnunen, 114)

5.10 Testauksessa tapahtuvat virheet

Mikäli testivaihetta ylipäätään on, tehdään siinä usein eniten virheitä. Testauksessa vältettäviä virheitä, jotka vaikuttavat testituloksiin on monia. Esimerkiksi suunnittelijoiden käyttöä testaajina, testikäyttäjän auttaminen testitilanteessa, virheiden/testitapahtumien puutteellinen kirjaaminen ja testaamiseen ei varata tarpeeksi aikaa. Testaamista loppuvaiheessa tulisi välttää, koska silloin on usein liian myöhäistä tehdä tarvittavia muutoksia. (Kinnunen, 115)

6 OPINNÄYTETYÖSSÄ KÄYTETYT LAITTEET JA OHJELMISTOT

6.1 Putty

Putty on SSH ja telnet asiakasohjelma. Windows alustalle alun perin Puttyn on kehittänyt Simon Tatham. Putty on avoimen lähdekoodin ohjelma, jonka lähdekoodi on vapaasti ladattavissa. Puttyä kehittää joukko vapaaehtoisia. (Putty)

6.2 WinSCP

WinSCP on ilmainen vapaan lähdekoodin tiedonsiirto-ohjelma. WinSCP tukee SFTP (SSH File Transfer Protocol) ja FTP (File Transfer Protocol) tiedonsiirtoprotokollia. Ohjelma mahdollistaa turvallisen tiedonsiirron suojattua yhteyttä pitkin. (WinSCP)

6.3 NetBeans

NetBeans on ilmainen sovelluskehitysympäristö. NetBeans tukee myös muitakin ohjelmointikieliä, kuten PHP, JavaFX, C, C++, Javascript.
NetBeans on avoimen lähdekoodin projekti joka pyrkii täyttämään sovelluskehittäjien, käyttäjien ja yritysten tarpeet.
Kaksi ilmaista perustuotetta kaupalliseen ja yksityiseen käyttöön ovat NetBeans IDE ja NetBeans Platform. Lähdekoodi on saatavilla molempiin tuotteisiin ja sitä voidaan käyttää lisenssin sallimissa rajoissa.
NetBeans projekti on energinen yhteisö, missä ihmiset voivat esittää kysymyksiä, antaa neuvoja ja myötävaikuttaa eri tavoin.
NetBeans kehitysalustaa on ladattu yli 18 miljoonaa kertaa ja yli 800 000 ohjelmistokehittäjää on mukana projektissa. NetBeans on menestyvä ja kasvava projekti josta kiitos kuuluu yksityisille ja yhteistyökumppaneille. (NetBeans)

6.3.1 Historiaa

NetBeansin lähdekoodi julkaistiin vapaasti saataville kesäkuussa 2000 Sun Microsystemin toimesta. Sun Microsystems toimi projektin tukijana vuoden 2010 tammikuuhun asti jolloin siitä tuli Oraclen tytäryhtiö.
NetBeansin kehitys alkoi Tsekin tasavallassa 1996 opiskelijaprojektina. NetBeansin alkuperäinen kehitysnimi oli Xelfi. Projektin tarkoituksena oli tehdä Delphin kaltainen Java-kehitysympäristö. Xelfi oli ensimmäinen Java kehitysympäristö, joka oli ohjelmoitu Javalla. Ensimmäiset versiot julkaistiin 1997. Vuonna 1999 NetBeans DeveloperX2 julkaistiin. Tässä versiossa oli tuki Swing-komponenteille. Kesäkuussa 1999 NetBeansista tuli osa Sun Microsystemiä. (NetBeans)

6.4 Eclipse

Eclipse on avoimen lähdekoodin projekti. Projektin tarkoituksena on kehittää kehitysympäristö jota voidaan laajentaa. Eclipseä voidaan käyttää myös muiden kuin Java sovellusten kehittämiseen. Eclipsesäätiö on voittoa tavoittelematon. Säätiö itse ei tee varsinaista kehitystyötä vaan se tehdään avoimen koodin periaatteella vapaaehtoisen työvoiman avulla. Työvoimana on yksityisiä ihmisiä ja yrityksiä.
Eclipsen aloitti alun perin IBM vuonna 2001. Vuonna 2004 aloitti toimintansa voittoa tavoittelematon yritys Eclipse säätiö, jonka tarkoituksena on edustaa yhteisöä. Yritys luotiin puolueettomuuden ja avoimuuden takaamiseksi.
Eclipse säätiö saa rahoituksensa keräämällä vuosimaksuja jäseniltään. Säätiötä johtaa johtajista koostuva johtokunta. Tärkeitä asiakkaita ja kehittäjiä kuuluu johtokuntaan. Johtokunnassa on myös äänestämällä valittuja jäseniä. Säätiön palkkalistoilla on myös vakituisia työntekijöitä huolehtimassa palveluista yhteisölle. Ohjelmistokehittäjiä , jotka työskentelevät Eclipse-projektien parissa- ei säätiö työllistä.
Eclipse on ladattavissa ilmaiseksi osoitteesta http://www.eclipse.org/ . Tuotetukea saa Eclipsen omilta keskustelualueilta internetistä. (Eclipse)

6.5 RXTX

RXTX on kolmannen osapuolen kehittämä Java kirjasto, jonka avulla voidaan käyttää rinnakkais- ja sarjaportteja JDK:n kanssa. RXTX on julkaistu GNU LGPL-lisenssin alaisuudessa. Kirjasto pohjautuu Sun:in tietoliikenneohjelmointirajapintaan. Kirjasto on suurimmilta osin yhteensopiva Suni:in kirjaston kanssa. Kirjasto toimii Windows/Linux-käyttöjärjestelmillä. Kirjaston käyttäminen vaati JDK:n asentamisen.(RXTX)

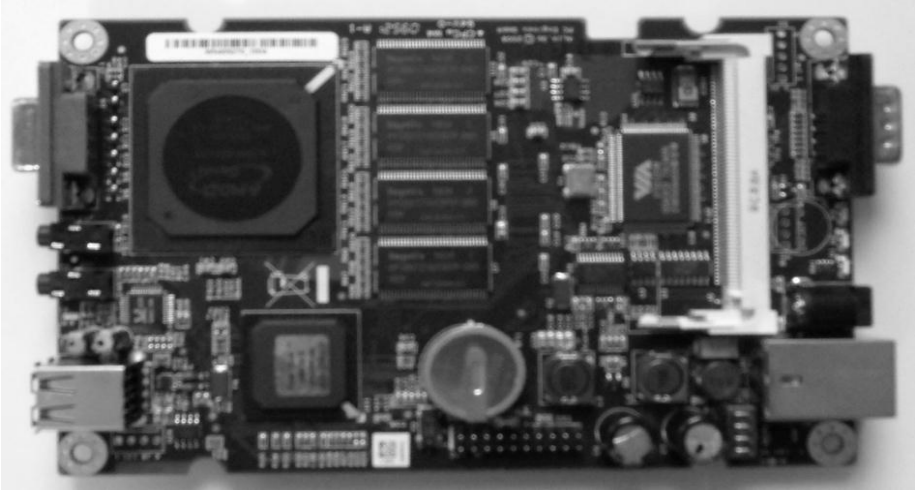
6.6 ALIX3D3

Alix3d3 on PC Engines yhtiön valmistama sulautettu-PC. Sulautetut tietokoneet ovat pienempiä ja vievät vähemmän virtaa kuin normaalit pöytäkoneet. Sulautettuja tietokoneita käytetään esimerkiksi teollisuudessa käyttöpäätteinä, palomuureina ja erilaisina verkkopäätteinä tai palvelimina.

6.6.1 Teknisiä tietoja

Alix3d3:n prosessorina on AMD:n 500MHz Geode prosessori. Prosessori on passiivijäähdytteinen eli siinä ei ole tuuletinta. DDR keskusmuistia löytyy 256MB. Massamuistille on CF-muistikorttiliitin. Ulkoisia laitteita varten on kaksi USB-liitintä. Näytön saa liitettyä tarvittaessa VGA-liitännän kautta. Laitteessa on myös kaksi miniPCI-liitintä. Ethernet-liitäntä mahdollistaa verkkokäytön. Ääniliittiminä on kuuloke- ja mikrofoniliitin. Käyttöjännitealue on 7V–20VDC. Tehonkulutus normaalikäytössä on noin 7-8W ja maksimi tehonkulutus 18–20W. Käyttöjännitteen syöttämiseen voidaan käyt-

tää myös POE:ta. Käyttöjärjestelmänä voidaan käyttää Microsoft Windows XP:tä, Linuxin eri jakeluversioita ja FreeBSD:n eri versioita. Laitteen fyysiset mitata ovat 100 x 160 mm. Laitteesta löytyy kolme led-valoa, joiden liittimiä voidaan käyttää GPIO:na. (PC Engines)



Kuva 2: Alix.

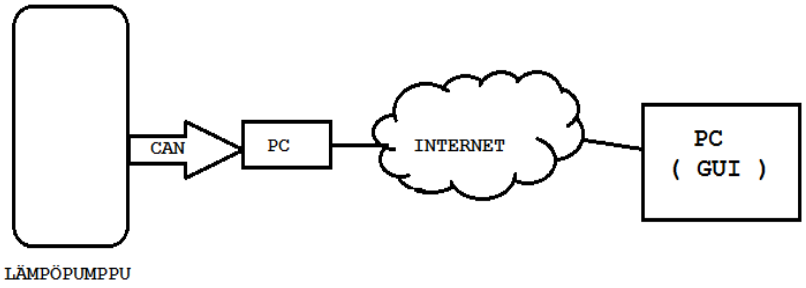
6.6.2 GPIO

GPIO mahdollistaa merkkiledien käytön lähtöinä ja tuloina. GPIO porttien toiminta voidaan määrittää ohjelmallisesti joko tuloiksi tai lähdöiksi. Tällöin on mahdollista ohjata muita laitteita GPIO:n kautta tai lukea ohjelmallisesti portista tilatieto. Porttia ohjataan yksinkertaisesti kirjoittamalla sinne 1 tai 0. Vastaavasti portista luettaessa saadaan sieltä lukuarvo 1 tai 0.

7 OHJELMISTON TOTEUTUS

7.1 Taustatiedot

Lämpöpumpulta luetaan dataa PC:lle. Data liikkuu CAN-väylässä lämpöpumpulla. PC:hen on liitetty gsm modeemi usb-väylään nettiyhteyttä varten. Nettiyhteyttä käytetään ohjelmiston siirtämiseen PC:lle etäkäyttönä ja käyttöliittymä hakee lämpötilat internetyhteyttä pitkin.



Kuva 3: Ohjelmiston toiminnallinen idea.

Kuva 4: Lämpöpumppu pannuhuoneessa.

7.2 Ohjelmointikielen valinta

Ohjelmointikielen valinnassa oli rajoittavana tekijänä kohdelaitteen käyttöjärjestelmä (Linux). Ohjelmointikieliä, joilla ohjelmiston olisi voinut toteuttaa, on lukuisia. Esimerkiksi C, C++, Java ja jopa PHP. C, C++ ja PHP:tä voidaan käyttää niin Windows kuin Linux alustalla. Ohjelmointikielen valinnassa päädyttiin Javaan, jolla tehdyt ohjelmat toimivat niin Windowsissa kuin Linuxissa. Javan valintaan ohjelmointikieleksi vaikuttivat myös koulussa käydyt Java-ohjelmointikurssit, jotka antoivat hyvän perustietouden Java-ohjelmoinnista.

7.3 Ohjelmiston suunnittelu ja toteutus

Suunnittelua ei ohjelmiston osalta varsinaisesti tehty. Käyttöliittymän ulkoasun suunnittelussa käytettiin toista ohjelmistoa suuntaa antamassa. Ohjelmiston suunnittelun apuna oli myös Perl-kielinen ohjelmisto toiselle lämpöpumpun versiolle. Ohjelmiston suunnittelussa päädyttiin ratkaisuun, jossa ohjelmisto rakentuu kahdesta eri osasta. Ensimmäinen osa (Reader) lukee tiedot lämpöpumpulta ja toinen osa (GUI) on käyttöliittymä, josta näkee lämpötilat.

7.4 READER

Ohjelmisto on jaettu erilaisiin luokkiin joista jokainen huolehtii luokalle määrätyistä tehtävistä. Luokkia on mm. asetustiedoston lukemiselle, sarjaportista lukemiselle, lämpötilojen tallentamiseen ohjelmiston sisällä muuttujiin ja lämpötilojen tulostamiseen näytölle. Ohjelmiston jakamisella luokkiin saadaan ohjelmakoodista helpommin hallittava ja selkeämpi kuin kirjoittamalla kaikki ohjelmakoodi yhteen tiedostoon. Luokkien käyttö uudelleen toisissa projekteissa on myös mahdollista.

7.4.1 Ohjelman rakenne ja toiminta

Java-ohjelmissa tärkeä osa on pääluokka, joka on samanniminen kuin itse ohjelma. Pääluokalla on yleensä sama nimi kuin ohjelmalla. Pääluokan tärkein osa on main-metodi. Main-metodi ajetaan aina automaattisesti ohjelman käynnistyessä. Main-metodin sisälle sijoitetaan ohjelmakoodi jota tarvitaan ohjelman käynnistyessä ja jota ei voi sijoittaa muihin luokkiin. Main metodin sisälle kirjoitettava koodi olisi hyvä pitää mahdollisimman lyhyenä selkeyden vuoksi.

```
public static void main ( String[] args )
     {

     }
```

Listaus 11: Main-metodi.

Ohjelmistossa main-metodissa esitellään boolean-tyypiset muuttujat. Muuttujien esittelyn jälkeen luodaan tarvittavat oliot luokista. Olioita voidaan välittää parametreina luokille. Olioiden luonnin jälkeen ohjelmistossa kutsutaan olion `cfgr`-metodia `ReadConfigFile()`. Metodi asetustiedosta lukemisen ja asetustietojen tallentamisen `rset`-olioon. Metodin palauttama arvo tallennetaan boolean-tyyppiseen muuttujaan `configOk`.

```
boolean configOk=false;

ReaderSettings rset=new ReaderSettings();
ConfigFileReader cfgr= new ConfigFileReader(rset);

configOk = cfgr.ReadConfigFile();
```

Listaus 12: Olioiden luonti.

Ehtolauseessa muuttujan `configOk` arvoa verrataan arvoon true (tosi). Mikäli muuttujan arvo on sama verrattavan arvon kanssa, suoritetaan ehtolauseen sisällä oleva ohjelmakoodi. Muuttujan arvo false (epätosi) tarkoittaa että asetustiedoston lukemisessa on tapahtunut virhe, jolloin ehtolauseen sisältämää koodia ei suoriteta ja siirrytään main metodin loppuun jolloin ohjelman suoritus päättyy.

Edellisen ehtolauseen sisällä tarkistetaan tiedoston `pudotus` olemassaolo uuden ehtolauseen ehdossa. Tarkistus tapahtuu muuttujan `pudotusStatFile` metodilla `exists()`, joka palauttaa arvon true tai false riippuen tiedoston olemassaolosta. Muuttujan `pudotusStatFile` tyyppi on tiedosto. Tiedosto on fyysinen tyhjä tiedosto, jonka olemassaolo ilmaisee lämpötilanpudotuksen tilan. Tiedosto tarvitaan, jotta ohjelma muistaisi lämpötilanpudotuksen viimeisimmän ohjelman suorituksen keskeytyessä esimerkiksi sähkökatkon vuoksi. Ehtolauseessa kutsutaan `rVal`-olion metodia `Disable()` tai `Enable()` pudotustiedoston olemassaolon perusteella, jolloin lämpötilan pudotus laitetaan päälle tai pois päältä.

```
File pudotusStatFile = new File("pudotus");

if(pudotusStatFile.exists())
{
```

Listaus 13: Tiedostomuuttujan luonti ja olomassaolon tarkistaminen.

Sarjaporttiyhteys avataan kutsumalla `scom`-olion metodia `connect()`, jolle annetaan parametrina kokonaislukumuuttuja `port`. Olio `scom` luodaan luokasta `SerialComm-Comm`. Muuttuja sisältää yhteydessä käytettävän portin numeron. Yhteyden avaaminen on try/catch rakenteen sisällä. Ensimmäisellä catch-lohkolla otetaan kiinni `NoSuchPortException`, jos avattavaa porttia ei ole olemassa. Toisella catch lohkolla otetaan kiinni yleiset virheet (Exception). Catch-lohkot asettavat boolean `portOk`-muuttujan arvoksi false.

```
try
{
     scom = new SerialCommComm(rset, rVal);
     scom.connect(port);
}
catch (NoSuchPortException nsp)
{
     ......
```

Listaus 14: Portin avaaminen.

Portin avaamisen jälkeen käynnistetään säikeet käyttöliittymälle ja tietojen tulostamiseksi näytölle. Säikeiden käynnistäminen on ehtolauseen sisällä. Ehtolauseen suorittamisen ehtona on, että `portOk`-muuttujan arvo on true. Lisäksi näytölle tulostuksella on oma ehtolause, jonka ehdossa verrataan `rset`-olion muuttujan `printtoscreen` arvoa arvoon true. Arvon ollessa true, käynnistetään säie.

7.4.2 SerialCommComm-luokka

Luokka hoitaa sarjaporttiyhteyden luonnin ja käynnistää säikeen jolla luetaan dataa sarjaportista. Luokan tärkein osa on metodi `Connect()`. Metodille annetaan parametrina sarjaportti johon yhteys luodaan. `ComPortIdentifier` tyyppiseen muuttujaan sijoitetaan arvo joka saadaan kutsumalla `ComPortIdentifier`-luokan metodia `getPortIdentifier` parametrilla portName. Parametri portName sisältää portin nimen merkkijonona. Portin avaamisen ensimmäisessä vaiheessa tarkistetaan ehtolauseen ehdolla, että portti ei ole jo käytössä. käytössä olevasta portista annetaan käyttäjälle ilmoitus näytölle. Toisessa vaiheessa luodaan `CommPort` tyyppinen muuttuja. Muuttujasta tarkistetaan ehtolauseen ehdolla, että se on tyypiltään sarjaportti. Portin ollessa sarjaportti luodaan `SerialPort` tyypin muuttuja. Muuttujan metodilla `setSerialPortParams` asetetaan sarjaportin asetukset kuten nopeus ja pariteetti. Syötevirta in luodaan `serialPort` muuttujan metodilla `getInputStream()`. Viimeisessä vaiheessa luodaan säie `SerialReader-luokasta`, jolle annetaan parametrina syötevirtamuuttuja `in`, `rset` ja `rVal` oliot. Säie käynnistetään kutsumalla metodia `start()`.

```
CommPortIdentifier portIdentifier = CommPortIdentifier.getPortIdentifier(portName);
if ( portIdentifier.isCurrentlyOwned() )
{
  System.err.println("Error: Port is currently in use");
}
else
  {
    CommPort commPort = portIdentifier.open(this.getClass().getName(),2000);

  if ( commPort instanceof SerialPort )
  {
    SerialPort serialPort = (SerialPort) commPort;
    serialPort.setSerialPortParams( this.rset.port_speed,SerialPort.DATABITS_8,
                                    SerialPort.STOPBITS_1,SerialPort.PARITY_NONE );

    InputStream in = serialPort.getInputStream();

    (new Thread(new SerialReader(in, this.rset, this.rVal))).start();
```

Listaus 15: Yhteyden luonti ja säikeen käynnistys.

7.4.3 SerialReader luokka

`SerialReader` luokan tarkoituksena on lukea sarjaportista dataa. Parametrina saadusta syötevirrasta luodaan puskuroitu syötevirta-olio. Lukeminen tapahtuu `run()`-metodissa silmukan sisällä. Silmukan ehdossa tallennetaan `line` nimiseen merkkijonomuuttujaan puskuroitu syötevirta metodilla `readLine()`. Muuttujan `line` sisältöä verrataan poissulkevasti null arvoon. Arvon ollessa null silmukan sisällä olevan koodin suoritus keskeytyy. Silmukassa on ehtolause, joka tarkistaa että sarjaportista tuleva tieto on oikeassa muodossa. Tulevan tiedon täytyy olla 20 merkkiä pitkä ja sisältää vain kirjaimet A-F, N, numerot 0-9 ja merkin :. Kirjainten ja numeroiden tarkistus tehdään metodilla `isValid()`. Tulevan tiedon ollessa kunnossa kutsutaan `vStore`-olion metodia `StoreValue()`, jolle annetaan parametrina `line`-muttuja. `vStore`-olio on luotu luokasta `StoreValues`. `Run`-metodin sisällä käytetään try/catch rakennetta IOException virheiden varalta.

```
while ((line = br.readLine()) != null)
{
     if (isValid(line) == true && line.length()==20)
     {
          vStore.StoreValue(line);
     }
               .........

public boolean isValid(String input)
     {
          return input.toUpperCase().matches("[0-9|A-F|:|N]+");
     }
```

Listaus 16: Tiedon oikeellisuuden tarkistaminen.

7.4.4 StoreValues luokka

StoreValues-luokka sisältää julkisen metodin `StoreValue()` ja privaatit metodit `Round()` ja `getValue()`. `Storevalue()` ottaa vastaan parametrina merkkijonon, joka sisältää sarjaportista luetun datarivin. Sarjaportista luettu data on muotoa `cn0:1001C04002030000`. Datasta otetaan väliaikaiseen muuttujaan merkit 4-12.

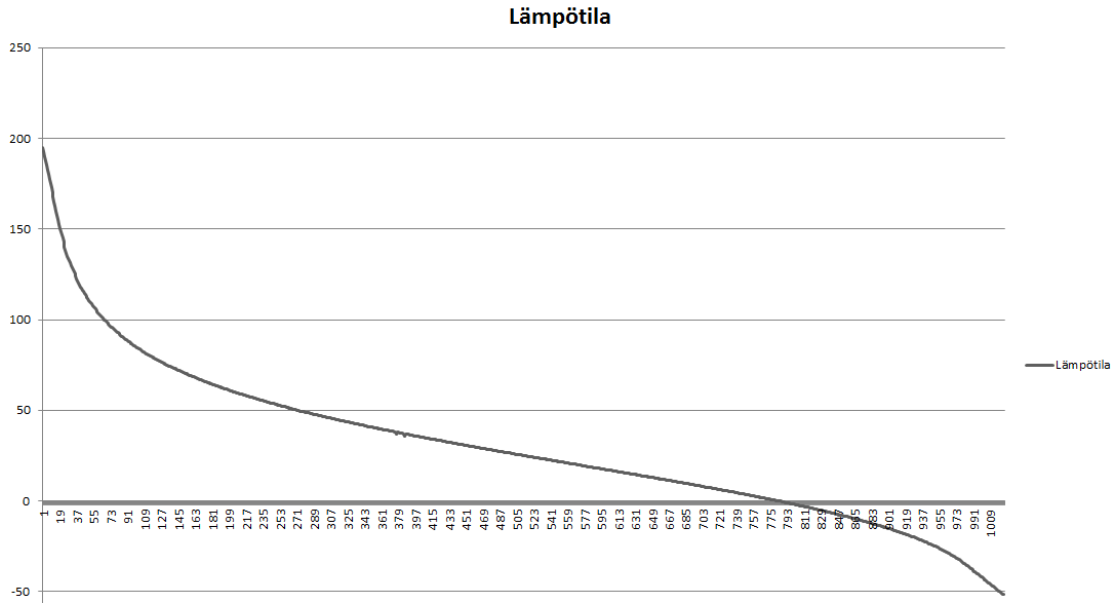
Ensimmäinen merkin järjestysnumero merkkijonoissa on 0. Nämä kahdeksan otettua merkkiä ovat anturin osoite. Osoitetta verrataan ehtolauseilla asetustiedostosta luettuihin. Osoitteen täsmätessä kutsutaan `getValue()` metodia jolle annetaan alkuperäinen data parametrina. Metodin palauttama arvo tallennetaan muuttujaan.

```
tmp = input.substring(0+4, 4+8); // from 4th char to 4+8
char

else if (tmp.equals(this.rst.GT2_outdoor)==true)
{
    this.rVal.SetGT2(getValue(input));
}
```

Listaus 15:

`GetValue()`-metodi palauttaa double-tyyppisen arvon. Metodi muuntaa CAN-datasta saadun arvon lämpötilaksi. Merkit 14-18 muutetaan kokonaisluvuksi. HEX-muotoinen luku voidaan muuttaa kokonaisluvuksi komennolla `Integer.parseInt(hex_luku,16)`, jossa hex_luku on hex-muotoinen luku merkkijonona. Todellinen lämpötila lasketaan polynomikaavalla kokonaisluvusta ja pyöristetään yhteen desimaaliin `round()`-metodilla. Polynomikaavaa tarvitaan, koska lämpötilakäyrä ei ole lineaarinen.



Kuva 5: lämpötilakäyrä

```
val1=(-0.00000000000331726395025513*Math.pow(dval,5));
val2=(0.000000091742268104023200000000*Math.pow(dval,4));
val3=(0.000009630689902772120000000000*Math.pow(dval,3));
val4=(0.004748302142269400000000000000*Math.pow(dval,2));
val5=(1.185334713876020000000000000000*dval);
val6=(167.516938862233);
value = (val1 + val2 - val3 + val4 - val5 + val6);

value=Round(value, 1);
```

Listaus 16: lämpötila-arvon laskukaava

Metodi `Round()` pyöristää annetun desimaaliluvun halutulla tarkkuudella. Metodille annetaan kaksi parametria pilkulla erotettuna. Ensimmäinen parametrin on pyöristettävä desimaaliluku ja toinen parametri desimaalien määrä. Metodi palauttaa desimaaliluvun pyöristettynä haluttuun tarkkuuteen.

```
private static double Round(double luku, int desimaalit)
{
      double p = (double)Math.pow(10,desimaalit);
      luku = luku * p;
      double tmp = Math.round(luku);

      return (double)tmp/p;
}
```

Listaus 17: Pyöristys-metodi.

7.4.5 ConfigFileReader-luokka

Luokka `ConfigFileReader` on asetustiedoston lukemista varten. Asetustiedosto on yksinkertainen tekstitiedosto. Luokka sisältää julkisen metodin ReadConfigFile() ja privaatit metodit `WasConfigOk(), GetIntValue(), GetStrinValue(), GetValueToBoolean() ja isHexaDesimal()`. Asetustiedoston nimi on määritelty merkkijonomuuttujaan `configfile`. Tämä tapahtuu yksinkertaisesti koodilla `String configfile = ”configfile.txt”;`. Tiedostonimessä tulisi olla vain pieniä kirjaimia. Windows-pohjaisissa järjestelmissä kirjainkoolla ei ole väliä, mutta Unix-pohjaisissa järjestelmissä kirjainkoolla on merkitystä. Esimerkiksi config.txt ja Config.txt ovat kaksi eri tiedostoa Unix-pohjaisessa järjestelmässä. Luokassa on privaatti 18 paikkainen boolean-tyyppinen taulukko. Taulukko luodaan `komennolla private boolean[ ]taulukon_nimi = new boolean[alkioiden_määrä];` jossa alkioiden määrä on kokonaisluku. Alkioiden numerointi alkaa nollasta, jolloin 18 paikkaisessa taulukossa viimeisen alkion paikkanumero on 17. Taulukosta on varattu rivi jokaista asetustiedoston riviä varten. Taulukkoon merkitään alkioon true, kun asetustiedostosta on luettu asetus oikein. Tiedostosta lukemiseksi luodaan syötevirtaolio, jolle annetaan parametrina tiedostonimi. Syötevirtaoliosta luodaan puskuroitu syötevirta, jotta tiedostosta voidaan lukea rivi kerrallaan. Puskuroidusta syötevirrasta luodaan olio komennolla `BufferedReader puskuroidun_olion_nimi = new`

`BufferedReader(syötevirtaolio);`. Puskuroidusta syötevirrasta luetaan rivi kerrallaan. Rivi luetan väliaikaiseen merkkijonomuuttujaan kutsumalla puskuroidun syötevirtaolion metodia `readLine()`. Esimerkiksi tmpline = buffReader.readLine();.

```
String file = "file.txt"
FileReader input = new FileReader(file);
BufferedReader bufRead = new BufferedReader(input);
```

Listaus 18: Puskuroidun syötevirtaolion luonti.

Silmukassa luetaan asetustiedostoa rivi kerrallaan. Ennen silmukan ajoa täytyy lukea rivi asetustiedostosta muuttujaan, jotta silmukan koodi ajettaisiin ainakin kerran. Asetustiedoston lukeminen lopetetaan, kun luettu rivi vastaa arvoa null. Tämä tapahtuu asettamalla while-silmukan ehdoksi `muuttuja != null`. Silmukan sisällä on if-else rakenne, joka koostuu useasta else if osasta. Rakenteen ehdoissa etsitään tiettyjä merkkijonoja, esimerkiksi merkkijonoa Port. Tämä tapahtuu kutsumalla tmpline-muuttujan metodia contains. Metodille annetaan etsittävä merkkijono parametrina. Parametrina annettava merkkijonon kirjaimet muunnetaan pieniksi kirjaimiksi, kuten myös luetulle riville on tehty. Ehdossa tarkistetaan myös että kyseistä asetusta vastaavan boolean-taulukon alkion arvo on false. Näin tämä varmistaa että kahta samanlaista asetusriviä ei lueta. Ehtolauseen ehdon toteutuessa kyseisen luettavan asetuksen alkion arvoksi boolean-taulukosta asetetaan true. Asetusolion muuttujaan haetaan arvo kutsumalla tarvittavaa metodia parametrilla joka sisältää asetustiedostosta luetun rivin.

```
if (tmpline.contains(ToLowerCase("Port")) == true && isinconfig[0]==false)
{
     isinconfig[0]=true;
     this.readset.comm_port=GetStringValue(line);
}
```

Listaus 18: Asetustiedosta rivin lukeminen.

Silmukan lopussa luetaan uusi rivi asetustiedosta. Silmukan suorittamisen jälkeen tarkistetaan boolean-taulukko kutsumalla metodia `WasConfigOk`. Metodin palauttama arvo tallennetaan väliaikaiseen muuttujaan jonka arvon `ReadConfigFile` palauttaa.

`ReadConfigFile`-metodin kaikki koodi on try/catch-rakenteen sisällä. Rakennetta käytetään `filenotfound` ja `ioexception` virheiden estämiseen. FileNotFound virhe ilmenee, mikäli avattavaa tiedostoa ei löydy. Tässä tapauksessa käyttäjälle kerrotaan että asetustiedostoa ei löytynyt.

`ToLowerCase`-metodi muuntaa parametrina saadusta merkkijonosta kaikki kirjaimet pieniksi kirjaimiksi. Muunnos tapahtuu kutsumalla merkkijono-muuttujan `metodia toLowerCase`. Saatu merkkijono tallennetaan muuttujaan jonka arvo palautetaan.

```
private String ToLowerCase(String input)
{
     String output="";

     output=input.toLowerCase();

     return output;
}
```

Listaus 19: Merkkikoon muuttaminen pieneksi.

`WasConfigOk`-metodille annetaan parametrina boolean taulukko. Taulukosta otetaan alkioiden määrä kokonaislukumuuttujaan. Tämä tapahtuu komennolla `muuttuja=taulukko.length;`. Taulukon kokoa käytetään metodissa silmukan ehdossa hyväksi. Silmukka käy läpi jokaisen taulukon alkion, joista etsitään arvoa false. Silmukka pyörii niin kauan kuin muuttujan i arvo on taulukon alkioiden määrää pienempi ja output muuttujan arvo on true. Alustettaessa boolean muuttujan output arvoksi on annettu true. Ehtolauseen ehdossa verrataan taulukon alkion arvoa arvoon false. Mikäli arvo on false, suoritetaan ehtolauseen koodi. Ehtolauseen koodissa output saa arvon false. Lopuksi ehtolauseen jälkeen kasvatetaan muuttujan i arvoa yhdellä komennolla i++. Lopuksi metodi palauttaa output muuttujan arvon `komennolla return output;`.

```
while(i<ln && output==true)
     {
          if(input[i]==false)
          {
               output=false;
               System.err.println("Configfile error");
          }
               i++;
     }
```

Listaus 20: Silmukka ja ehtolause.

`GetIntValue`-metodi palauttaa parametrina saadusta merkkijonomuuttujasta kokonaisluku arvon. Metodissa otetaan kokonaislukumuuttujiin talteen parametrina saadun merkkijonoon pituus muuttujaan `len` ja =-merkin paikka muuttujaan `indx`. Merkkijonosta otetaan osa talteen väliaikaisen merkkijono muuttujaan. Talteen otettava osa otetaan väliltä `indx+1` ja `len`. Talteen otetusta osasta korvataan välilyönnit komennolla `tmp.replace(" ","");`. tällä tavoin ei mahdollisesti merkkijonoon jääneet välilyönnit sotke muunnosta kokonaisluvuksi. Merkkijono muutetaan kokonaisluvuksi ja tallennetaan muuttujaan.

Tämä tapahtuu kutsumalla Integer olion metodia `parseInt()` esimerkiksi `kokonaislukumuuttuja=Integer.parseInt(merkkijono);`. Metodissa käytetään try/catch-rakennetta virheiden varalta. Virheen sattuessa näytetään virheilmoitus, jossa näkyy myös metodin saama alkuperäinen rivi. Lopuksi palautetaan saatu kokonaisluku arvo.

```
  len = inp.length();
  indx = inp.indexOf("=");

  tmp = inp.substring(indx + 1, len);
  tmp = tmp.replace(" ", "");

  outp = Integer.parseInt(tmp);
}
catch (Exception e)
{
  System.err.println("NON NUMERIC VALUE IN: " + inp);
}
```

Listaus 21: Metodin rakenne.

`GetStringValue`-metodi toimii pääpiirteittään samalla tavalla `GetIntValue`-metodin kanssa. Metodeiden erona on `GetStringValue`-metodin palauttama merkkijono arvo. Metodissa on myös ylimääräinen tarkistus merkkijonoarvolle. Arvoa verrataan arvoihin ”” ja `null`, jotka molemmat merkitsevät tyhjää merkkijonoa eri tavoin. Merkkijonon täsmätessä jompaankumpaan arvoon näytetään käyttäjälle virheilmoitus, jossa näkyy alkuperäinen asetusrivi ja viesti puuttuvasta arvosta.

```
if (tmp.equals("") == true || tmp.equals(null)==true)
{
     System.out.println("NO VALUE SET IN: " + inp);
}
```

Listaus 22: Virheilmoitus.

`GetValueToBoolean`-metodin tarkoituksena on palauttaa parametrina annetusta asetusrivistä true tai false arvo. Metodissa on boolean-muuttuja `output`, jonka oletusarvoksi on asetettu false. Metodissa kutsutaan `GetIntValue`-metodia, jonka palauttama arvo otetaan talteen väliaikaisen muuttujaan. Väliaikaisen muuttujan arvoa verrataan arvoon 0 ja 1. Arvon ollessa 1 asetetaan output muuttujan arvoksi true ja arvon ollessa 0 jätetään `output`-muuttujaan oletusarvo false. Arvon ollessa jotain muuta kuin 1 tai 0, tulostetaan näytölle virheilmoitus ja käytetään output-muuttujalle oletusarvoa false. Virheilmoituksessa näkyy alkuperäinen asetustiedoston rivi.

`IsHexaDesimal`-metodin tarkoituksena on tarkistaa onko parametrina annettu merkkijono heksadesimaalimuodossa. Tämä voidaan tehdä käyttämällä merkkijonomuuttujan metodia `matches`. `Matches`-metodille annetaan parametrina merkit jotka ovat sallittuja merkkijonolle. Heksadesimaalin tapauksessa sallitut merkit ovat nume-

rot 0-9 ja kirjaimet A-F. Metodi palauttaa false tai true arvon riippuen vertailun tuloksesta.

```
public boolean isHexaDecimal(String input)
{
     return input.toUpperCase().matches("[0-9|A-F]+");
}
```

Listaus 23: Onko merkkijono heksadesimaali muodossa.

7.4.6 ReadedValues-luokka

`RededValues`-luokasta luokka sisältää synkronoidut metodit lämpötilojen tallentamiseen privaatteihin muuttujiin, `GetValues`-metodin kaikkien muuttujien arvojen palauttamiseen yhtenä merkkijonona, metodin lämpötilan pudotuksen tilan muuttamiseen ja tarkistamisen ja `Enable/Disable` -metodit GPIO:n ohjaukseen.
Metodit muuttujien arvojen lukemiseksi ja asettamiseksi on synkronoitu. Metodi muuttujan arvon muuttamisen saa parametrina arvon joka tallennetaan metodissa luokan privaattiin muuttujaan. Metodi muuttajan arvon lukemiseksi palauttaa privaatin muuttujan arvon. Jokaiselle muuttujalle on molemmat metodit. Metodit toisistaan erottaa nimissä olevat `Get` ja `Set`.

```
public synchronized void SetGT1(double _input)
      {
            this.GT1_forwardflow=_input;
      }
      public synchronized double GetGT1()
      {
            return this.GT1_forwardflow;
      }
```

Listaus 24: Set ja Get-metodit

`GetValues`-metodin tarkoituksena on palauttaa kaikki lämpötilat yhtenä merkkijonona. Merkkijonoja voidaan yhdistää yksinkertaisesti +-merkin avulla. Metodissa eri merkkijonojen väliin laitetaan : -merkki, jota käytetään apumerkkinä pilkottaessa merkkijono taulukkoon käyttöliittymän puolella. Merkkijonoon lisätään myös boolean-muutujan arvo, joka muutetaan merkkijonoksi ensin. Muunnos tapahtuu `Boolean.toString(boolean_muuttuja)` komennolla.

`Enable/Disable`-metodit ohjaavat GPIO:n toimintaa. GPIO:ta käsitellään Linux-käyttöjärjestelmässä tavallisena tiedostona johonka voidaan kirjoittaa ja josta voidaan lukea. Lämpötilanpudotusta ohjatessa kirjoitetaan GPIO:n 0 tai 1. Kirjoittamista varten luodaan `filewriter` tulostevirta-olio, jolle annetaan parametrina tiedoston nimi. Filewriter tulostevirtaoliosta luodaan puskuroitu tulostevirta-olio. tiedostoon kirjoitetaan kutsumalla puskuroidun tulostevirran metodia `write()`. Metodilla annetaan parametrina tiedostoon kirjoitettava teksti. Lopuksi tulostevirtaolio suljetaan kutsumal-

la olion `close()`-metodia. Molemmissa metodeissa käytetään try/catch rakennetta estämään mahdollisesti kirjoittamisessa tulevat virheet. Virhetilanteessa annetaan käyttäjälle ilmoitus, jossa kerrotaan ongelman olevan GPIO:ssa.

```
BufferedWriter out = new BufferedWriter(new FileWriter(this.rSet.gpio));
out.write("1");
out.close();
```

Listaus 25: Olioiden luonti ja tiedostoon kirjoittaminen.

7.4.7 ServerForGui-luokka

`ServerForGui`-luokasta luotu olio toimii palvelimena käyttöliittymälle, josta käyttöliittymä kyselee lämpötilat. Palvelinosa on toteutettu siten, että useampi asiakasohjelma voi olla siihen yhtä aikaa yhteydessä. `ServerForGui`-luokka toimii omassa säikeessään, joten se sisältää metodin `run()`. Run-metodissa luodaan `serversocket`-olio. Olio luodaan komennolla `ServerSocket serverSocket = new ServerSocket(port)`, jossa `port` on yhteydessä käytettävän portin numeron sisältävä kokonaislukumuuttuja. Olio luodaan try/catch-rakenteessa, joka näyttää virheilmoituksen portin avaamisen epäonnistumisesta. `Run()`-metodissa while-silmukassa luodaan uusi säikeistetty olio jokaiselle avatulle yhteydelle. Silmukassa olevaa koodia jaetaan niin kauan kuin avattua porttia kuunnellaan. Olio luodaan `GuiMultiServerThread`-luokasta, jolle annetaan parametrit `serversocket.accept()`, `rset` ja `rVal`. Olion luomisen jälkeen kutsutaan sen `start()`-metodia. Silmukan lopussa suljetaan `serverSocket`-olio kutsumalle sen `close()`-metodia.

```
while (listening)
{
     new GuiMultiServerThread(serverSocket.accept(), rset, rVal).start();
}
serverSocket.close();
```

Listaus 28: Säikeistetyn olion luonti jokaiselle uudelle yhteydelle.

7.4.8 GuimultiServerThread-luokka

Luokka on säikeistetty luokka. Luokasssa on olio `rVal`, joka luodaan `readedvalues-luokasta`. Luokassa on `run()`-metodi. Luokassa luodaan syöte ja tulostevirrat. Syöte ja tulostevirtoja käytetään kommunikointiin. `Run()`-metodi sisältää silmukan, jossa luetaan portista tietoa rivi kerrallaan. Silmukassa ehtoina on, että luettava rivi ei ole tyhjä ja boolean muuttujan `ok` arvo on `true`. Dataa luetaan muuttujaan kutsumalla puskuroidun syötevirtaolion metodia `readLine()`. Luettu rivi lähetetään parametrina `gsport`-olion, joka luodaan luokasta `GSProtocol`, metodille `processInput()`. Metodin palauttama arvo otetaan talteen kokonaislukumuuttujaan. Muuttujan arvoa käytetään switch-rakenteessa, joka suorittaa eri koodinpätkiä `proces-`

sInput-metodin palauttaman arvon mukaan. Arvo voi olla väliltä 1-4 tai -1. Arvolla -1 asetetaan ok-muuttujan arvoksi false. Arvolla 1 kutsutaan tulostevirtaolion metodia writeline() jolle annetaan parametrina tekstin merkkijono, joka koostuu tekstistä !STATUS: ja rVal-olion metodin GetStatus() palauttamasta tekstiksi muunnetusta boolean arvosta. Arvolla 2 kutsutaan rVal-olion metodia enable() ja tulostevirtaolion metodia println() parametrilla !ENABLED. Arvolla 3 kutsutan rVal-olion metodia disable ja tulostevirtaolion metodia println() parametrilla !DISABLED. Arvolla 4 kutsutaan tulostevirtaolion metodia println parametrilla, joka sisältää tekstin !VALUES: ja rVal-olion metodin GetValues() palauttaman merkkijonon. Silmukan suorittamisen loputtua suljetaan tuloste- ja syötevirrat kutsumalla näiden metodia close().

7.4.9 GSProtocol-luokka

Luokka saa parametrina merkkijonon. Merkkijono tallennetaan muuttujaan input. Merkkijono kaikki merkit muunnetaan isoiksi kirjaimiksi kutsumalla input-muuttujan metodia toUpperCase(). If-else if-rakenteella etsitään input-muuttujasta merkkijonoja !STATUS, !ENABLE, !DISABLE, !GET_VALUES ja !BYE. Merkkijonon etsiminen tapahtuu rakenteen ehdoista kutsumalla input-muuttujan metodia matches(), joka saa parametrina etsittävän merkkijonon. Rakenne palauttaa kokonaisluku arvon väliltä 1-4 tai arvon -1 riippuen merkkijonosta.

7.4.10 Valueprinter-luokka

Valueprinter-luokan tarkoituksena on tulostaa lämpötila-arvoja näytölle. Luokka toimii omassa säikeessään, jolloin luokka voi hakea tietyin väliajoin lämpötila-arvot ohjelman sisäisistä muuttujista ja tulostaa ne näytölle. Asetustiedostossa voidaan määritellä tulostetaanko arvoja näytölle ja kuinka usein tämä tehdään. Luokka sisältää metodit run(), getDate() ja getTime(). Luokassa on sleeptime-muuttuja johon sijoitetaan asetustiedostosta luettu sekuntiarvo kerrottuna tuhannella. Run()-metodin sisällä on for-silmukka, jonka ehtona on ;;. Silmukan sisältämää koodia ajetaan loputtomasti. Silmukassa tulostetaan näytölle kellonaika, päiväys ja lämpötiloja System.out.println() komennoilla. Jokaisella lämpötila-arvolle on oma tulostuskomento. Tulostuskomennoissa on lainausmerkeissä anturin nimi tekstinä. Anturin nimen jälkeen kutsutaan rVal-olion get metodeja lämpötilojen saamiseksi. Tulostuskomentojen jälkeen on ohjelmakoodi, jolla säie saadaan odottamaan haluttu aika ennen suorituksen jatkamista.

```
System.out.println("GT1_forwardflow " + rVal.GetGT1() + " C");
```

Listaus 29: Näytölle tulostaminen.

```
try
{
      Thread.sleep(sleeptime); // säikeen odotus ...
}
catch (InterruptedException e) // jos tapahtuu virhe odotuksessa
{
      System.err.println("Valueprinter");
      System.err.println(e.toString());
}
```

Listaus 30: Säikeessä odotus.

`getDate` ja `getTime`-metodien tarkoituksena on palauttaa päivämäärä ja kellonaika suomalaisessa muodossa.

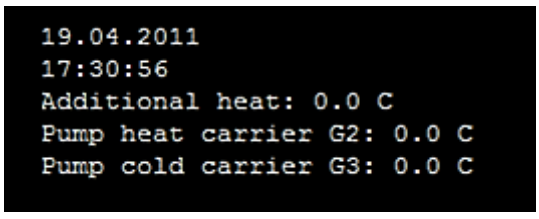
```
private String getDate()
{
      // haetaan päiväys ja laitetaan ulos suomalaisessa muodossa
  DateFormat dateFormat = new SimpleDateFormat("dd.MM.yyyy");
  Date date = new Date();

  return dateFormat.format(date);
}

private String getTime()
{
      // haetaan aika ja palautetaan
      // suomalaisessa muodossa ( 24h kello )
      DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss");
      Date date = new Date();

      return dateFormat.format(date);
}
```

Listaus 31: getDate ja getTime-metodit.



Kuva 6: Näytölle tulostettuja arvoja.

7.5 Asetustiedosto

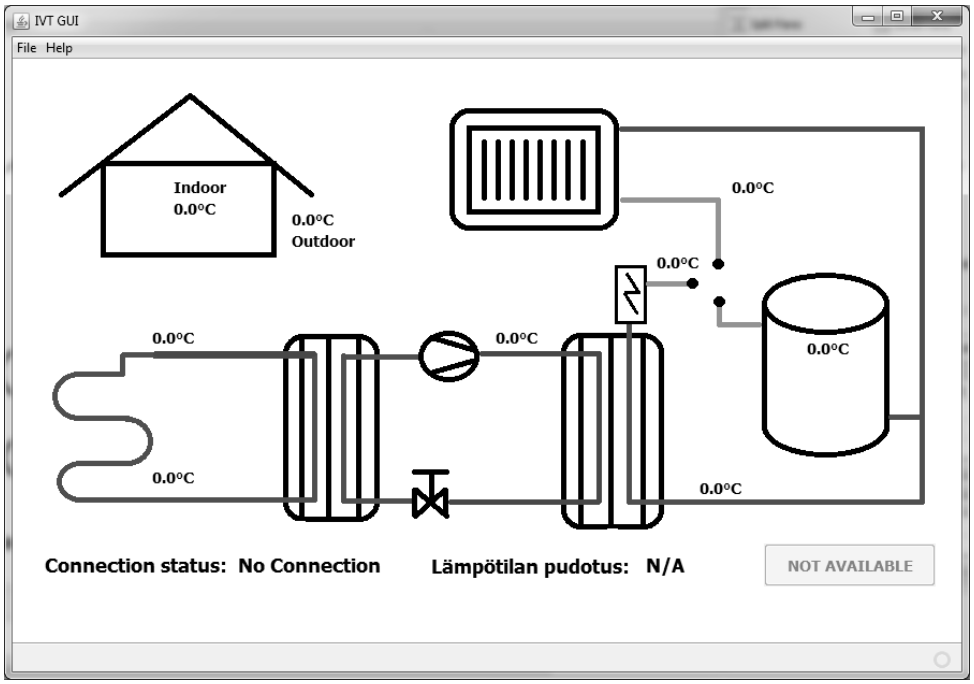
Asetustiedostosta (config.cfg) ohjelmisto lukee perusasetukset esim. käytettävän sarjaportin ja portin nopeuden. Asetustiedostossa määritellään sarjaportista tietoa luettaessa käytetyn puskurin koko, Hex muotoiset osoitteet lämpöpumppuun kytketyille antureille. Asetustiedostosta voidaan määrittää myös lämpötilat tulostettaviksi näytöllä ja viive sekunteina kuinka usein tiedot tulostetaan. Asetustiedostoon voidaan kirjoittaa kommentteja lisäämällä rivin alkuun #- merkki, jolla alkavat rivit ohitetaan ohjelmistossa luettaessa asetustiedostosta. Jokainen asetusrivi on muotoa `<asetuksen nimi> = <arvo>`. Kirjainkoolla on merkitystä ainoastaan asetettaessa arvo `port`-asetukseen unix-pohjaisissa järjestelmissä, joissa portin nimi on muotoa `/dev/ttyS0`.

```
# COMMENT LINE
# PORT SETTINGS
PORT = COM1
PORT_SPEED = 9600
```

Listaus 32: Esimerkki asetuksista asetustiedostossa.

7.6 Käyttöliittymä

Käyttöliittymän toteutuksessa tärkeimpänä lähtökohtana oli toiminnallisuus. Käyttöliittymä itsestään on hyvin yksinkertainen. Ainoat käytetyt käyttöliittymäkomponentit ovat painike ja tekstikenttä (label). Käyttöliittymän toteutukseen käytettiin NetBeans kehitysalustaa, joka mahdollistaa käyttöliittymien rakentamisen helposti. Painiketta käytetään lämpötilanpudotuksen ohjaamiseen. Tekstikenttiä käytetään lämpötilojen näyttämiseen, yhteyden tilan näyttämiseen ja tekstiselitteisiin. Käyttöliittymässä käytetään luokkia selkeyttämään ohjelmakoodin rakennetta.



Kuva 7: Valmis käyttöliittymä.

7.6.1 Toiminta, rakenne ja toteutus

Käyttöliittymässä ohjelmakoodi on kirjoitettu `GUIView`-luokkaan. `GuiView`-luokka on NetBeansin luoma ja se sisältää sovelluskehittimen tuottamaa ohjelmakoodia. Sovelluskehittimen tuottama koodi sisältää esimerkiksi painikkeiden ja tekstikenttien sijoittelun näytöllä, niiden koon ja muut mahdolliset parametrit. `GUIView`-Luokassa luodaan oliot käyttöliittymä komponenteille, lämpötiloille ja verkkoyhteydelle. `GUIView`-luokan `GUIView`-metodin lopussa kutsutaan metodia `StartGuiClient`. `StartGuiClient`-metodin tarkoitus on käynnistää asiakasohjelmakomponentti säie, joka luo verkkoyhteyden Readerin palvelimeen. Kyseisessä metodissa käynnistetään myös säie päivittämään käyttöliittymässä näkyviä lämpötiloja.

```
boolean ok=con.Connect();

if (ok == true)
{

     Runnable runnableGuiClient = new GuiClientThread(con, sVal, gComp);
     Runnable runnableGuiUpdater = new GuiUpdaterThread(con);

     Thread tGui = new Thread(runnableGuiClient);
     Thread tUpd = new Thread(runnableGuiUpdater);


     tGui.start();
     tUpd.start();
```

Listaus 33: Säikeiden luonti ja käynnistys.

Yhteys olion (con) metodi `Connect()` luo verkkoyhteyden. Metodin palauttama arvo tallennetaan boolean muuttujaan ok. Muuttujan ok arvoa verrataan ehtolauseessa arvoon true. Muuttujan ok arvon poiketessa arvosta true, ei verkkoyhteyden luominen onnistunut ja säikeitä ei käynnistetä. Säikeet luodaan luokista `GuiClientThread` ja `GuiUpdaterThread`.
`GUIView` luokassa on tapahtumankäsittelijä painikkeelle. Ilman tapahtumankäsittelijää ei painikkeet toimi. Tapahtumankäsittelijässä kutsutaan `gComp`-olion metodia `GetStatus` jonka arvo tallennetaan boolean muuttujaan `tmp`. Tapahtuman käsittelijässä on ehtolause jossa kutsutaan `con`-olion metodia `Out()`. Parametrina metodille on `!ENABLE` tai `!DISABLE` riippuen lämpötilan pudotuksen tilasta.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{
     boolean tmp=gComp.GetStatus();

     if (tmp==false)
     {
          con.Out("!ENABLE");
     }
     if (tmp==true)
     {
          con.Out("!DISABLE");
     }
}
```

Listaus 34: Tapahtumakäsittelijä.

7.6.2 GuiUpdaterThread

GuiUpdaterThread luokka on yksinkertainen luokka. Luokan muodostimessa luodaan yhteys-oliosta kopio, joka saadaan muodostimelle parametrina.
Luokassa on metodi `run`, joka ajetaan, kun säie käynnistetään. Metodissa on ikuinen silmukka, joka kutsuu `con`-olion `Out`-metodia parametrilla `!GET_VALUES`. Silmukassa kutsutaan `Thread`-olion metodia `sleep()` parametrilla `sleeptime`. Parametri `sleeptime` on int tyyppinen kokonaislukumuuttuja, joka sisältää ajan millisekunteina, jonka säie odottaa. `Sleep`-metodin kanssa käytetään try/catch rakennetta.

```
public void run()
{
    for (;;) // loop that never ends ....
    {
        try
        {
            con.Out("!GET_VALUES");
            Thread.sleep(sleeptime);
        }
        catch (InterruptedException e)
        {
```

Listaus 35: Run-metodi.

7.6.3 Yhteyden luonti

Connection-luokka hoitaa verkkoyhteyden luomisen ja sulkemisen. Luokasta löytyy metodit `Connect`, `Disconnect` ja `Out`. Luokassa kohdekoneen osoite ja portti on määritelty merkkijono muuttujaan `host`. Osoite voidaan antaa esimerkiksi muodossa 192.168.1.1 tai www.osoite.fi. Kohdekone muunnetaan `InetAddres`-tyyppiseen muuttujaan `addr InetAddress`-luokan `getByName`-metodilla, joka saa verkkosoitteen sisältävän merkkijonomuuttujan arvon parametrina. `Socket`-olio luodaan `new socket` komennolla, jolle annetaan parametrina `addr`-muuttuja ja kokonaisluku muuttuja joka sisältää käytettävän portin numeron pilkulla erotettuna. Tiedon lukemiseksi portista luodaan syötevirta `in`. Tiedon lähettämiseksi palvelimille luodaan tulostevirta `out`. Yhteys luodaan `try/catch`-lohkon sisällä. `Catch`-lohkoilla käsitellään virhetilanteet `UnknownHostException, IOException` ja `Exception`. `UnknownHostException` käsittelee virhetilanteet joissa kohdekoneeseen ei saada yhteyttä. `IOException` käsittelee syöte ja tulostusvirtojen luonnissa tapahtuvat virheet. `Exception` käsittelee kaikki muut virheet ja sen `catch`-lohko sijoitetaan viimeiseksi. `Catch`-lohkoissa boolean-muuttujaan `ok` tallennetaan arvo `false`. `Connect`-metodi palauttaa boolean arvon, jonka sisältö riippuu yhteyden luonnin onnistumisesta. Yhteyden luonnin onnistuessa palautetaan `true` ja virhetilanteessa `false`.

```
private InetAddress addr;
try
{
    addr = InetAddress.getByName(host);
    socket = new Socket(this.addr, this.port);
    out = new PrintWriter(kkSocket.getOutputStream(), true);
    in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
```

Listaus 36: Verkkoyhteyden luonti.

Metodin `disconnect` tehtävänä on sulkea syötevirrat ja verkkoyhteys. Tämä tapahtuu yksinkertaisesti kutsumalla `in, out` ja `socket` -olioiden `close()`-metodeja try/catch rakenteen sisällä.

7.6.4 Kommunikointi palvelimen kanssa

Kommunikointi palvelimen kanssa tapahtuu `GuiClientThread` luokan avulla. Kyseinen luokka ajetaan omana säikeenään ja se sisältää run()-metodin. Luokka saa parametrina `GuiComponents`, `Connection` ja `StoreValues` oliot. `Run` -metodissa on silmukka jonka sisällä olevaa koodia ajetaan niin pitkään kun silmukan ehdot ovat voimassa. Koodin ajamisen ehtoina on, että palvelimelta tulee dataa ja `ok`-muuttuja on tilassa `true`. Palvelimelta tulevaa dataa luetaan teksti-muuttujaan `fromServer` `con`-olion, joka on paikallinen kopio parametrina tulleesta `connection` luokasta, metodilla `readLine`(). Silmukan ehdoissa `fromServer`-muuttujan arvoa verrataan `null`-arvoon pois sulkevasti. Null-arvo merkitsee täysin tyhjää (alustamatona). Tyhjä muuttujan sisältö merkitsee, että palvelimelta ei tule dataa tai yhteys on katkennut. Kaikki muuttujassa `fromServer` olevan tekstin kirjaimet muutetaan isoiksi kirjaimiksi kutsumalla `fromServer` muuttujan metodia `toUpperCase()`.

```
while (((fromServer = this.con.in.readLine()) != null) && (this.running == true))
{
     System.out.println("StartWhile");
     fromServer = fromServer.toUpperCase();

     ............

     else if (fromServer.contains("!VALUES")==true)
     {
          String tmp=fromServer.replace("!VALUES:", "");
          sVal.UpdateStoredValues(tmp);
     }
```

Listaus 37: silmukka ja kommunikointi.

Muuttujan `fromServer` arvoa verrataan myös if/else rakenteessa. If/else rakenteella voidaan suorittaa erilaisia tehtäviä palvelimelta tulevien viestien mukaan. If/else rakenteessa tutkitaan esimerkiksi, että sisältääkö palvelimelta tuleva tieto arvoa `!VALUES`. Arvon vastatessa tutkittavaa arvoa voidaan kutsua eri olioiden metodeja tehtävien suorittamiseksi. `!VALUES` arvolla kutsutan `sVal`-olion metodia `UpdateStoredValues()` parametrilla `tmp`. Muuttujaan `tmp` tallennetaan `fromServer`-muuttujan sisältö josta otetaan pois `!VALUES` osa kutsumalla `fromServer`-muutujan metodia `replace`(). `Replace()`-metodi saa kaksi parametria joista ensimmäinen on korvattava merkkijono ja toinen merkijono-parametri jolla korvattava teksti korvataan. Palvelimelta `!BYEZ` viestin tullessa muutetaan boolean muuttujan `ok` arvoksi `false` jolloin silmukkaa ei enää ajeta. Silmukka ja ehtolauseet ovat sama try/catch rakenteen sisällä.

7.6.5 StoreValues –luokka

`StoreValues`-luokka toimii siltana `GuiclientThread` ja `GuiComponents` luokkien välillä. Luokka saa parametrina `GuiComponents`-olion. Luokassa on yksi metodi `UpdateStoredValues` joka saa parametrina merkkijonon. Merkkijono tallennetaan merkkijonomuuttujaan `in`.

```
String[] tmp;
tmp = in.split(":");
gComp.UpdateGuiComponents(tmp);
```

Listaus 38: Merkkijonon muuntaminen taulukoksi.

Metodissa luodaan taulukko `tmp`-komennolla `String[ ] tmp`. Taulukosta tulee merkkijono taulukko. Hakasulkeet muuttujan esittelyssä asettavat muuttujan taulukoksi. Merkkijono in pilkotaan taulukoksi merkkijonon metodilla `split()` joka saa parametrina merkin `:`, jota käytetään pilkkomisessa. Saatu taulukko tallentuu taulukkomuuttujaan `tmp`. Saatua taulukkoa käytetään kutsuttaessa `gComp`-olion metodia `UpdateGuicomponents()`.

7.6.6 GuiComponents-luokka

Luokan tarkoituksena on huolehtia muuttuvien tekstikenttien ja painikkeen päivittämisen näytöllä. Luokka saa parametrina ohjelmassa käytetyt tekstikentät ja painikkeen. Luokka sisältää metodit `UpdateGuiComponents()`, `Connected()`, `NotConnected()`, `GetStatus()` ja `Pudotus`. Luokassa on merkkijono muuttuja `deg`, joka sisältää astemerkin ja merkin C. Astemerkki saadaan koodilla `"\u00b0"`.
Metodi `UpdateguiComponents()` saa parametrina taulukon, joka sisältää lämpötila-arvot. Taulukosta arvot sijoitetaan oikeisiin tekstikenttiin kutsumalla tekstikentän metodia `setText()`. Metodi saa parametrina taulukon in ja muuttujan `deg`, jossa hakasuluissa on taulukon lopussa indeksinumero josta arvo saadaan. Plus-merkillä muuttujien välissä muodostetaan kahdesta merkkijonosta yksi.

```
private String deg="\u00b0" + "C";
     ........
this.lbl_GT2_outdoor.setText(in[1]+deg);
```

Listaus 39: Tekstin sijoittaminen tekstikenttään taulukkomuuttujan solusta.

`Connected` ja `NotConnected` metodit päivittävät ohjelman tilatietotekstikentän tekstin ohjelman tilan mukaan. Yhteyden ollessa kunnossa lukee kentässä *Connected*, muulloin lukee *No connection*.

Metodi Pudotus huolehtii painikkeen tekstistä. Lämpötilanpudotuksen ollessa päällä lukee painikkeessa *Turn Off* ja pudotuksen ollessa pois päältä lukee painikkeessa *Turn On*.

`GetStatus`-metodi palauttaa boolean arvon *true* tai *false* sen mukaan että onko lämpötilan pudotus päällä.

8 PÄÄTÄNTÄ

EU:n tiukentuvat päästömääräykset ja sähköenergian hinnan nousu pakottavat ihmiset etsimään vaihtoehtoisia lämmitysjärjestelmiä. Yksi vaihtoehtoisista lämmitysjärjestelmistä on maalämpöpumppu. Ohjaus ja analysointiohjelmistoille on tulevaisuudessakin kehittämistarvetta. Myös lämpöpumppujen valmistajien tulisi herätä ja panostaa myös ohjelmistokehitykseen. Moni lämpöpumpun ostaja olisi lähes varmasti valmis jopa maksamaan ohjelmistosta.

Statlink on ainut tiedossani oleva IVT:n lämpöpumpuille kehitetty ohjelmisto. Ohjelmisto ei tue uusimpia pumppumalleja tietojeni mukaan joten opinnäytetyön kohteena olleelle ohjelmistolle oli tarvetta.

Työ oli omalta osalta haastavin ja mielenkiintoisin projekti koko opiskeluajalta. Koulussa saamani ohjelmoinnin opetus kattoi suurimman osan opinnäytetyössä tarvittavista tiedoista ja taidoista. Ohjelmiston kehittämisvaiheen aikana google:n hakua käyttäen löytyi materiaalia mm. sarjaportin kautta kommunikointiin Java:lla ja vinkkejä vastaan tulleiden pienten ohjelmointiongelmien ratkaisuun. Ohjelmointiongelmia, joihin itse en olisi löytänyt ratkaisua, ei tullut vastaan projektin aikana. Ohjelmistosta löytyi myös testausvaiheessa muutamia pikku bugeja, jotka sain korjattua suhteellisen nopeasti. Valmiissa ohjelmistossa ainoastaan lämpötilanpudotus ei saatu toimimaan. Omissa testauksissa maalämpöpumpun yhteyteen liitettyä PC:tä vastaavalla laitteella ja viimeisimmällä versiolla ohjelmistosta en pystynyt toistamaan tai havaitsemaan kyseistä ongelmaa lukuisista yrityksistä huolimatta, joten päädyimme Huhtisen Jounin kanssa kahteen eri vaihtoehtoon ongelman aiheuttajasta. Ensimmäisenä vaihtoehtona päädyimme työnantajan kehitysversion ja oman version koodissa olevaan pieneen eroon ja toisena vaihtoehtona gsm-modeemiin aiheuttamaan suureen viiveeseen tiedonsiirrossa. Voitaneen sanoa, että ohjelmiston kehitys onnistui tavoitteiden mukaisesti omalta osaltani lämpötilan pudotuksessa ilmennyttä mahdollista ongelmaa lukuun ottamatta.

Ohjelmistoa voisi vielä kehittää lisäämällä mahdollisuuden tallentaa lämpötila-arvoja tietokantaan ja mobiililaitteille yksinkertaisempi käyttöliittymä. Tietokantaan tallennetuista arvoista voitaisiin nähdä esimerkiksi pidemmältä aikaväliltä ulko- ja sisälämpötilojen vaihteluita. Mobiilikäyttöliittymä mahdollistaisi lämpöpumpun toiminnan tarkkailun ja ohjauksen pieneltä näytöltä.

LÄHTEET

Barret Daniel J., Silverman Richard 2001, SSH, The Secure Shell – The Definative Guide

Burd Barry, 2005, Beginning Programming with Java For Dummies , 2nd Edition

Deitel H.M., 2006, Java™, How to Program, Seventh Edition

Eclipse [viitattu 15.3.2011] saatavissa: http://netbeans.org/

Itä-Suomen yliopiston verkkosivut [viitattu13.3.2011] saatavissa: http://www.uef.fi/

IVT 2000, Käyttöohje Premiumline X11-X15 1.0 fi

Kinnunen, Jukka 2000, JAVA –OHJELMOINTIKIELI, Pohjois-Savon Ammattikorkea-koulu

Kinnunen, Jukka, Käytettävyys ja käyttöliittymän suunnittelu, Savonia ammattikor-keakoulu

NetBeans [viitattu 15.3.2011] saatavissa: http://netbeans.org/

Oracle [viitattu 16.3.2011] saatavissa: http://www.oracle.com/

PC Engines [viitattu 14.3.2011] saatavissa: http://pcengines.ch/

Putty [viitattu 14.3.2011] saatavissa: http://www.chiark.greenend.org.uk/

RXTX Official wiki [viitattu: 16.4.2011] saatavissa:
http://rxtx.qbang.org/wiki/index.php/FAQ

Senera:n verkkosivut [viitattu 2.3.2011] saatavissa:
http://www.senera.fi/Maalampopumppu

SULPU Suomen Lämpöpumppuyhdistys ry [viitattu 2.3.2011] saatavissa:
http://www.sulpu.fi

WinSCP [viitattu 14.3.2011] saatavissa: http://winscp.net/eng/index.php

www.savonia.fi

Savonia
ammattikorkeakoulu