I T - D I P L O M U D D A N N E L S E N

OPERATIVSYSTEMER OG PROCESINTERAKTION

EKSAMENSPROJEKT
AF
JAN SCHRØDER HANSEN

FORÅR 2010

1. INDLEDNING

Dette eksamensprojekt er lavet i forbindelse med faget Operativsystemer og Procesinteraktion på IT- Diplomuddannelsen, Ingeniørhøjskolen i København.

Faget har taget udgangspunkt i bogen Operating Systems, med undertitlen ”Internals and Desing Principles” af forfatteren William Stallings, samt ekstra materiale fra underviser Bo Holst-Christensen.

2. OPGAVEBESKRIVELSE

For at komme igennem noget af materialet i faget, specielt kapitel 5 (Concurrency: Mutual Exclusion and Synchronization) og til dels kapitel 6 (Concurrency: Deadlock and Starvation), har jeg valgt at lave et rengøringsrobotsimuleringsprogram. Som visuelt viser hvordan robotter rengører et areal, som er opdelt i felter.

Der skal være 3 robotter, som skal holde arealet rent. Hver robot får sin egen tråd. Arealet der skal rengøres opdeles i 10 gange 10 felter. Hvert felt kan kun have en robot stående af gangen. Et af felterne vil være en skraldespand.

Der skal være en tilfældighedsgenerator, som med mellemrum genererer noget snavs på felterne. Denne snavsgenerator får også sin egen tråd.

Robotterne skal søge efter snavs uden at støde ind i hinanden. Hver robot kan rengøre 5 snavsede felter, hvorefter den må en tur til skraldespanden, for at blive tømt for snavs. Når en robot skal beslutte sit næste træk, kan den se alle felter omkring sig, dvs. 8 felter, hvis robotten ikke står ved en kant. Hvis en eller flere af disse 8 felter er snavset, vælges et tilfældigt snavset felt af disse. Ellers vælges der et tilfældigt rent felt. Hver robot husker også de sidste 6 felter den har besøgt, disse felter undgås når der skal vælges nyt felt. Med mindre at dette ”låser” robotten. Robotten kan komme til at låse sig selv inde i et hjørne vha. denne ”felthukommelse”, hvis dette sker så nulstilles denne ”felthukommelse”.

Alle robotter og snavsgeneratoren, som alle arbejder i hver sin tråd, får også en log, så der kan følges med i hvilke skridt, de enkelte tråde gennemgår. Der vil også være en log for selve skraldespanden, så man kan se, hvor meget snavs der er blevet modtaget fra de enkelte robotter.

Centralt i programmet vil være det areal der skal rengøres, kaldet board i programmet. Dette board benyttes til håndtering af, hvilke felter der er rene og snavset, hvor skraldespanden og de enkelte robotter står. Dvs. at boardet har overblikket, og det er her synkroniseringen, mellem de enkelte tråde foregår. Dette betyder også, at det ikke er et rigtigt simuleringsprogram, da de enkelte robotter ikke er autonome, men hele tiden ”spørger” boardet.

Der er heller ikke deadlock problemer i dette program, da en robot kun kan låse et felt af gangen. En deadlock situation kræver som regel at to eller flere ressourcer, låses i forskellige rækkefølge af to eller flere tråde/processer.

Selve programmet laves i sproget Java[1], og gør brug af Java’s muligheder inde for trådprogrammering.

[1] Java, se www.java.com

2.1. AFGRÆNSING

For at afgrænse opgaven har jeg valgt kun at benytte 3 robotter. For at undgå for mange konflikter, f.eks. hvis skraldespanden står i et hjørne, så kan en robot spærres inde ved, at der står 3 andre fyldte robotter, og venter på at komme til skraldespanden. Dette kan løses ved at definere nogle felter som kø til skraldespanden, og et andet felt som udgang fra skraldespanden.

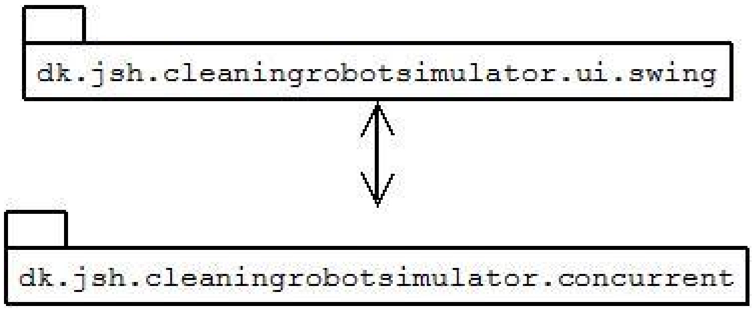
Løsningen er heller ikke dækket ind med hensyn til unit tests.

3. DESIGN

Følgende er en kort beskrivelse af designet af applikationen. Startende med et pakkediagram efterfulgt af et designklassediagram.

3.1. INDELING AF KODE I LAG

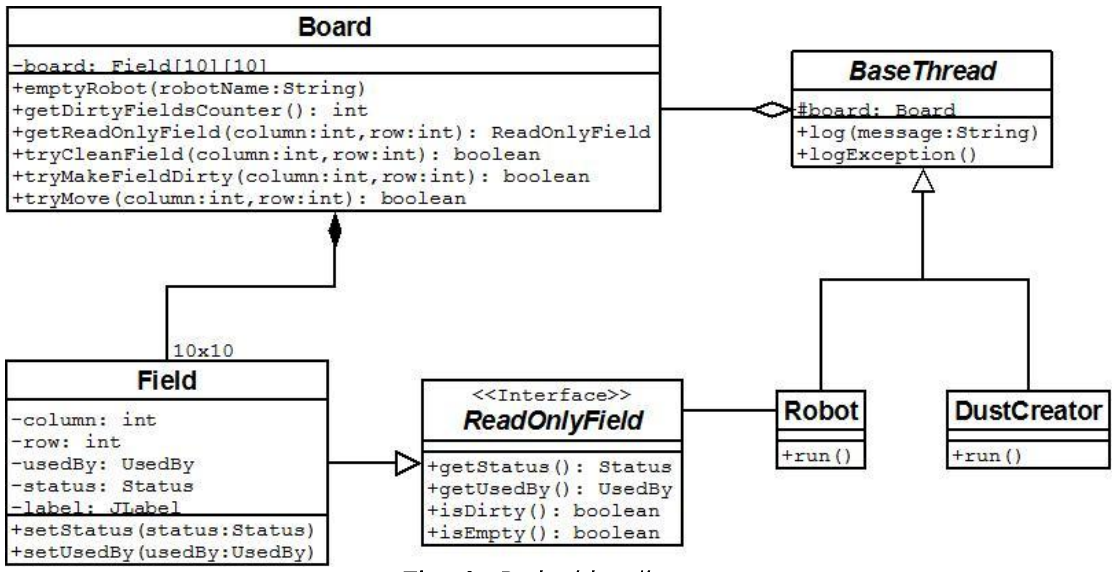
Jeg har inddelt koden i 2 overordnet lag vha. java pakker, se følgende diagram.


dk.jsh.cleaningrobotsimulator.ui.swing
dk.jsh.cleaningrobotsimulator.concurrent

Figur 1- Pakkediagram

Pakke	Beskrivelse
dk.jsh.cleaningrobotsimulator.ui.swing	Kode til håndtering af brugerfladen.
dk.jsh.cleaningrobotsimulator.concurrent	Kode til håndtering af tråd-programmering/samtidighed.

Koden til håndtering af brugerfladen beskrives ikke yderligere, da det ikke er relevant for opgaven. Det følgende er et klassediagram over de væsentligste klasser, som er i dk.jsh.claeningrobotsimulator.concurrent pakken.



Figur 2 - Design klassediagram

Robot og DustCreator arver begge fra den abstrakte klasse BaseThread, som igen arver fra Thread (standard del af Java). Dvs. at objekter af klasserne Robot og DustCreator, kan startes som tråde. I applikationen startes der 3 Robot tråde og en DustCreator tråd. BaseThread indeholder også et Board, som benyttes af Robot og DustCreator trådene.

Boardet består af 10 gange 10 Field's. Det er i selve Board klassen alt synkronisering sker, idet metoderne emptyRobot, tryCleanField, tryMakeFieldDirty og tryMove alle er synkroniseret.

Board metoden getReadOnlyField benyttes af Robot til at undersøge felter omkring sig. Denne metode er ikke synkroniseret, det er derfor at den i stedet for at returnere Field's, så returnerer den ReadOnlyField's. Dvs. at når en robot tråd er ved at undersøge hvilke mulige felter, den kan rykke hen på eller rengøre, så låser denne proces ikke for, at de andre robotter kan rykke eller rengøre. ReadOnlyField har heller ikke nogen set metoder, det er for at sikre at en Robot tråd ikke ændre et Fields status. Det er kun Board objektet der har lov til det.

Dvs. at en Robot tråd ikke kan være sikker på den status, som står i et ReadOnlyField. Status kan være ændret af en anden Robot. Det er også derfor de 3 Board metoder tryCleanField, tryMakeFieldDirty og tryMove alle returnerer en boolean, som er true hvis operationen lykkes eller false hvis ikke.

F.eks. en robot henter alle 8 felter omkring sig (som ReadOnlyField's), undersøger disse, finder et felt som skal rengøres, og ikke er optaget af en anden robot, så kalder den først tryMove() metoden, som f.eks. returnerer false, fordi feltet er overtaget af en anden robot i mellemtiden.

4. TRÅDET PROGRAMMERING I JAVA

En tråd i Java kan laves ved at arve fra java klassen Thread, eller ved at implementere interfacet Runable. Begge kræver at der er en run metode, som er den metode som indeholder den kode, der skal afvikles i en selvstændig tråd. Som det fremgår af Figur 2 - Design klassediagram, har klasserne Robot og DustCreator begge en run metode. I programmet startes der tråde af disse klasser (se bilag View.java). Robot tråde startes her ved at lave et nyt objekt af klassen Robot og kalde start metoden på denne, som så starter selve tråden.

DustCreator objektet startes vha. af en scheduler (se bilag View.java). Her bruges en scheduler, med en fast pause mellem hvert run. I programmet bliver DustCreator startet med 30 sekunders intervaller.

For at undgå konflikter robotterne imellem, benytter jeg, java keyworded synchronized, dette benyttes kun af klassen Board. Enten på hele metodekaldet, eller inde i nogle blokke i metoderne. Synchronize er en mutual exclusion også kaldet critical section. Dvs. at kun en tråd kan eksekvere kode her af gangen. I java kan hele motoder være en critical section. Men det er faktisk en kort notation for synchronize(this). Se følgende:

```
public boolean tryMove(..) {
        synchronize(this) {
                //Så længe en tråd er i denne kritiske sektion, kan andre tråde ikke benytte kode i dette
                //objekt, som også synkroniserer på this.
        }
}
```

Det samme kunne opnås ved at skrive public synchronized boolean tryMove(..){..}
”this” er i dette tilfælde er objektet selv. Man kan også synkronisere på andre objekter. Grunden til at jeg ikke bruger synchronized på alle metoder i Board, skyldes at noget af koden i de enkelte metoder, godt kan køres på samme tid i forskellige tråde. F.eks. i tryMove metoden valideres de argumenter, som metoden kaldes med, før metoden laver en critical section vha. synchronize.

Board klassen, betragter jeg som en monitor, da dens opbygning minder om det, som er beskrevet i bogen Operation System i kapitel 5 afsnit 5.4 Monitors. Fields og andre attributter tilgås kun gennem Board klassens metoder. synchronize(this) søger for at kun en proces (her tråd) af gangen, kan arbejde med Board klasses interne attributter. I bogen Java Concurrency in practice[2], kaldes dette også for ”Java monitor pattern”.

Udover det jeg har benyttet i programmet, så tilbyder java mange andre muligheder i forbindelse med trådprogrammering. F.eks. Atomic Types, en AtomicInteger sikre at kun en tråd af gangen, kan udfører f.eks. en addition. Derudover er der forskellige typer af semaphore, der kan benyttes som låsemekanismer. Dvs. at man kan lave noget programlogik, hvor en tråd ”låser” en semaphor, andre tråde må vente på at denne tråd låser semaphoren op igen, før de kan komme til. Problemet med semaphore er, at de kan blive spredt ud over mange programfiler, og det kan derfor være svært, at vedligeholde og fejlfinde.

[2] Java Concurrency in Practice, af Brian Goetz m.fl., se www.javaconcurrencyinpractice.com

Under programmeringen har jeg gjort brug af følgende værktøjer og andre ressourcer:

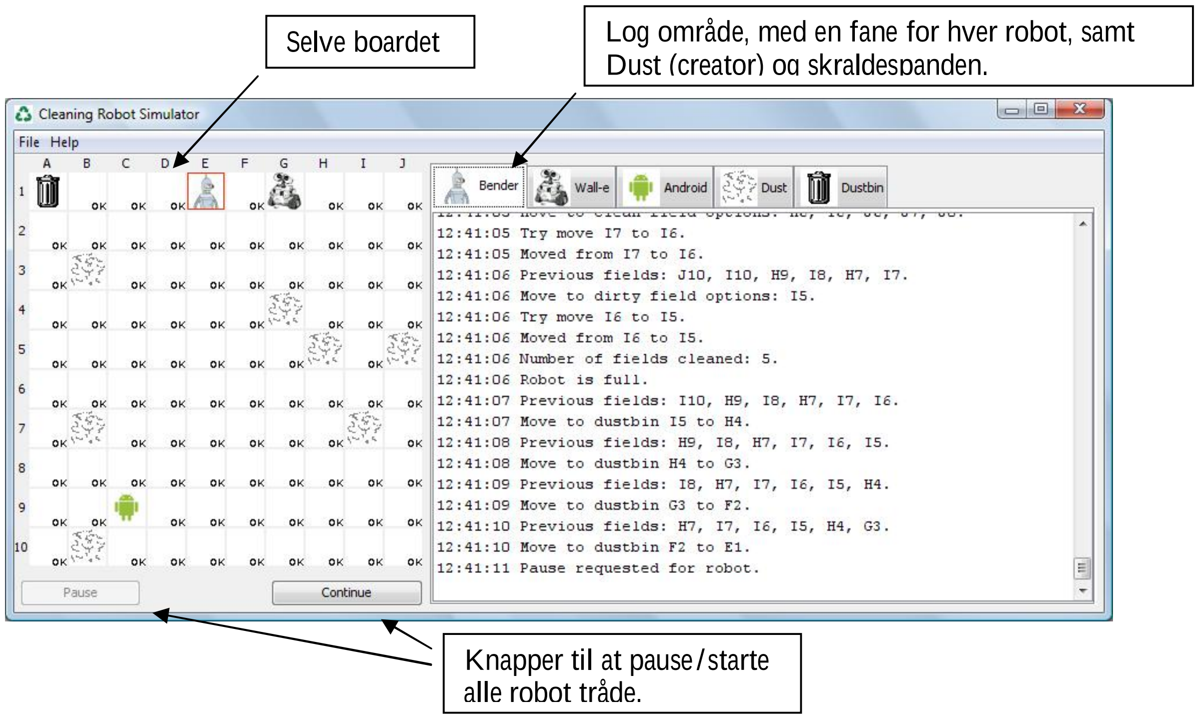
- NetBeans[3] – Java IDE, Java udviklingsmiljø.
- Maven2[4] - Build, deploy, dependency management tool
- SubVersion[5] via GoogleCode - Repository til al kode. Bl.a for at have backup af koden på en anden maskine, og for at få historik på mine koderettelser. Goolge tilbyder at stille lagerplads til rådighed via deres GoogleCode[6]. Mod at man frigiver koden som open source[7]. Hele dette projekt inkl. denne tekst kan også findes under GoogleCode, se code.google.com/p/cleaning-robot-simulator/.

Alle diagrammer er lavet vha. Dia[8], som har skabeloner til UML diagrammer. De enkelte diagrammer ligger også på den vedlagte CD. Se bilag for indholdet på den vedlagte CD.

[3] NetBeans IDE, se netbeans.org
[4] Maven2, se maven.apache.org/
[5] SubVersion, se subversion.tigris.org
[6] GoogleCode, se code.google.com
[7] Open source, se da.wikipedia.org/wiki/Open_source
[8] Dia, se projects.gnome.org/dia

Programmet startes ved at finde programmet på den vedlagte cd, programmet hedder "cleaning-robot-simulator-1.0.jar" og ligger i kataloget Program. I Windows startes programmet ved at dobbeltklikke på filen i Stifinder (File Explore). Alternativ kan programmet startes ved at skrive "java -jar cleaning-robot-simulator-1.0.jar" fra en kommandolinje. Under alle omstændigheder kræver programmet, at der er installeret en Java 6 runtime[9], på den pågældende maskine.

Når programmet startes, fremkommer følgende dialog:


Selve boardet
Log område, med en fane for hver robot, samt Dust (creator) og skraldespanden.
Cleaning Robot Simulator
File Help
Bender
Wall-e
Android
Dust
Dustbin
Pause
Continue
Knapper til at pause/starte alle robot tråde.

```
12:41:05 Try move I7 to I6.
12:41:05 Moved from I7 to I6.
12:41:06 Previous fields: J10, I10, H9, I8, H7, I7.
12:41:06 Move to dirty field options: I5.
12:41:06 Try move I6 to I5.
12:41:06 Moved from I6 to I5.
12:41:06 Number of fields cleaned: 5.
12:41:06 Robot is full.
12:41:07 Previous fields: I10, H9, I8, H7, I7, I6.
12:41:07 Move to dustbin I5 to H4.
12:41:08 Previous fields: H9, I8, H7, I7, I6, I5.
12:41:08 Move to dustbin H4 to G3.
12:41:09 Previous fields: I8, H7, I7, I6, I5, H4.
12:41:09 Move to dustbin G3 to F2.
12:41:10 Previous fields: H7, I7, I6, I5, H4, G3.
12:41:10 Move to dustbin F2 to E1.
12:41:11 Pause requested for robot.
```

Figur 3 - Selve programmet

[9] Java runtime kan hentes fra java.sun.com

Symboler på boardet:

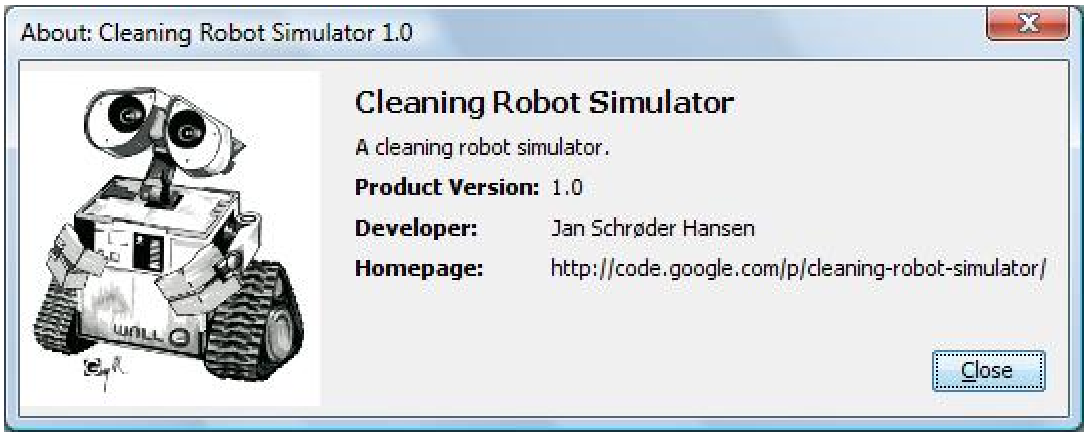
Robotter	Robotterne får en rød kant når de er fyldte. Eksempel på en fyldt robot.
Felter	Rent felt OK, snavset felt
Skraldespand	Skraldespand som ikke er i brug. Skraldespand som benyttes af en robot.

De to knapper "Pause" og "Continue" kan benyttes til at stoppe/genstarte alle Robotterne, så man evt. kan nærlæse de enkelte logfaneblade.

Hver robot har et logfaneblad, hvor robottens handlinger kan aflæses. Hvilke felter den har besøgt, hvilke snavsede felter der er omkring den? En robot forsøger altid at gå til et snavset felt, hvis der er sådan et i dens omkreds. Hvis der ingen snavset felter er, vælges der et tilfældigt rent felt, den ikke har besøgt inden for 6 træk. Når en robot har rengjort 5 felter, er den fyldt, og går til skaldepanden. De enkelte felter navngives med A til J for kolonner og med 1 til 10 for rækker. F.eks. "A1" er der hvor skraldespanden står.

På "Dust" logfanen kan man se hvilke felter der bliver gjort snavset. Og på "Dustbin" logfanen fremgår det hvilke robotter, som har tømt sit snavs over i skraldespanden, og hvor meget snavs der totalt er modtaget.

Menuen i programmer indeholder "File -> Exit" som afslutter programmet. Og "Help -> About…" viser følgende dialog, som fortæller lidt om programmet:


About: Cleaning Robot Simulator 1.0

Cleaning Robot Simulator

A cleaning robot simulator.

Product Version: 1.0

Developer: Jan Schrøder Hansen

Homepage: http://code.google.com/p/cleaning-robot-simulator/

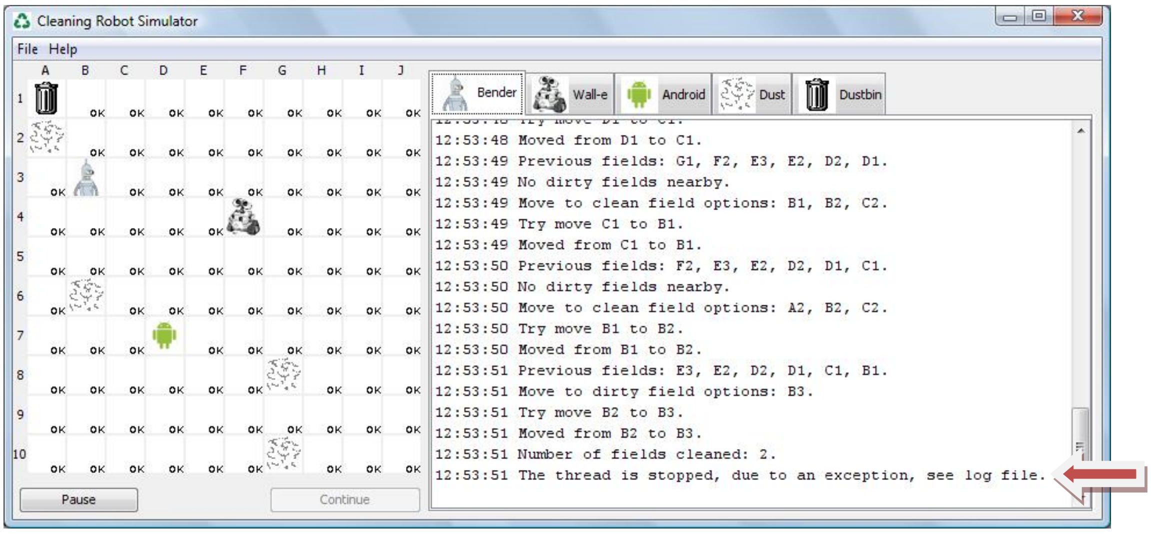
Figur 4 - About dialog

Robotternes navne og ikoner er lånt fra:

- Android styresystemet fra Google, se www.android.com
- Wall-E, Disney animationsfilm fra 2008, se en.wikipedia.org/wiki/WALL-E
- Bender, tegneserien Futurama af Matt Groening (Også kendt for serien The Simpsons), se da.wikipedia.org/wiki/Futurama

6.1. FEJLHÅNDTERING

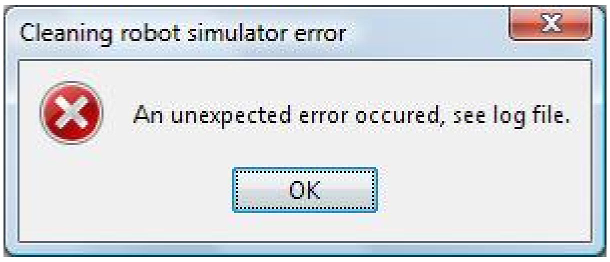
Hvis der opstår en fejl i en af Robotterne, vil det fremgå af den pågældende robots logfane. Se den røde pil på følgende eksempel.



```
12:53:48 Moved from D1 to C1.
12:53:49 Previous fields: G1, F2, E3, E2, D2, D1.
12:53:49 No dirty fields nearby.
12:53:49 Move to clean field options: B1, B2, C2.
12:53:49 Try move C1 to B1.
12:53:49 Moved from C1 to B1.
12:53:50 Previous fields: F2, E3, E2, D2, D1, C1.
12:53:50 No dirty fields nearby.
12:53:50 Move to clean field options: A2, B2, C2.
12:53:50 Try move B1 to B2.
12:53:50 Moved from B1 to B2.
12:53:51 Previous fields: E3, E2, D2, D1, C1, B1.
12:53:51 Move to dirty field options: B3.
12:53:51 Try move B2 to B3.
12:53:51 Moved from B2 to B3.
12:53:51 Number of fields cleaned: 2.
12:53:51 The thread is stopped, due to an exception, see log file.
```

Figur 5 - Excempel på en exception i en tråd

Hvis der sker en fejl i selve applikationen, fremkommer følgende dialog.


Cleaning robot simulator error

An unexpected error occured, see log file.

Figur 6- Applikationsfejl

Når det trykkes på "OK" knappen, lukkes programmet.

I begge ovenstående fejlsituationer, henvises til en logfil. Her kan en java stacktrace af selve fejlen ses. På en Windows maskine vil denne fil typisk ligge følgende sted:

C:\Users\<User>\AppData\Local\Temp\cleaning-robot-simulator.log.0

7. KONKLUSION

Formålet med opgaven var at lave en rengøringsrobotsimulator i java. Som visuelt viser hvordan 3 robotter holder et areal, bestående af 10 gange 10 felter rent. Jeg har måske brugt en del tid på den visuelle side af sagen, mest fordi det er sjovt. Men det viser ganske tydeligt hvordan de enkelte robotter, i hver sin tråd interagerer med arealet (Board'et).

Som nævnt tidligere er det ikke et ægte simuleringsprogram, fordi de enkelte robotter ikke er autonome. Det er Board'et som bestemmer, om en robot må flytte til et felt eller ej. Men da opgaven også gik ud på at komme noget af materialet fra bogen igennem, specielt kapitel 5 (Concurrency: Mutual Exclusion and Synchronization) og til dels kapitel 6 (Concurrency: Deadlock and Starvation), synes jeg at det er lykkes meget godt. Jeg har været inde på trådprogrammering, samtidighed, mutual exclusion, semaphore og monitors.

Jeg er ikke løbet ind i de store problemer med hensyn til deadlock og starvation i programmet. Starvation er håndteret af mit design, hvor det er Board, som hele tiden begrænser de enkelte robotter, så de ikke løber løbsk. Og deadlock undgås ved, at der kun er en robot, som kan står på et felt af gangen. Hvis der var mere end 3 robotter, kunne en robot eksempelvis blive låst inde i et hjørne. Det betragter jeg ikke som en deadlock situation, da denne robot så bare må vente til de andre robotter har flyttet sig. Som det fremgår af bogen, er designet vigtigt når man udvikler programmer med flere tråde. Det der i bogen kaldes "Deadlock prevention".

8. BILAG

8.1. KODE

Følgende er javakoden, som er udviklet i forbindelse med denne opgave. Startende med koden i javapakken: dk.jsh.cleaningrobotsimulator.concurrent, derefter kommer pakken dk.jsh.cleaningrobot.ui.swing.

dk.jsh.cleaningrobotsimulator.concurent

\dk\jsh\cleaningrobotsimulator\concurrent\Constants.java

```java
package dk.jsh.cleaningrobotsimulator.concurrent;

import java.text.SimpleDateFormat;

/**
 * Cleaning robot simulator constants.
 * @author Jan S. Hansen
 */
public class Constants {

    public final static int MAX_ROWS = 10;
    public final static int MAX_COLUMNS = 10;
    public final static int MAX_DIRTY_FIELDS = 10;
    public final static int MAX_CLEANED_FIELDS = 5;
    public final static SimpleDateFormat timeFormat =
            new SimpleDateFormat("HH:mm:ss");

    //Empty private constructor to prevent that this class can be instantiated.
    private Constants() {
    }
}

```

\dk\jsh\cleaningrobotsimulator\concurrent\Board.java

```java
package dk.jsh.cleaningrobotsimulator.concurrent;

import java.util.Date;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JTextArea;
import org.jdesktop.application.ResourceMap;

/**
 * A Board class. A board consist of 10x10 fields
 * (see {@link dk.jsh.cleaningrobotsimulator.concurrent.Field Field}).<br>
 * Each field can be either clean or dirty (one is the dustbin).
 * A field can only hold one robot.<br>
 * This class is thread safe.
 * @see dk.jsh.cleaningrobotsimulator.cuncurrent.Field
 * @author Jan S. Hansen
 */
public class Board {

    //Thread safety - following fields is guarded by "this".
    private Field[][] board;
    private int dirtyFieldsCounter;
    private long fieldsCleaned;
    private JTextArea jTextAreaDustbin;

    //Read-only fields.
    private ResourceMap resourceMap;

    /**
```

```java
     * Constructor.
     */
    public Board(ResourceMap resourceMap, JTextArea jTextAreaDustbin) {
        this.resourceMap = resourceMap;
        this.jTextAreaDustbin = jTextAreaDustbin;
        board = new Field[Constants.MAX_ROWS][Constants.MAX_COLUMNS];
        //Clean board
        for (int row = 0; row < Constants.MAX_ROWS; row++) {
            for (int column = 0; column < Constants.MAX_COLUMNS; column++) {
                Field field = new Field(column, row,
                        Field.Status.CLEAN, Field.UsedBy.EMPTY);
                Icon icon = resourceMap.getIcon("RobotSimulator.clean");
                field.jLabel.setIcon(icon);
                board[row][column] = field;
            }
        }
        setField(9, 0, Field.Status.CLEAN, Field.UsedBy.BENDER,
                "RobotSimulator.bender");
        setField(9, 9, Field.Status.CLEAN, Field.UsedBy.WALL_E,
                "RobotSimulator.wall-e");
        setField(0, 9, Field.Status.CLEAN, Field.UsedBy.ANDROID,
                "RobotSimulator.android");
        setField(0, 0, Field.Status.DUSTBIN, Field.UsedBy.EMPTY,
                "RobotSimulator.dustbin");
    }

    /**
     * Try to move a robot from one field to another field.
     * @param fromColumn from column
     * @param fromRow from row
     * @param toColumn to column
     * @param toRow to row
     * @param robotIconResource robot icon resource
     * @return true if move was a success.
     */
    public boolean tryMove(int fromColumn, int fromRow,
            int toColumn, int toRow, String robotIconResource) {
        testFieldArguments(fromColumn, fromRow);
        testFieldArguments(toColumn, toRow);
        boolean moveOk = false;
        synchronized (this) {
            Field fromField = getField(fromColumn, fromRow);
            Field toField = getField(toColumn, toRow);
            if (toField.isEmpty() && !fromField.isEmpty()) {
                toField.setUsedBy(fromField.getUsedBy());
                fromField.setUsedBy(Field.UsedBy.EMPTY);
                moveOk = true;
                //Set icons
                if (fromColumn == 0 && fromRow == 0) {
                    fromField.jLabel.setIcon(
                            resourceMap.getIcon("RobotSimulator.dustbin"));
                } else {
                    if (fromField.isDirty()) {
                        fromField.jLabel.setIcon(
                                resourceMap.getIcon("RobotSimulator.dirt"));
                    } else {
                        fromField.jLabel.setIcon(
                                resourceMap.getIcon("RobotSimulator.clean"));
                    }
                }
                if (toRow == 0 && toColumn == 0) {
                    toField.jLabel.setIcon(resourceMap.getIcon(
                            "RobotSimulator.recycle"));
                } else {
                    toField.jLabel.setIcon(resourceMap.getIcon(
                            robotIconResource));
                }
            }
            return moveOk;
        }
    }

    /**
     * Try to make a field dirty.
     * @param column fields column
```

```
 * @param row fields row
 * @return true if it was a success.
 */
public boolean tryMakeFieldDirty(int column, int row) {
    testFieldArguments(column, row);
    boolean ok = false;
    synchronized (this) {
        if (dirtyFieldsCounter + 1 <= Constants.MAX_DIRTY_FIELDS) {
            if (column == 0 && row == 0) { //Dustbin
               throw new IllegalArgumentException("Dustbin can't be dirty");
            }
            Field field = getField(column, row);
            if (field.isEmpty() && !field.isDirty()) {
                field.setStatus(Field.Status.DIRTY);
                dirtyFieldsCounter++;
                ok = true;
                field.jLabel.setIcon(
                        resourceMap.getIcon("RobotSimulator.dirt"));
            }
        }
        return ok;
    }
}

/**
 * Changes a fields status to clean.
 * @param column fields column
 * @param row fields row
 * @return true if it was a success.
 */
public boolean tryCleanField(int column, int row) {
    boolean ok = false;
    testFieldArguments(column, row);
    synchronized (this) {
        if (column == 0 && row == 0) { //Dustbin
            throw new IllegalArgumentException("Dustbin can't be cleaned");
        }
        Field field = getField(column, row);
        if (field.isDirty()) {
            field.setStatus(Field.Status.CLEAN);
            dirtyFieldsCounter--;
            ok = true;
        }
        return ok;
    }
}

/**
 * Empties a robot for dust and log a message to the Dustbin log.
 * @param robotName robot name, used in log message.
 */
public synchronized void emptyRobot(String robotName) {
    fieldsCleaned+=Constants.MAX_CLEANED_FIELDS;
    //Clear textArea after 2000 lines. TODO: Create a FIFO JTextArea
    if (jTextAreaDustbin.getLineCount() > 2000) {
        jTextAreaDustbin.setText("");
    }

    StringBuilder timeAndMessage =
            new StringBuilder(Constants.timeFormat.format(new Date()));
    timeAndMessage.append(" Dust from ").append(robotName);
    timeAndMessage.append(" recieved - Total recieved: ");
    timeAndMessage.append(fieldsCleaned).append(".\n");
    jTextAreaDustbin.append(timeAndMessage.toString());
}

/**
 * Returns dirty fields counter.
 * @return dirty fields counter
 */
public synchronized int getDirtyFieldsCounter() {
    return dirtyFieldsCounter;
}

/**
```

```
     * Returns a read only field. To prevent that a field is updated outside
     * this board instance.
     * @param column fields column
     * @param row fields row
     * @return field a ReadOnlyField
     */
    public ReadOnlyField getReadOnlyField(int column, int row) {
        testFieldArguments(column, row);
        synchronized (this) {
            return board[row][column];
        }
    }

    /**
     * Returns a field.
     * @param column fields column
     * @param row fields row
     * @return field a Field
     */
    private Field getField(int column, int row) {
        return board[row][column];
    }

    /**
     * Set a Fields Status and UsedBy.
     * @param column Fields column
     * @param row Fields row
     * @param status Fields Status
     * @param usedBy Fields UsedBy
     * @param iconResource Icon resource
     */
    private void setField(int column, int row, Field.Status status,
            Field.UsedBy usedBy, String iconResource) {
        testFieldArguments(column, row);
        Field field = board[row][column];
        field.setStatus(status);
        field.setUsedBy(usedBy);
        ImageIcon imageIcon = resourceMap.getImageIcon(iconResource);
        field.jLabel.setIcon(imageIcon);
    }

    /**
     * Test if is is valid column and row arguments.
     * @param column column
     * @param row row
     * @throws IllegalArgumentException Illegal row or column.
     */
    private void testFieldArguments(int column, int row)
            throws IllegalArgumentException {
        if (column < 0 || column >= Constants.MAX_COLUMNS
                || row < 0 || row >= Constants.MAX_ROWS) {
            throw new IllegalArgumentException("Error in column or row: ("
                    + column + ", " + row + ")");
        }
    }
}

```

\dk\jsh\cleaningrobotsimulator\concurrent\ReadOnlyField.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package dk.jsh.cleaningrobotsimulator.concurrent;

import javax.swing.JLabel;

/**
 * A read only Field interface. Used by Board, so it can return a read only
 * field.
 * @author Jan S. Hansen
```

```java
 */
public interface ReadOnlyField {
   public enum Status {CLEAN, DIRTY, DUSTBIN}
   public enum UsedBy {BENDER, WALL_E, ANDROID, EMPTY}

   public int getColumn();
   public int getRow();
   public Status getStatus();
   public UsedBy getUsedBy();
   public boolean isEmpty();
   public boolean isDirty();
   public JLabel getLabel();
}


```

\dk\jsh\cleaningrobotsimulator\concurrent\Field.java

```java
package dk.jsh.cleaningrobotsimulator.concurrent;

import javax.swing.JLabel;

/**
 * Field value object.<br>
 * A field has a row and a column, both starting from 0.<br>
 * A field has a status that can be CLEAR, DIRTY or DUSTBIN, and <br>
 * a field can be used by either robot BENDER, WALL-E of ANDROID or else the
 * field is EMPTY.
 * @author Jan S. Hansen
 */
public class Field implements ReadOnlyField {

   private Status status;
   private UsedBy usedBy;
   private int column;
   private int row;
   public JLabel jLabel;

   /**
    * Constructor.
    * @param column Fields column
    * @param row Fields row
    * @param status Fields Status
    * @param usedBy Fields UsedBy
    */
   public Field(int column, int row, Status status, UsedBy usedBy) {
      this.column = column;
      this.row = row;
      this.status = status;
      this.usedBy = usedBy;
      this.jLabel = new JLabel();
   }

   /**
    * Gets Fields column.
    * @return column number
    */
   @Override
   public int getColumn() {
      return column;
   }

   /**
    * Gets Fields row.
    * @return row number
    */
   @Override
   public int getRow() {
      return row;
   }

   /**
    * Gets Fields Status.
```

```java
     * @return Status
     */
    @Override
    public Status getStatus() {
        return status;
    }

    /**
     * Sets Fields Status
     * @param status Status
     */
    public void setStatus(Status status) {
        this.status = status;
    }

    /**
     * Gets Fields UsedBt
     * @return UsedBy
     */
    @Override
    public UsedBy getUsedBy() {
        return usedBy;
    }

    /**
     * Sets Fields UsedBy
     * @param usedBy UsedBy
     */
    public void setUsedBy(UsedBy usedBy) {
        this.usedBy = usedBy;
    }

    /**
     * Returns true if Field is empty.
     * @return true if empty
     */
    @Override
    public boolean isEmpty() {
        return usedBy == UsedBy.EMPTY;
    }

    /**
     * Returns true if Field is dirty.
     * @return true if dirty
     */
    @Override
    public boolean isDirty() {
        return status == Status.DIRTY;
    }

    /**
     * Returns JLabel.
     * @return JLabel
     */
    @Override
    public JLabel getLabel() {
        return jLabel;
    }

    /**
     * Test if this field is equal to a given object.</br>
     * Row and column is tested.
     * @param obj object to Test
     * @return true if equal.
     */
    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Field other = (Field) obj;
        if (this.column != other.column) {
            return false;
        }
        if (this.row != other.row) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 5;
        hash = 61 * hash + this.column;
        hash = 61 * hash + this.row;
        return hash;
    }
}

```

\dk\jsh\cleaningrobotsimulator\concurrent\BaseThread.java

```java
package dk.jsh.cleaningrobotsimulator.concurrent;

import java.util.Date;
import java.util.logging.Logger;
import javax.swing.JTextArea;
import org.jdesktop.application.ResourceMap;

/**
 * Abstract class with common thread functions
 * @author Jan S. Hansen
 */
abstract public class BaseThread extends Thread {
    protected JTextArea jTextArea;
    protected Board board;
    protected ResourceMap resourceMap;
    protected Logger exceptionLogger; //Logging of exceptions in a log file.

    /**
     * Constructor.
     * @param threadName thread name
     * @param board A Board object
     * @param jTextArea A JTextArea to use as log for this thread
     * @param resourceMap A ResourceMap
     */
    public BaseThread(String threadName, Board board, JTextArea jTextArea,
            ResourceMap resourceMap) {
        this.board = board;
        this.jTextArea = jTextArea;
        this.resourceMap = resourceMap;
        //If an exceptions occurs, the this name will be part of the exception
        //stacktrace.
        this.setName(threadName);
        exceptionLogger = Logger.getLogger(Robot.class.getName());
        setUncaughtExceptionHandler(new SimpleThreadExceptionHandler());
    }

    /**
     * Log a message a the JTestArea. See constructor.
     * @param message message to log.
     */
    protected void log(String message) {
        //Clear textArea after 2000 lines. TODO: Create a FIFO JTextArea
        if (jTextArea.getLineCount() > 2000) {
            jTextArea.setText("");
        }

        StringBuilder timeAndMessage =
                new StringBuilder(Constants.timeFormat.format(new Date()));
        timeAndMessage.append(" ").append(message).append("\n");
        jTextArea.append(timeAndMessage.toString());
    }

    /**
```

```java
     * Log that an exception has occured in the thread.
     */
    protected void logException() {
        log("The thread is stopped, due to an exception, see log file.");
    }
}

```

\dk\jsh\cleaningrobotsimulator\concurrent\Robot.java

```java
package dk.jsh.cleaningrobotsimulator.concurrent;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Random;
import java.util.logging.Level;
import javax.swing.JTextArea;
import org.jdesktop.application.ResourceMap;

/**
 * Robot thread.
 * @author Jan S. Hansen
 */
public class Robot extends BaseThread {

    private boolean stopRequested = false;
    private boolean pauseRequested = false;
    private String resource;
    private String fullResource;
    private int column;
    private int row;
    private ReadOnlyField[] prevFields =
            new ReadOnlyField[]{null, null, null, null, null, null};
    private int nextPrevField;
    private int fieldsCleaned;
    Random randomGenerator = new Random();

    /**
     * Constructor.
     * @param threadName Thread name
     * @param board A Board object
     * @param jTextArea A JTextArea to use as log for this thread
     * @param resourceMap A ResourceMap
     * @param resource Robots normal icon resource
     * @param fullResource Robots full icon resource
     * @param row Robots start row position
     * @param column Robots start column position
     */
    public Robot(String threadName, Board board, JTextArea jTextArea,
            ResourceMap resourceMap,
            String resource, String fullResource,
            int row, int column) {
        super(threadName, board, jTextArea, resourceMap);
        this.resource = resource;
        this.fullResource = fullResource;
        this.column = column;
        this.row = row;
        this.resourceMap = resourceMap;
    }

    /**
     * The threads run method.
     */
    @Override
    public void run() {
        log("Thread for robot is now running.");
        while (!isStopRequested()) {
            if (isPauseRequested()) {
                paused();
            } else {
                cleaning();
            }
        }
        log("Thread for robot is now stopped");
    }

    /**
     * Robot is in cleaning mode
     */
    private void cleaning() {
        logPrevFields();
        addToPrevFields(board.getReadOnlyField(column, row));
        if (fieldsCleaned >= Constants.MAX_CLEANED_FIELDS) { //Goto bin
            gotoDustbinMode();
        } else { //Search and clean
            cleaningMode();
        }
        sleepForSecs(1);
    }

    /**
     * Search for dirty nearby fields. If found clean a random
     * dirty field, else goto randon a nearby empty and clean field.
     */
    private void cleaningMode() {
        //Search and clean
        ReadOnlyField moveToField = getNextField();
        if (moveToField == null) {
            clearPrevFields();
        } else {
            int toColumn = moveToField.getColumn();
            int toRow = moveToField.getRow();
            logMove("Try move", row, column, toRow, toColumn);
            if (board.tryMove(column, row, toColumn, toRow, resource)) {
                logMove("Moved from", row, column, toRow, toColumn);
                if (moveToField.isDirty()) {
                    if (board.tryCleanField(toColumn, toRow)) {
                        fieldsCleaned++;
                        log("Number of fields cleaned: " + fieldsCleaned + ".");
                        if (fieldsCleaned >= Constants.MAX_CLEANED_FIELDS) {
                            log("Robot is full.");
                        }
                    } else {
                        log("*** The field is no longer dirty, after moving " + "robot.");
                    }
                }
                row = toRow;
                column = toColumn;
            } else {
                log("*** Move failed.");
            }
        }
    }

    /**
     * Move robot closer to the dustbin. If robot is on the dustbin field, the
     * robot is emptied.
     */
    private void gotoDustbinMode() {
        //Goto bin
        int toRow = row > 0 ? row - 1 : 0;
        int toColumn = column > 0 ? column - 1 : 0;
        if (board.tryMove(column, row, toColumn, toRow, fullResource)) {
            logMove("Move to dustbin", row, column, toRow, toColumn);
            if (toRow == 0 && toColumn == 0) {
                fieldsCleaned = 0;
                board.emptyRobot(this.getName());
                clearPrevFields();
                log("Robot is emptied.");
            }
            row = toRow;
            column = toColumn;
        } else {
            log("*** Move to dustbin failed.");
        }
    }

```

```
/**
 * Paused this thread for 1 second.
 */
private void paused() {
    sleepForSecs(1);
}

/**
 * Makes this thread goto sleep for a given number of seconds.
 * @param secs seconds
 */
private void sleepForSecs(int secs) {
    try {
        int msecs = secs * 1000;
        int i = 0;
        while (i < (msecs / 100) && !isStopRequested()) {
            sleep(100);
            i++;
        }
    } catch (InterruptedException ex) {
        exceptionLogger.log(Level.SEVERE, null, ex);
        logException();
        requestStop();
    }
}

/**
 * Request this thread to stop
 */
public synchronized void requestStop() {
    log("Stop requested for robot.");
    stopRequested = true;
}

/**'
 * Returns true if this thread is requested to stop.
 * @return true if this thread is requested to stop
 */
private synchronized boolean isStopRequested() {
    return stopRequested;
}

/**
 * Request this thread to go into pause mode.
 */
public synchronized void requestPause() {
    log("Pause requested for robot.");
    pauseRequested = true;
}

/**
 * Request this thread to go into running mode.
 */
public synchronized void continueAfterPause() {
    log("Continue requested for robot.");
    pauseRequested = false;
}

/**
 * Returns true if this thread is requested to go into pause mode.
 * @return true if this thread is requested to go into pause mode
 */
private synchronized boolean isPauseRequested() {
    return pauseRequested;
}

/**
 * Returns the next field the Robot should try to go to. Dirty Fields has
 * priority.
 * @return A Field or null if no move is possible
 */
private ReadOnlyField getNextField() {
    List<ReadOnlyField> moveToCleanFieldOptions =
            new ArrayList<ReadOnlyField>();
    List<ReadOnlyField> moveToDirtyFieldOptions =
            new ArrayList<ReadOnlyField>();
    //Test fields above
    int testColumn = column - 1;
    int testRow = row - 1;
    for (testColumn = column - 1; testColumn <= column + 1; testColumn++) {
        if (validRowColumn(testColumn, testRow)) {
            ReadOnlyField field = board.getReadOnlyField(testColumn, testRow);
            if (field.isEmpty()) {
                if (field.isDirty()) {
                    moveToDirtyFieldOptions.add(field);
                } else {
                     if(!isFieldInPrevFields(field)) {
                        moveToCleanFieldOptions.add(field);
                     }
                }
            }
        }
    }
    //Test field to the left
    testRow = row;
    testColumn = column - 1;
    if (validRowColumn(testColumn, testRow)) {
        ReadOnlyField field = board.getReadOnlyField(testColumn, testRow);
        if (field.isEmpty()) {
            if (field.isDirty()) {
                moveToDirtyFieldOptions.add(field);
            } else {
                 if(!isFieldInPrevFields(field)) {
                    moveToCleanFieldOptions.add(field);
                 }
            }
        }
    }
    //Test field to the right
    testColumn = column + 1;
    if (validRowColumn(testColumn, testRow)) {
        ReadOnlyField field = board.getReadOnlyField(testColumn, testRow);
        if (field.isEmpty()) {
            if (field.isDirty()) {
                moveToDirtyFieldOptions.add(field);
            } else {
                 if(!isFieldInPrevFields(field)) {
                    moveToCleanFieldOptions.add(field);
                 }
            }
        }
    }
    //Test fields below
    testColumn = column - 1;
    testRow = row + 1;
    for (testColumn = column - 1; testColumn <= column + 1; testColumn++) {
        if (validRowColumn(testColumn, testRow)) {
            ReadOnlyField field =
                    board.getReadOnlyField(testColumn, testRow);
            if (field.isEmpty()) {
                if (field.isDirty()) {
                    moveToDirtyFieldOptions.add(field);
                } else {
                     if(!isFieldInPrevFields(field)) {
                        moveToCleanFieldOptions.add(field);
                     }
                }
            }
        }
    }
    ReadOnlyField field = null;
    if (!moveToDirtyFieldOptions.isEmpty()) {
        logMoveToOptions("Move to dirty field options",
                moveToDirtyFieldOptions);
        //Return random
        int index = randomGenerator.nextInt(moveToDirtyFieldOptions.size());
        field = moveToDirtyFieldOptions.get(index);
    } else { //No dirty fields to move to, try clean fields.
        log("No dirty fields nearby.");
        if (!moveToCleanFieldOptions.isEmpty()) {
```

```java
            logMoveToOptions("Move to clean field options",
                    moveToCleanFieldOptions);
            int index = randomGenerator.nextInt(
                    moveToCleanFieldOptions.size());
            field = moveToCleanFieldOptions.get(index);
        } else {
            log("*** Robot is locked, no move is possible!");
        }
    }
    return field;
}

/**
 * Test if a given column and row is valid.
 * @param column Column
 * @param row Row
 * @return true if valid pair of column and row
 */
private boolean validRowColumn(int column, int row) {
    boolean ok = true;
    if (row < 0 || row >= Constants.MAX_ROWS
            || column < 0 || column >= Constants.MAX_COLUMNS) {
        ok = false;
    }
    if (column == 0 && row == 0) { //Dustbin
        ok = false;
    }
    return ok;
}

/**
 * Add a Field to a circular buffer with previous fields this Robot has
 * visited.
 * @param field ReadOnlyField to add to buffer
 */
private void addToPrevFields(ReadOnlyField field) {
    prevFields[nextPrevField] = field;
    nextPrevField++;
    if (nextPrevField > prevFields.length - 1) {
        nextPrevField = 0;
    }
}

/**
 * Clear a circular buffer with previous fields this Robot has
 * visited.
 */
private void clearPrevFields() {
    log("Clear prev. fields.");
    for (int i = 0; i < prevFields.length; i++) {
        prevFields[i] = null;
    }
    nextPrevField = 0;
}

/**
 * Returns true if this field is in the circular buffer with previous
 * fields.
 * @param field ReadOnlyField to test
 * @return true if this field is in the circular buffer with previous
 * fields.
 */
private boolean isFieldInPrevFields(ReadOnlyField field) {
    int i = 0;
    boolean fieldFound = false;
    while (!fieldFound && i < prevFields.length) {
        if (field.equals(prevFields[i])) {
            fieldFound = true;
        } else {
            i++;
        }
    }
    return fieldFound;
}

```

```java
    /**
     * Log a move.
     * @param message Message before from and to text.
     * @param fromRow from row
     * @param fromColumn from column
     * @param toRow to row
     * @param toColumn to column
     */
    private void logMove(String message,
            int fromRow, int fromColumn,
            int toRow, int toColumn) {
        StringBuilder timeAndMessage =
                new StringBuilder(Constants.timeFormat.format(new Date()));
        timeAndMessage.append(" ").append(message).append(" ");
        timeAndMessage.append((char) (fromColumn + 65));
        timeAndMessage.append(++fromRow).append(" to ");
        timeAndMessage.append((char) (toColumn + 65));
        timeAndMessage.append(++toRow).append(".\n");
        jTextArea.append(timeAndMessage.toString());
    }

    /**
     * Log all move to options.
     * @param message Message before options
     * @param fields A List of Fields
     */
    private void logMoveToOptions(String message, List<ReadOnlyField> fields) {
        StringBuilder timeAndMessage =
                new StringBuilder(Constants.timeFormat.format(new Date()));
        timeAndMessage.append(" ").append(message);
        String before = ": ";
        for (ReadOnlyField field : fields) {
            timeAndMessage.append(before);
            timeAndMessage.append((char) (field.getColumn() + 65));
            timeAndMessage.append(field.getRow() + 1);
            before = ", ";
        }
        timeAndMessage.append(".\n");
        jTextArea.append(timeAndMessage.toString());
    }

    /**
     * Log prev. fields.
     */
    private void logPrevFields() {
        StringBuilder timeAndMessage =
                new StringBuilder(Constants.timeFormat.format(new Date()));
        timeAndMessage.append(" Previous fields: ");
        String before = null;
        int i = nextPrevField;
        boolean noPrevFields = true;
        for (int c = 0; c < prevFields.length; c++) {
            ReadOnlyField field = prevFields[i];
            if (field != null) {
                noPrevFields = false;
                if (before != null) {
                    timeAndMessage.append(before);
                }
                timeAndMessage.append((char) (field.getColumn() + 65));
                timeAndMessage.append(field.getRow() + 1);
                before = ", ";
            }
            i++;
            if (i > prevFields.length - 1) {
                i  = 0;
            }
        }
        if (noPrevFields) {
            timeAndMessage.append("No previous fields");
        }
        timeAndMessage.append(".\n");
        jTextArea.append(timeAndMessage.toString());
    }
}

```

```java
package dk.jsh.cleaningrobotsimulator.concurrent;

import java.util.Date;
import java.util.Random;
import java.util.logging.Level;
import javax.swing.JTextArea;
import org.jdesktop.application.ResourceMap;

/**
 * Dust creator thread.
 * @author Jan S. Hansen
 */
public class DustCreator extends BaseThread {
   Random randomGenerator = new Random();

   /**
    * Constructor.
    * @param threadName Thread name
    * @param board A Board object
    * @param jTextArea A JTextArea to use as log for this thread.
    * @param resourceMap A ResourceMap
    */
   public DustCreator(String threadName, Board board, JTextArea jTextArea,
           ResourceMap resourceMap) {
       super(threadName, board, jTextArea, resourceMap);
   }

   /**
    * The threads run method.
    */
   @Override
   public void run() {
       log("Thread for dust creator is now running.");
       int dirtyFields = board.getDirtyFieldsCounter();
       log("Dirty fields on board: " + dirtyFields);
       while (dirtyFields < Constants.MAX_DIRTY_FIELDS) {
           int row = randomGenerator.nextInt(Constants.MAX_ROWS);
           int column = randomGenerator.nextInt(Constants.MAX_COLUMNS);
           if (row != 0 || column != 0) { //Dustbin
               logTrySetFieldDirty(row, column);
               if (board.tryMakeFieldDirty(column, row)) {
                   dirtyFields++;
                   log("Dirt added.");
               }
               else {
                   log("Failed.");
               }
           }
           sleepForSecs(1);
       }
      log("Thread for dust creator is now finished.");
   }

   /**
    * Log a "Try put dirt on field" message.
    * @param row fields row, used in log message, converted to row + 1
    * @param column fields column, used in log message, converted to A, B, C
    * etc.
    */
   private void logTrySetFieldDirty(int row, int column) {
       StringBuilder timeAndMessage =
               new StringBuilder(Constants.timeFormat.format(new Date()));
       timeAndMessage.append(" Try put dirt on field ");
       timeAndMessage.append((char)(column + 65));
       timeAndMessage.append(++row).append(".\n");;
       jTextArea.append(timeAndMessage.toString());
   }

   /**
    * Makes this thread goto sleep for a given number of seconds.
```

```java
     * @param secs seconds
     */
    private void sleepForSecs(int secs) {
        try {
            sleep(secs * 1000);
        } catch (InterruptedException ex) {
            exceptionLogger.log(Level.SEVERE, null, ex);
            logException();
        }
    }
}

```

\dk\jsh\cleaningrobotsimulator\concurrent\SimpleThreadExceptionHandler.java

```java
package dk.jsh.cleaningrobotsimulator.concurrent;

import java.io.PrintWriter;
import java.io.StringWriter;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * This class is used to handle uncaught exceptions in threads.
 * @author Jan S. Hansen
 */
public class SimpleThreadExceptionHandler
        implements Thread.UncaughtExceptionHandler {
    private Logger logger;

    /**
     * Constructor.
     */
    public SimpleThreadExceptionHandler() {
        logger = Logger.getLogger(getClass().getName());
    }

    /**
     * Log uncaugth exceptions to a log file and to the standard error stream.
     * @param thread The thread that throw the exception
     * @param exception Exception.
     */
    @Override
    public void uncaughtException(Thread thread, Throwable exception) {
        exception.printStackTrace();
        StringWriter sw = new StringWriter();
        exception.printStackTrace(new PrintWriter(sw));
        logger.log(Level.SEVERE, "Uncaught exception in thread",
                thread.getName());
        logger.log(Level.SEVERE, "Uncaught exception in thread", sw.toString());
        if (thread instanceof Robot) {
            Robot robot = (Robot)thread;
            robot.logException();
        }
    }
}

```

dk.jsh.cleaningrobotsimulator.ui.swing

\dk\jsh\cleaningrobotsimulator\ui\swing\CleaningRobotSimulator.java

```java
package dk.jsh.cleaningrobotsimulator.ui.swing;

import java.util.logging.FileHandler;
import java.util.logging.Handler;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.jdesktop.application.Application;
import org.jdesktop.application.SingleFrameApplication;

/**
 * The main class of the application.
 * @author Jan S. Hansen
 */
public class CleaningRobotSimulator extends SingleFrameApplication {

    /**
     * At startup create and show the main frame of the application.
     */
    @Override protected void startup() {
        show(new View(this));
    }

    /**
     * Setup log file.
     */
    private static void setupLog() {
        try {
            //%t - Means that the log is located in the Systems Temp directory
            Handler fh = new FileHandler("%t/cleaning-robot-simulator.log",
                    10000, 5);
            Logger logger = Logger.getLogger("");
            logger.addHandler(fh);
            logger.setLevel(Level.INFO);
            logger.info("Application started.");
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    /**
     * This method is to initialize the specified window by injecting resources.
     * Windows shown in our application come fully initialized from the GUI
     * builder, so this additional configuration is not needed.
     */
    @Override protected void configureWindow(java.awt.Window root) {
    }

    /**
     * A convenient static getter for the application instance.
     * @return the instance of CleaningRobotSimulator
     */
    public static CleaningRobotSimulator getApplication() {
        return Application.getInstance(CleaningRobotSimulator.class);
    }

    /**
     * Main method launching the application.
     */
    public static void main(String[] args) {
        setupLog();
        Thread.setDefaultUncaughtExceptionHandler(
                new SimpleMainThreadExceptionHandler());
        launch(CleaningRobotSimulator.class, args);
    }
}

```

\dk\jsh\cleaningrobotsimulator\ui\swing\View.java

```java
package dk.jsh.cleaningrobotsimulator.ui.swing;

import dk.jsh.cleaningrobotsimulator.concurrent.Board;
import dk.jsh.cleaningrobotsimulator.concurrent.Constants;
import dk.jsh.cleaningrobotsimulator.concurrent.DustCreator;
import dk.jsh.cleaningrobotsimulator.concurrent.ReadOnlyField;
import dk.jsh.cleaningrobotsimulator.concurrent.Robot;
import java.awt.GridBagConstraints;
import java.awt.Insets;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.WindowEvent;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.ImageIcon;
import org.jdesktop.application.Action;
import org.jdesktop.application.ResourceMap;
import org.jdesktop.application.SingleFrameApplication;
import org.jdesktop.application.FrameView;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;

/**
 * The application's main frame.
 * @author Jan S. Hansen
 */
public class View extends FrameView {

    private Board board;
    private ResourceMap resourceMap;
    private Robot bender;
    private Robot android;
    private Robot wallE;
    private Logger logger;

    /**
     * Constructor.
     */
    public View(SingleFrameApplication app) {
        super(app);
        logger = Logger.getLogger(View.class.getName());

        resourceMap = getResourceMap();

        //Cacth windowClosing event
        JFrame jFrame = this.getFrame();
        jFrame.addWindowListener(new java.awt.event.WindowAdapter() {

            @Override
            public void windowClosing(WindowEvent winEvt) {
                quit();
            }
        });

        //Set icon in upper left corner
        ImageIcon image = resourceMap.getImageIcon("RobotSimulator.recycle");
        jFrame.setIconImage(image.getImage());

        //Initialize UI
        initComponents();

        //Create board
        board = new Board(resourceMap, jTextAreaDustbin);

        //Set tab icons
        jTabbedPane1.setIconAt(0, resourceMap.getIcon("RobotSimulator.bender"));
        jTabbedPane1.setIconAt(1, resourceMap.getIcon("RobotSimulator.wall-e"));
        jTabbedPane1.setIconAt(2, resourceMap.getIcon("RobotSimulator.android"));
        jTabbedPane1.setIconAt(3, resourceMap.getIcon("RobotSimulator.dirt"));
        jTabbedPane1.setIconAt(4, resourceMap.getIcon("RobotSimulator.dustbin"));
        jTabbedPane1.setSelectedIndex(0);

        jButtonContinue.setEnabled(false);

        //Set JFrame's min. hight and width.
        jFrame.addComponentListener(new ComponentAdapter() {
            private final static int MIN_WIDTH = 855;
            private final static int MIN_HIGHT = 450;
            @Override
            public void componentResized(ComponentEvent e) {
                JFrame frame = (JFrame) e.getSource();
                int width = frame.getWidth() < MIN_WIDTH
                        ? MIN_WIDTH : frame.getWidth();
                int hight = frame.getHeight() < MIN_HIGHT
                        ? MIN_HIGHT : frame.getHeight();
                frame.setSize(width, hight);
            }
        });

        createUIBoard();

        //Start robot threads
        bender = new Robot("Bender", board, jTextAreaBender, resourceMap,
                "RobotSimulator.bender", "RobotSimulator.bender-full", 0, 9);
        bender.start();

         android = new Robot("Android", board, jTextAreaAndroid, resourceMap,
                 "RobotSimulator.android", "RobotSimulator.android-full", 9, 0);
         android.start();

         wallE = new Robot("Wall-E", board, jTextAreaWallE, resourceMap,
                 "RobotSimulator.wall-e", "RobotSimulator.wall-e-full", 9, 9);
         wallE.start();

         //Get a scheduler
         ScheduledExecutorService scheduler =
                 Executors.newSingleThreadScheduledExecutor();
         //Run DustCreator with a 30 secs. delay between each run.
         scheduler.scheduleWithFixedDelay(
                 new DustCreator("DustCreator", board, jTextAreaDust,
                 resourceMap), 0, 30, TimeUnit.SECONDS);
    }

    /**
     * Adds fields from board to UI.
     */
    private void createUIBoard() {
        GridBagConstraints gridBagConstraints = new java.awt.GridBagConstraints();
        Insets insets = new Insets(1, 1, 1, 1);
        for (int row = 0; row < Constants.MAX_ROWS; row++) {
            for (int column = 0; column < Constants.MAX_COLUMNS; column++) {
                ReadOnlyField field = board.getReadOnlyField(column, row);
                JLabel jLabel = field.getLabel();
                gridBagConstraints.gridx = column + 1;
                gridBagConstraints.gridy = row + 1;
                gridBagConstraints.insets = insets;
                mainPanel.add(jLabel, gridBagConstraints);
            }
        }
    }

    /**
     * Show about box action.
     */
    @Action
    public void showAboutBox() {
        if (aboutBox == null) {
            JFrame mainFrame = CleaningRobotSimulator.getApplication().getMainFrame();
            aboutBox = new AboutBox(mainFrame);
            aboutBox.setLocationRelativeTo(mainFrame);
        }
        CleaningRobotSimulator.getApplication().show(aboutBox);
    }

    /**
     * Pause button action.
     */
    @Action
    public void pause() {
        bender.requestPause();
        android.requestPause();
        wallE.requestPause();
        jButtonPause.setEnabled(false);
        jButtonContinue.setEnabled(true);
    }

    /**
     * Continue button action.
     */
    @Action
    public void cont() {
        bender.continueAfterPause();
        android.continueAfterPause();
        wallE.continueAfterPause();
        jButtonPause.setEnabled(true);
        jButtonContinue.setEnabled(false);
    }

    /**
     * Quit application action.
     */
    @Action
    public void quit() {
        bender.requestStop();
        android.requestStop();
        wallE.requestStop();
        while (bender.isAlive() || android.isAlive() || wallE.isAlive()) {
            try {
                Thread.sleep(50);
            } catch (InterruptedException ex) {
                logger.log(Level.SEVERE, "Error waiting for robots to stop.",
                        ex);
            }
        }
        logger.log(Level.INFO, "Application stopped.");
        System.exit(0);
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        java.awt.GridBagConstraints gridBagConstraints;

        menuBar = new javax.swing.JMenuBar();
        javax.swing.JMenu fileMenu = new javax.swing.JMenu();
        javax.swing.JMenuItem exitMenuItem = new javax.swing.JMenuItem();
        javax.swing.JMenu helpMenu = new javax.swing.JMenu();
        javax.swing.JMenuItem aboutMenuItem = new javax.swing.JMenuItem();
        mainPanel = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jLabel8 = new javax.swing.JLabel();
        jLabel9 = new javax.swing.JLabel();
        jLabel10 = new javax.swing.JLabel();
        jLabel11 = new javax.swing.JLabel();
        jLabel12 = new javax.swing.JLabel();
        jLabel13 = new javax.swing.JLabel();
        jLabel14 = new javax.swing.JLabel();
        jLabel15 = new javax.swing.JLabel();
        jLabel16 = new javax.swing.JLabel();
        jLabel17 = new javax.swing.JLabel();
        jLabel18 = new javax.swing.JLabel();
        jLabel19 = new javax.swing.JLabel();
        jLabel20 = new javax.swing.JLabel();
        jLabel21 = new javax.swing.JLabel();
        jTabbedPane1 = new javax.swing.JTabbedPane();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTextAreaBender = new javax.swing.JTextArea();
        jScrollPane2 = new javax.swing.JScrollPane();
        jTextAreaWallE = new javax.swing.JTextArea();
        jScrollPane3 = new javax.swing.JScrollPane();
        jTextAreaAndroid = new javax.swing.JTextArea();
        jScrollPane4 = new javax.swing.JScrollPane();
        jTextAreaDust = new javax.swing.JTextArea();
        jScrollPane5 = new javax.swing.JScrollPane();
        jTextAreaDustbin = new javax.swing.JTextArea();
        jButtonPause = new javax.swing.JButton();
        jButtonContinue = new javax.swing.JButton();

        menuBar.setName("menuBar"); // NOI18N

        org.jdesktop.application.ResourceMap resourceMap = org.jdesktop.application.Application.getInstance(dk.jsh.cleaningrobotsimulator.ui.swing.CleaningRobotSimulator.class).getContext().getResourceMap(View.class);
        fileMenu.setText(resourceMap.getString("fileMenu.text")); // NOI18N
        fileMenu.setName("fileMenu"); // NOI18N

        javax.swing.ActionMap actionMap = org.jdesktop.application.Application.getInstance(dk.jsh.cleaningrobotsimulator.ui.swing.CleaningRobotSimulator.class).getContext().getActionMap(View.class, this);
        exitMenuItem.setAction(actionMap.get("quit")); // NOI18N
        exitMenuItem.setIcon(resourceMap.getIcon("exitMenuItem.icon")); // NOI18N
        exitMenuItem.setName("exitMenuItem"); // NOI18N
        fileMenu.add(exitMenuItem);

        menuBar.add(fileMenu);

        helpMenu.setText(resourceMap.getString("helpMenu.text")); // NOI18N
        helpMenu.setName("helpMenu"); // NOI18N

        aboutMenuItem.setAction(actionMap.get("showAboutBox")); // NOI18N
        aboutMenuItem.setIcon(resourceMap.getIcon("aboutMenuItem.icon")); // NOI18N
        aboutMenuItem.setDisabledIcon(resourceMap.getIcon("aboutMenuItem.disabledIcon")); // NOI18N
        aboutMenuItem.setName("aboutMenuItem"); // NOI18N
        helpMenu.add(aboutMenuItem);

        menuBar.add(helpMenu);

        mainPanel.setMinimumSize(new java.awt.Dimension(313, 240));
        mainPanel.setName("mainPanel"); // NOI18N
        mainPanel.setLayout(new java.awt.GridBagLayout());

        jLabel1.setText(resourceMap.getString("jLabel1.text")); // NOI18N
        jLabel1.setName("jLabel1"); // NOI18N
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        mainPanel.add(jLabel1, gridBagConstraints);

        jLabel2.setText(resourceMap.getString("jLabel2.text")); // NOI18N
        jLabel2.setName("jLabel2"); // NOI18N
        mainPanel.add(jLabel2, new java.awt.GridBagConstraints());

        jLabel3.setText(resourceMap.getString("jLabel3.text")); // NOI18N
        jLabel3.setName("jLabel3"); // NOI18N
        mainPanel.add(jLabel3, new java.awt.GridBagConstraints());

        jLabel4.setText(resourceMap.getString("jLabel4.text")); // NOI18N
        jLabel4.setName("jLabel4"); // NOI18N
        mainPanel.add(jLabel4, new java.awt.GridBagConstraints());

        jLabel5.setText(resourceMap.getString("jLabel5.text")); // NOI18N
        jLabel5.setName("jLabel5"); // NOI18N
        mainPanel.add(jLabel5, new java.awt.GridBagConstraints());

        jLabel6.setText(resourceMap.getString("jLabel6.text")); // NOI18N
        jLabel6.setName("jLabel6"); // NOI18N
        mainPanel.add(jLabel6, new java.awt.GridBagConstraints());

        jLabel7.setText(resourceMap.getString("jLabel7.text")); // NOI18N
        jLabel7.setName("jLabel7"); // NOI18N
        mainPanel.add(jLabel7, new java.awt.GridBagConstraints());

        jLabel8.setText(resourceMap.getString("jLabel8.text")); // NOI18N
        jLabel8.setName("jLabel8"); // NOI18N
        mainPanel.add(jLabel8, new java.awt.GridBagConstraints());

        jLabel9.setText(resourceMap.getString("jLabel9.text")); // NOI18N
        jLabel9.setName("jLabel9"); // NOI18N
        mainPanel.add(jLabel9, new java.awt.GridBagConstraints());

        jLabel10.setText(resourceMap.getString("jLabel10.text")); // NOI18N
        jLabel10.setName("jLabel10"); // NOI18N
        mainPanel.add(jLabel10, new java.awt.GridBagConstraints());

        jLabel11.setText(resourceMap.getString("jLabel11.text")); // NOI18N
        jLabel11.setName("jLabel11"); // NOI18N
        mainPanel.add(jLabel11, new java.awt.GridBagConstraints());

        jLabel12.setText(resourceMap.getString("jLabel12.text")); // NOI18N
        jLabel12.setName("jLabel12"); // NOI18N
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 1;
        mainPanel.add(jLabel12, gridBagConstraints);

        jLabel13.setText(resourceMap.getString("jLabel13.text")); // NOI18N
        jLabel13.setName("jLabel13"); // NOI18N
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 2;
        mainPanel.add(jLabel13, gridBagConstraints);

        jLabel14.setText(resourceMap.getString("jLabel14.text")); // NOI18N
        jLabel14.setName("jLabel14"); // NOI18N
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 3;
        mainPanel.add(jLabel14, gridBagConstraints);

        jLabel15.setText(resourceMap.getString("jLabel15.text")); // NOI18N
        jLabel15.setName("jLabel15"); // NOI18N
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 4;
        mainPanel.add(jLabel15, gridBagConstraints);

        jLabel16.setText(resourceMap.getString("jLabel16.text")); // NOI18N
        jLabel16.setName("jLabel16"); // NOI18N
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 5;
        mainPanel.add(jLabel16, gridBagConstraints);

        jLabel17.setText(resourceMap.getString("jLabel17.text")); // NOI18N
        jLabel17.setName("jLabel17"); // NOI18N
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 6;
        mainPanel.add(jLabel17, gridBagConstraints);

        jLabel18.setText(resourceMap.getString("jLabel18.text")); // NOI18N
        jLabel18.setName("jLabel18"); // NOI18N
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 7;
        mainPanel.add(jLabel18, gridBagConstraints);

        jLabel19.setText(resourceMap.getString("jLabel19.text")); // NOI18N
        jLabel19.setName("jLabel19"); // NOI18N
```

```
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 8;
mainPanel.add(jLabel19, gridBagConstraints);

jLabel20.setText(resourceMap.getString("jLabel20.text")); // NOI18N
jLabel20.setName("jLabel20"); // NOI18N
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 9;
mainPanel.add(jLabel20, gridBagConstraints);

jLabel21.setText(resourceMap.getString("jLabel21.text")); // NOI18N
jLabel21.setName("jLabel21"); // NOI18N
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 10;
mainPanel.add(jLabel21, gridBagConstraints);

jTabbedPane1.setBorder(javax.swing.BorderFactory.createEmptyBorder(1, 1, 1, 1));
jTabbedPane1.setName("jTabbedPane1"); // NOI18N

jScrollPane1.setName("jScrollPane1"); // NOI18N

jTextAreaBender.setColumns(20);
jTextAreaBender.setEditable(false);
jTextAreaBender.setRows(5);
jTextAreaBender.setName("jTextAreaBender"); // NOI18N
jScrollPane1.setViewportView(jTextAreaBender);

jTabbedPane1.addTab(resourceMap.getString("jScrollPane1.TabConstraints.tabTitle"), jScrollPane1); // NOI18N

jScrollPane2.setName("jScrollPane2"); // NOI18N

jTextAreaWallE.setColumns(20);
jTextAreaWallE.setEditable(false);
jTextAreaWallE.setRows(5);
jTextAreaWallE.setName("jTextAreaWallE"); // NOI18N
jScrollPane2.setViewportView(jTextAreaWallE);

jTabbedPane1.addTab(resourceMap.getString("jScrollPane2.TabConstraints.tabTitle"), jScrollPane2); // NOI18N

jScrollPane3.setName("jScrollPane3"); // NOI18N

jTextAreaAndroid.setColumns(20);
jTextAreaAndroid.setEditable(false);
jTextAreaAndroid.setRows(5);
jTextAreaAndroid.setName("jTextAreaAndroid"); // NOI18N
jScrollPane3.setViewportView(jTextAreaAndroid);

jTabbedPane1.addTab(resourceMap.getString("jScrollPane3.TabConstraints.tabTitle"), jScrollPane3); // NOI18N

jScrollPane4.setName("jScrollPane4"); // NOI18N

jTextAreaDust.setColumns(20);
jTextAreaDust.setEditable(false);
jTextAreaDust.setRows(5);
jTextAreaDust.setName("jTextAreaDust"); // NOI18N
jScrollPane4.setViewportView(jTextAreaDust);

jTabbedPane1.addTab(resourceMap.getString("jScrollPane4.TabConstraints.tabTitle"), jScrollPane4); // NOI18N

jScrollPane5.setName("jScrollPane5"); // NOI18N

jTextAreaDustbin.setColumns(20);
jTextAreaDustbin.setEditable(false);
jTextAreaDustbin.setRows(5);
jTextAreaDustbin.setName("jTextAreaDustbin"); // NOI18N
jScrollPane5.setViewportView(jTextAreaDustbin);

jTabbedPane1.addTab(resourceMap.getString("jScrollPane5.TabConstraints.tabTitle"), jScrollPane5); // NOI18N

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridwidth = 15;
gridBagConstraints.gridheight = 12;
```

```
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.weightx = 1.0;
        gridBagConstraints.weighty = 1.0;
        gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);
        mainPanel.add(jTabbedPane1, gridBagConstraints);

        jButtonPause.setAction(actionMap.get("pause")); // NOI18N
        jButtonPause.setText(resourceMap.getString("jButtonPause.text")); // NOI18N
        jButtonPause.setName("jButtonPause"); // NOI18N
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 11;
        gridBagConstraints.gridwidth = 4;
        gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.SOUTH;
        gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);
        mainPanel.add(jButtonPause, gridBagConstraints);

        jButtonContinue.setAction(actionMap.get("cont")); // NOI18N
        jButtonContinue.setText(resourceMap.getString("jButtonContinue.text")); // NOI18N
        jButtonContinue.setName("jButtonContinue"); // NOI18N
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 7;
        gridBagConstraints.gridy = 11;
        gridBagConstraints.gridwidth = 4;
        gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.SOUTH;
        gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 0);
        mainPanel.add(jButtonContinue, gridBagConstraints);

        setComponent(mainPanel);
        setMenuBar(menuBar);
    }// </editor-fold>
    // Variables declaration - do not modify
    private javax.swing.JButton jButtonContinue;
    private javax.swing.JButton jButtonPause;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel18;
    private javax.swing.JLabel jLabel19;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel20;
    private javax.swing.JLabel jLabel21;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane3;
    private javax.swing.JScrollPane jScrollPane4;
    private javax.swing.JScrollPane jScrollPane5;
    private javax.swing.JTabbedPane jTabbedPane1;
    private javax.swing.JTextArea jTextAreaAndroid;
    private javax.swing.JTextArea jTextAreaBender;
    private javax.swing.JTextArea jTextAreaDust;
    private javax.swing.JTextArea jTextAreaDustbin;
    private javax.swing.JTextArea jTextAreaWallE;
    private javax.swing.JPanel mainPanel;
    private javax.swing.JMenuBar menuBar;
    // End of variables declaration
    private JDialog aboutBox;
}
```

\dk\jsh\cleaningrobotsimulator\ui\swing\AboutBox.java

```java
package dk.jsh.cleaningrobotsimulator.ui.swing;

import org.jdesktop.application.Action;

/**
 * About box dialog.
 * @author Jan S. Hansen
 */
public class AboutBox extends javax.swing.JDialog {

    /**
     * Constructor.
     * @param parent parent frame
     */
    public AboutBox(java.awt.Frame parent) {
        super(parent);
        initComponents();
        getRootPane().setDefaultButton(closeButton);
    }

    /**
     * Close about box action.
     */
    @Action
    public void closeAboutBox() {
        dispose();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        closeButton = new javax.swing.JButton();
        javax.swing.JLabel appTitleLabel = new javax.swing.JLabel();
        javax.swing.JLabel versionLabel = new javax.swing.JLabel();
        javax.swing.JLabel appVersionLabel = new javax.swing.JLabel();
        javax.swing.JLabel vendorLabel = new javax.swing.JLabel();
        javax.swing.JLabel appVendorLabel = new javax.swing.JLabel();
        javax.swing.JLabel homepageLabel = new javax.swing.JLabel();
        javax.swing.JLabel appHomepageLabel = new javax.swing.JLabel();
        javax.swing.JLabel appDescLabel = new javax.swing.JLabel();
        javax.swing.JLabel imageLabel = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        org.jdesktop.application.ResourceMap resourceMap = 
org.jdesktop.application.Application.getInstance(dk.jsh.cleaningrobotsimulator.ui.swing.CleaningRobotSimulator.class).getCont
ext().getResourceMap(AboutBox.class);
        setTitle(resourceMap.getString("title")); // NOI18N
        setModal(true);
        setName("aboutBox"); // NOI18N
        setResizable(false);

        javax.swing.ActionMap actionMap = 
org.jdesktop.application.Application.getInstance(dk.jsh.cleaningrobotsimulator.ui.swing.CleaningRobotSimulator.class).getCont
ext().getActionMap(AboutBox.class, this);
        closeButton.setAction(actionMap.get("closeAboutBox")); // NOI18N
        closeButton.setName("closeButton"); // NOI18N

        appTitleLabel.setFont(appTitleLabel.getFont().deriveFont(appTitleLabel.getFont().getStyle() | java.awt.Font.BOLD, 
appTitleLabel.getFont().getSize()+4));
        appTitleLabel.setText(resourceMap.getString("Application.title")); // NOI18N
        appTitleLabel.setName("appTitleLabel"); // NOI18N

        versionLabel.setFont(versionLabel.getFont().deriveFont(versionLabel.getFont().getStyle() | java.awt.Font.BOLD));
        versionLabel.setText(resourceMap.getString("versionLabel.text")); // NOI18N
        versionLabel.setName("versionLabel"); // NOI18N

        appVersionLabel.setText(resourceMap.getString("Application.version")); // NOI18N
        appVersionLabel.setName("appVersionLabel"); // NOI18N

        vendorLabel.setFont(vendorLabel.getFont().deriveFont(vendorLabel.getFont().getStyle() | java.awt.Font.BOLD));
        vendorLabel.setText(resourceMap.getString("vendorLabel.text")); // NOI18N
        vendorLabel.setName("vendorLabel"); // NOI18N

        appVendorLabel.setText(resourceMap.getString("Application.vendor")); // NOI18N
        appVendorLabel.setName("appVendorLabel"); // NOI18N

        homepageLabel.setFont(homepageLabel.getFont().deriveFont(homepageLabel.getFont().getStyle() | java.awt.Font.BOLD));
        homepageLabel.setText(resourceMap.getString("homepageLabel.text")); // NOI18N
        homepageLabel.setName("homepageLabel"); // NOI18N

        appHomepageLabel.setText(resourceMap.getString("Application.homepage")); // NOI18N
        appHomepageLabel.setName("appHomepageLabel"); // NOI18N

        appDescLabel.setText(resourceMap.getString("appDescLabel.text")); // NOI18N
        appDescLabel.setName("appDescLabel"); // NOI18N

        imageLabel.setIcon(resourceMap.getIcon("imageLabel.icon")); // NOI18N
        imageLabel.setName("imageLabel"); // NOI18N

        org.jdesktop.layout.GroupLayout layout = new org.jdesktop.layout.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(layout.createSequentialGroup()
                .add(imageLabel)
                .add(18, 18, 18)
                .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING)
                    .add(org.jdesktop.layout.GroupLayout.LEADING, layout.createSequentialGroup()
                        .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                            .add(versionLabel)
                            .add(vendorLabel)
                            .add(homepageLabel))
                        .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                        .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                            .add(appVersionLabel)
                            .add(appVendorLabel)
                            .add(appHomepageLabel)))
                    .add(org.jdesktop.layout.GroupLayout.LEADING, appTitleLabel)
                    .add(org.jdesktop.layout.GroupLayout.LEADING, appDescLabel, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, 346, Short.MAX_VALUE)
                    .add(closeButton))
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(imageLabel, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .add(layout.createSequentialGroup()
                .addContainerGap()
                .add(appTitleLabel)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(appDescLabel, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(versionLabel)
                    .add(appVersionLabel))
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(vendorLabel)
                    .add(appVendorLabel))
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)
                .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)
                    .add(homepageLabel)
                    .add(appHomepageLabel))
                .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED, 33, Short.MAX_VALUE)
                .add(closeButton)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>
```

```java

    // Variables declaration - do not modify
    private javax.swing.JButton closeButton;
    // End of variables declaration

}

```

\dk\jsh\cleaningrobotsimulator\ui\swing\SimpleMainThreadExceptionHandler.java

```java
package dk.jsh.cleaningrobotsimulator.ui.swing;

import java.io.PrintWriter;
import java.io.StringWriter;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
import javax.swing.SwingUtilities;

/**
 * Main thread uncaught exception handler.
 * @author Jan S. Hansen
 */
public class SimpleMainThreadExceptionHandler
        implements Thread.UncaughtExceptionHandler {
    private Logger logger;

    /**
     * Constructor.
     */
    public SimpleMainThreadExceptionHandler() {
        logger = Logger.getLogger(getClass().getName());
    }

    /**
     * Log uncaugth exceptions to a log file and show an error dialog.
     * @param thread The thread that throw the exception
     * @param exception Exception to log.
     */
    @Override
    public void uncaughtException(final Thread thread,
            final Throwable exception) {
       if (SwingUtilities.isEventDispatchThread()) {
            showAndLogException(thread, exception);
        } else {
            SwingUtilities.invokeLater(new Runnable() {
                @Override
                public void run() {
                    showAndLogException(thread, exception);
                }
            });
        }
    }

   /**
    * Log exception in log file and show an error dialog.
    * @param thread The thread that throw the exception
    * @param exception Exception to log.
    */
   private void showAndLogException(Thread thread, Throwable exception) {
            exception.printStackTrace();
            StringWriter sw = new StringWriter();
            exception.printStackTrace(new PrintWriter(sw));
            logger.log(Level.SEVERE, "Uncaught exception in main thread",
                    sw.toString());
            JOptionPane.showMessageDialog(null,
                "An unexpected error occured, see log file.",
                "Cleaning robot simulator error",
                JOptionPane.ERROR_MESSAGE);
            System.exit(1);
   }
}

```

8.2. INDHOLD PÅ VEDLAGTE CD

Indholdet på den vedlagte CD er inddelt i følgende 3 kataloger:

- Løsning – Indeholder javakode, diverse resourcer samt Maven projektfil. Lavet vha. SubVersions eksportfunktion.
- Program – Indeholder en cleaning-robot-simulator-1.0.jar.
- Rapport – Indeholder denne rapport i Word 2007 format og i PDF format.