IT-DIPLOMUDDANNELSEN

Operativsystemer og Procesinteraktion

Eksamensprojekt

af

Jan Schrøder Hansen

FORÅR 2010

**INDHOLD**

[1. Indledning 3](#_Toc261709844)

[2. Opgavebeskrivelse 3](#_Toc261709845)

[2.1. Afgrænsing 3](#_Toc261709846)

[3. Lidt om trådet programmering i JAVA 4](#_Toc261709847)

[4. Design 4](#_Toc261709848)

[4.1. Indeling af kode i lag 4](#_Toc261709849)

[4.2. Design klassediagram 5](#_Toc261709850)

[5. Programmering 5](#_Toc261709851)

[6. Brugervejledning 6](#_Toc261709852)

[6.1. Fejlhåndtering 7](#_Toc261709853)

[7. Konklusion 8](#_Toc261709854)

[8. Bilag 9](#_Toc261709855)

[8.1. Kode 9](#_Toc261709856)

[8.2. Indhold på vedlagte CD 12](#_Toc261709857)

# 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.

# Opgavebeskrivelse

For at komme igennem noget af materialet i faget, har jeg valgt at lave et rengøringsrobot simulerings program.

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

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

Robotterne skal så 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, så kan de 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, så vælges et tilfældigt felt af disse. Ellers vælges der et tilfældigt rent felt. Hvor robot husker også de sidste 5 felter den har besøgt, disse felter undgås når der skal vælges nyt felt. Med mindre at dette ”låser” robotten. Robotten kan låse sig selv inde i et hjørne v.h.a denne felt ”hukommelse”, hvis dette sker så nulstilles denne felt hukommelse.

Hver robot og snavs generatoren, som jo 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 i 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 helt autonome, men hele tiden ”spørger” boardet.

Selve programmet laves i sproget Java[[1]](#footnote-2).

## 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 fulde robotter og venter på at komme til skraldespanden. Dette kunne dog 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.

# Lidt om trådet programmering i JAVA

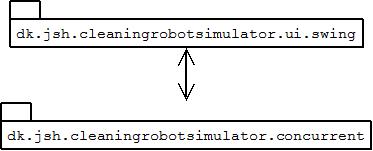
TODO

# Design

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

## Indeling af kode i lag

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



Figur - Pakkediagram

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

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

## Design klassediagram

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 Robot og DustCreator kan startes som tråde. I applikationen startes der 3 Robot tråde og en DustCreator tråd. BaseThread indeholder også et Board, som så benyttes af Robot og DustCreator.

Boardet består af 10x10 Field’s. Det er i selve Board klassen alt synkronisering sker, idet metoder emptyRobot, tryCleanField, tryMakeFieldDirty og tryMove alle er synkroniseret.

Board metode getReadOnlyField benyttes af Robot til at undesøge felter omkring sig. Denne er ikke synkroniseret, det er derfor at den i stedet for at retunere Field’s, så retuneres der ReadOnlyField’s. Dvs. at en Robot tråd, ikke kan være sikker på den status, som står i et ReadOnlyField. Det er også derfor de 3 Board metoder tryCleanField, tryMakeFieldDirty og tryMove alle returnere en boolean, som er true hvis operationen lykkes elles false hvis ikke.

# Programmering

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

* NetBeans[[2]](#footnote-3) – Java IDE, Java udviklingsmiljø.
* Maven2[[3]](#footnote-4) - Build, deploy, dependency management tool
* SubVersion[[4]](#footnote-5) - 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[[5]](#footnote-6). Mod at man frigiver koden som ”open source[[6]](#footnote-7)”. Hele dette projekt, inkl. denne tekst kan også findes under GoogleCode, se [code.google.com/p/cleaning-robot-simulator/](http://code.google.com/p/cleaning-robot-simulator/).

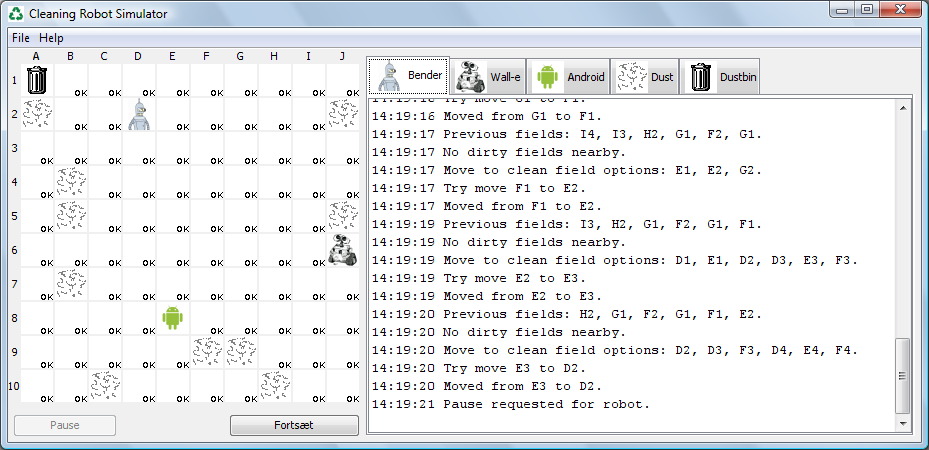
Alle diagrammer er lavet vha. Dia[[7]](#footnote-8), som har skabeloner til UML diagrammer. De enkelte diagrammer ligger også på den vedlagte CD. Se [bilag](#_Indhold_på_vedlagte) for indholdet på den vedlagte CD.

# Brugervejledning

Når programmet startes, fremkommer følgende dialog:

Log område, med en fane for hver robot, samt Dust (creator) og skraldespanden.

Selve boardet



Knapper til at pause/starte alle robot tråde.

Figur 3 - Selve programmet

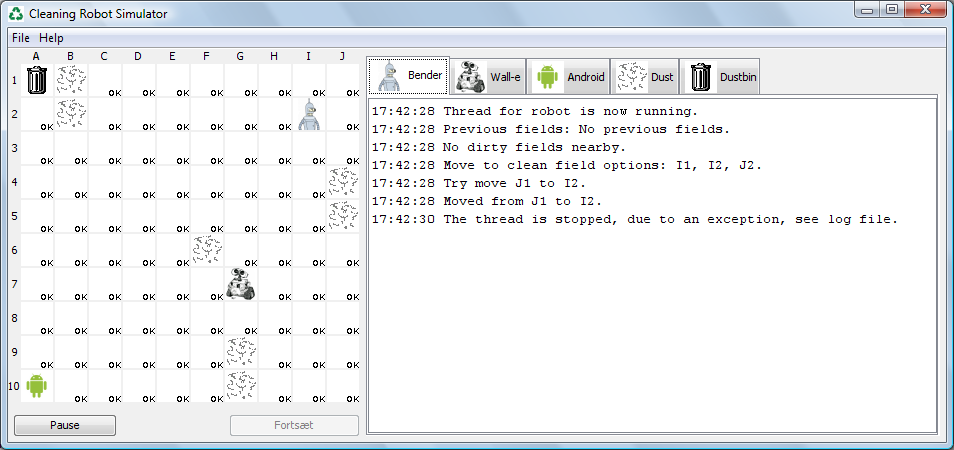
Symboler på boardet.

|  |  |
| --- | --- |
| Robotter | C:\GoogleCode\cleaning-robot-simulator\src\main\resources\dk\jsh\cleaningrobotsimulator\ui\swing\resources\icons\Android32x32.PNGC:\GoogleCode\cleaning-robot-simulator\src\main\resources\dk\jsh\cleaningrobotsimulator\ui\swing\resources\icons\Bender32x32.PNGC:\GoogleCode\cleaning-robot-simulator\src\main\resources\dk\jsh\cleaningrobotsimulator\ui\swing\resources\icons\Wall-e32x32.PNG Robotterne får en rød kant når du er fulde. Ex. C:\GoogleCode\cleaning-robot-simulator\src\main\resources\dk\jsh\cleaningrobotsimulator\ui\swing\resources\icons\Android32x32-full.png |
| Felter | Rent felt C:\GoogleCode\cleaning-robot-simulator\src\main\resources\dk\jsh\cleaningrobotsimulator\ui\swing\resources\icons\clean32x32.PNG, snavset felt C:\GoogleCode\cleaning-robot-simulator\src\main\resources\dk\jsh\cleaningrobotsimulator\ui\swing\resources\icons\dirt32x32.PNG |
| Skraldespand | C:\GoogleCode\cleaning-robot-simulator\src\main\resources\dk\jsh\cleaningrobotsimulator\ui\swing\resources\icons\dustbin32x32.png som ikke er i brug. C:\GoogleCode\cleaning-robot-simulator\src\main\resources\dk\jsh\cleaningrobotsimulator\ui\swing\resources\icons\recycle32x32.PNG skraldespand som benyttes af en robot. |

De to knapper ”Pause” og ”Forsæt” kan benyttes til at stoppe/genstarte alle Robotterne, så man evt. kan nærlæse de enkelte log faneblade.

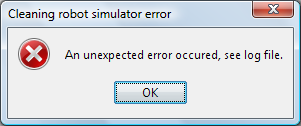
## Fejlhåndtering

Hvis der opstår en fejl i en af i en af Robotterne, vil det fremgå af den pågældende robots log fane. Se følgende ex.



Figur 4 - Excempel på en exception i en tråd

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



Figur 5- Applikations fejl

Når det trykkes på ”OK” knappen, så lukkes programmet.

Begge ovenstående fejlsituationer henviser til en log fil. Her kan en 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

# Konklusion

TODO:

# Bilag

## Kode

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

#### dk.jsh.cleaningrobotsimulator.concurent

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

1 package dk.jsh.cleaningrobotsimulator.concurrent;

2

3 import java.util.Date;

4 import javax.swing.Icon;

5 import javax.swing.ImageIcon;

6 import javax.swing.JTextArea;

7 import org.jdesktop.application.ResourceMap;

8

9 /\*\*

10 \* A Board class. A board consist of 10x10 fields

11 \* (see {@link dk.jsh.cleaningrobotsimulator.concurrent.Field Field}).<br>

12 \* Each field can be either clean or dirty (one is the dustbin).

13 \* A field can only hold one robot.<br>

14 \* This class is thread safe.

15 \* @see dk.jsh.cleaningrobotsimulator.cuncurrent.Field

16 \* @author Jan S. Hansen

17 \*/

18 public class Board {

19

20 //Thread safety - following fields is guarded by "this".

21 private Field[][] board;

22 private int dirtyFieldsCounter;

23 private long fieldsCleaned;

24 private JTextArea jTextAreaDustbin;

25

26 //Read-only fields.

27 private ResourceMap resourceMap;

28

29 /\*\*

30 \* Constructor.

31 \*/

32 public Board(ResourceMap resourceMap, JTextArea jTextAreaDustbin) {

33 this.resourceMap = resourceMap;

34 this.jTextAreaDustbin = jTextAreaDustbin;

35 board = new Field[Constants.MAX\_ROWS][Constants.MAX\_COLUMNS];

36 //Clean board

37 for (int row = 0; row < Constants.MAX\_ROWS; row++) {

38 for (int column = 0; column < Constants.MAX\_COLUMNS; column++) {

39 Field field = new Field(column, row,

40 Field.Status.CLEAN, Field.UsedBy.EMPTY);

41 Icon icon = resourceMap.getIcon("RobotSimulator.clean");

42 field.jLabel.setIcon(icon);

43 board[row][column] = field;

44 }

45 }

46 setField(9, 0, Field.Status.CLEAN, Field.UsedBy.BENDER,

47 "RobotSimulator.bender");

48 setField(9, 9, Field.Status.CLEAN, Field.UsedBy.WALL\_E,

49 "RobotSimulator.wall-e");

50 setField(0, 9, Field.Status.CLEAN, Field.UsedBy.ANDROID,

51 "RobotSimulator.android");

52 setField(0, 0, Field.Status.DUSTBIN, Field.UsedBy.EMPTY,

53 "RobotSimulator.dustbin");

54 }

55

56 /\*\*

57 \* Try to move a robot from one field to another field.

58 \* @param fromColumn from column

59 \* @param fromRow from row

60 \* @param toColumn to column

61 \* @param toRow to row

62 \* @param robotIconResource robot icon resource

63 \* @return true if move was a success.

64 \*/

65 public boolean tryMove(int fromColumn, int fromRow,

66 int toColumn, int toRow, String robotIconResource) {

67 testFieldArguments(fromColumn, fromRow);

68 testFieldArguments(toColumn, toRow);

69 boolean moveOk = false;

70 synchronized (this) {

71 Field fromField = getField(fromColumn, fromRow);

72 Field toField = getField(toColumn, toRow);

73 if (toField.isEmpty() && !fromField.isEmpty()) {

74 toField.setUsedBy(fromField.getUsedBy());

75 fromField.setUsedBy(Field.UsedBy.EMPTY);

76 moveOk = true;

77 //Set icons

78 if (fromColumn == 0 && fromRow == 0) {

79 fromField.jLabel.setIcon(

80 resourceMap.getIcon("RobotSimulator.dustbin"));

81 } else {

82 if (fromField.isDirty()) {

83 fromField.jLabel.setIcon(

84 resourceMap.getIcon("RobotSimulator.dirt"));

85 } else {

86 fromField.jLabel.setIcon(

87 resourceMap.getIcon("RobotSimulator.clean"));

88 }

89 }

90 if (toRow == 0 && toColumn == 0) {

91 toField.jLabel.setIcon(resourceMap.getIcon(

92 "RobotSimulator.recycle"));

93 } else {

94 toField.jLabel.setIcon(resourceMap.getIcon(

95 robotIconResource));

96 }

97 }

98 return moveOk;

99 }

100 }

101

102 /\*\*

103 \* Try to make a field dirty.

104 \* @param column fields column

105 \* @param row fields row

106 \* @return true if it was a success.

107 \*/

108 public boolean tryMakeFieldDirty(int column, int row) {

109 testFieldArguments(column, row);

110 boolean ok = false;

111 synchronized (this) {

112 if (dirtyFieldsCounter + 1 <= Constants.MAX\_DIRTY\_FIELDS) {

113 if (column == 0 && row == 0) { //Dustbin

114 throw new IllegalArgumentException("Dustbin can't be dirty");

115 }

116 Field field = getField(column, row);

117 if (field.isEmpty() && !field.isDirty()) {

118 field.setStatus(Field.Status.DIRTY);

119 dirtyFieldsCounter++;

120 ok = true;

121 field.jLabel.setIcon(

122 resourceMap.getIcon("RobotSimulator.dirt"));

123 }

124 }

125 return ok;

126 }

127 }

128

129 /\*\*

130 \* Changes a fields status to clean.

131 \* @param column fields column

132 \* @param row fields row

133 \* @return true if it was a success.

134 \*/

135 public boolean tryCleanField(int column, int row) {

136 boolean ok = false;

137 testFieldArguments(column, row);

138 synchronized (this) {

139 if (column == 0 && row == 0) { //Dustbin

140 throw new IllegalArgumentException("Dustbin can't be cleaned");

141 }

142 Field field = getField(column, row);

143 if (field.isDirty()) {

144 field.setStatus(Field.Status.CLEAN);

145 dirtyFieldsCounter--;

146 ok = true;

147 }

148 return ok;

149 }

150 }

151

152 /\*\*

153 \* Empties a robot for dust and log a message to the Dustbin log.

154 \* @param robotName robot name, used in log message.

155 \*/

156 public synchronized void emptyRobot(String robotName) {

157 fieldsCleaned+=Constants.MAX\_CLEANED\_FIELDS;

158 //Clear textArea after 2000 lines. TODO: Create a FIFO JTextArea

159 if (jTextAreaDustbin.getLineCount() > 2000) {

160 jTextAreaDustbin.setText("");

161 }

162

163 StringBuilder timeAndMessage =

164 new StringBuilder(Constants.timeFormat.format(new Date()));

165 timeAndMessage.append(" Dust from ").append(robotName);

166 timeAndMessage.append(" recieved - Total recieved: ");

167 timeAndMessage.append(fieldsCleaned).append(".\n");

168 jTextAreaDustbin.append(timeAndMessage.toString());

169 }

170

171 /\*\*

172 \* Returns dirty fields counter.

173 \* @return dirty fields counter

174 \*/

175 public synchronized int getDirtyFieldsCounter() {

176 return dirtyFieldsCounter;

177 }

178

179 /\*\*

180 \* Returns a read only field. To prevent that a field is updated outside

181 \* this board instance.

182 \* @param column fields column

183 \* @param row fields row

184 \* @return field a ReadOnlyField

185 \*/

186 public ReadOnlyField getReadOnlyField(int column, int row) {

187 testFieldArguments(column, row);

188 synchronized (this) {

189 return board[row][column];

190 }

191 }

192

193 /\*\*

194 \* Returns a field.

195 \* @param column fields column

196 \* @param row fields row

197 \* @return field a Field

198 \*/

199 private Field getField(int column, int row) {

200 return board[row][column];

201 }

202

203 /\*\*

204 \* Set a Fields Status and UsedBy.

205 \* @param column Fields column

206 \* @param row Fields row

207 \* @param status Fields Status

208 \* @param usedBy Fields UsedBy

209 \* @param iconResource Icon resource

210 \*/

211 private void setField(int column, int row, Field.Status status,

212 Field.UsedBy usedBy, String iconResource) {

213 testFieldArguments(column, row);

214 Field field = board[row][column];

215 field.setStatus(status);

216 field.setUsedBy(usedBy);

217 ImageIcon imageIcon = resourceMap.getImageIcon(iconResource);

218 field.jLabel.setIcon(imageIcon);

219 }

220

221 /\*\*

222 \* Test if is is valid column and row arguments.

223 \* @param column column

224 \* @param row row

225 \* @throws IllegalArgumentException Illegal row or column.

226 \*/

227 private void testFieldArguments(int column, int row)

228 throws IllegalArgumentException {

229 if (column < 0 || column >= Constants.MAX\_COLUMNS

230 || row < 0 || row >= Constants.MAX\_ROWS) {

231 throw new IllegalArgumentException("Error in column or row: ("

232 + column + ", " + row + ")");

233 }

234 }

235 }

236

## Indhold på vedlagte CD

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

* Løsning – Indeholder java kode, htmlfiler m.m samt NetBeans projektfil.
* Program – Indeholder en bari.war samt JavaDB skemafil til oprettelse af databasen.
* Rapport – Indeholder denne rapport i Word 2007 format og diagrammer i Dia-format.

1. Java, se [www.java.com](http://www.java.com) [↑](#footnote-ref-2)
2. NetBeans IDE, se [netbeans.org](http://netbeans.org/) [↑](#footnote-ref-3)
3. Maven2, se [maven.apache.org/](http://maven.apache.org/) [↑](#footnote-ref-4)
4. SubVersion, se [subversion.tigris.org](http://subversion.tigris.org/) [↑](#footnote-ref-5)
5. GoogleCode, se [code.google.com](http://code.google.com) [↑](#footnote-ref-6)
6. Open source, se [da.wikipedia.org/wiki/Open\_source](http://da.wikipedia.org/wiki/Open_source) [↑](#footnote-ref-7)
7. Dia, se [projects.gnome.org/dia](http://projects.gnome.org/dia/) [↑](#footnote-ref-8)