IT-DIPLOMUDDANNELSEN

Operativsystemer og Procesinteraktion

Eksamensprojekt

af

Jan Schrøder Hansen

FORÅR 2010

**INDHOLD**

[1. Indledning 3](#_Toc262909246)

[2. Opgavebeskrivelse 3](#_Toc262909247)

[2.1. Afgrænsing 4](#_Toc262909248)

[3. Design 4](#_Toc262909249)

[3.1. Indeling af kode i lag 4](#_Toc262909250)

[3.2. Design klassediagram 5](#_Toc262909251)

[4. Lidt om trådet programmering i JAVA 6](#_Toc262909252)

[5. Programmeringsmiljø 7](#_Toc262909253)

[6. Brugervejledning 8](#_Toc262909254)

[6.1. Fejlhåndtering 9](#_Toc262909255)

[7. Konklusion 10](#_Toc262909256)

[8. Bilag 11](#_Toc262909257)

[8.1. Kode 11](#_Toc262909258)

[8.2. Indhold på vedlagte CD 37](#_Toc262909259)

# 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, specielt kapitel 5 (Concurrency: Mutual Exclusion and Synchronization) og til dels kapitel 6 (Concurrency: Deadlock and Starvation), har jeg valgt at lave et rengøringsrobot simulerings program. Som visuelt viser hvordan robotter rengøre en flade, 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 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 6 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.

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]](#footnote-2) og gør brug af Java’s muligheder inde for trådprogrammering.

## 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.

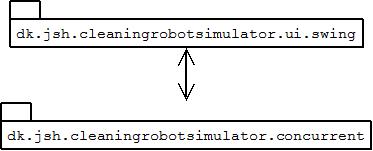
Løsningen er heller ikke dækket ind med hensyn til unit tests.

# 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 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 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 returnerer 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 nogle set metoder, det er for at sikre at en Robot tråd ikke ændre et Fields status.

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 returnere en boolean, som er true (sand) hvis operationen lykkes eller false hvis ikke.

F.eks. en robot henter alle 8 felter omkring sig (ReadOnlyField), 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. retunere false, fordi feltet er overtaget af en anden robot i mellemtiden.

# Lidt om trådet programmering i JAVA

En thread 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, så 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 vha. at lave et nyt objekt af klassen 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 et 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 metode kaldet, 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 kritisk sektion. Men det er faktisk en kort notation for at synchronize(this). Se følgende:

public boolean tryMove(..) {

//Kode som ikke er synkroniseret

synchronize(this) {

//Så længe en tråd er i denne kritiske sektion, kan andre tråde ikke benytte kode i dette //objekt, som også synkronisere på this.

}

}

Det samme kunne opnås ved at skrive public synchronized boolean tryMove(..){..}

”this” i dette tilfælde er bare objektet selv. Man kan også synkronisere på andre objekter.

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]](#footnote-3), kaldes dette også for ”Java monitor pattern”

Ud over det jeg har benyttet i programmet, så tilbyder java mange andre muligheder i forbindelse med tråd programmering. 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, som kan benyttes, som låse mekanismer. Dvs. at man kan lave noget programlogik hvor en tråd ”låser” en semaphor, og andre tråde må så 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 program filer, og det kan derfor være svært at vedligeholde, og fejlfinde i kode, der benytter mange semaphore.

# Programmeringsmiljø

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

* NetBeans[[3]](#footnote-4) – Java IDE, Java udviklingsmiljø.
* Maven2[[4]](#footnote-5) - Build, deploy, dependency management tool
* SubVersion[[5]](#footnote-6) - 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]](#footnote-7). Mod at man frigiver koden som open source[[7]](#footnote-8). 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[[8]](#footnote-9), 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.

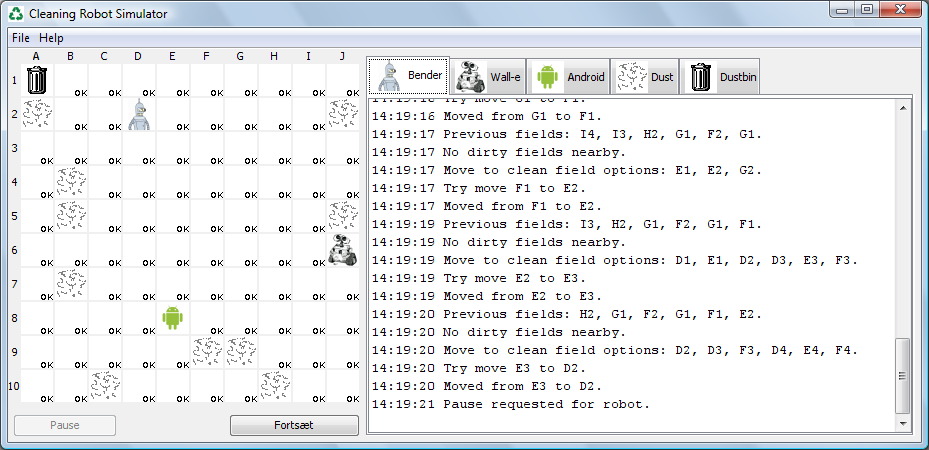
# Brugervejledning

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 dobbeltklippe 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 på den pågælden maskine.

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. |

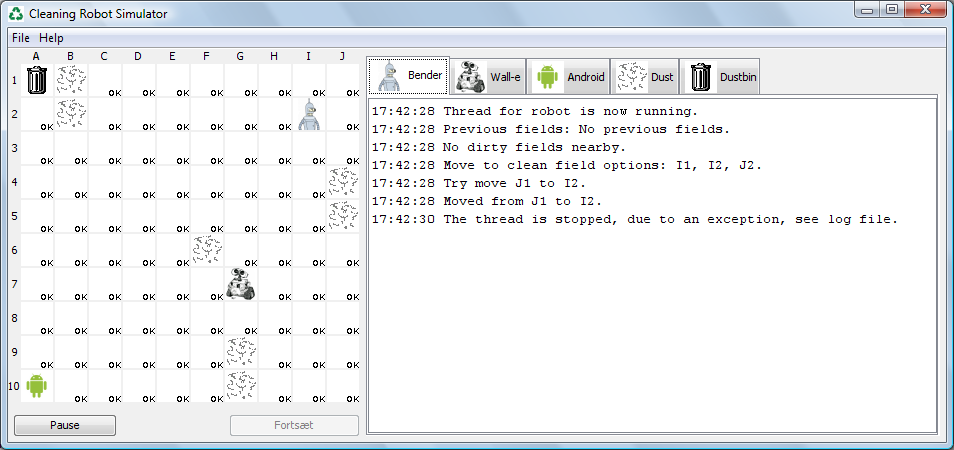
De to knapper ”Pause” og ”Forsæt” kan benyttes til at stoppe/genstarte alle Robotterne, så man evt. kan nærlæse de enkelte log faneblade.

Hver robot har et logfaneblad, hvor robottens status kan læses, 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, så vælges der et tilfældigt rent felt, som den ikke har besøgt inden for 6 træk. Når en robot har rengjort 5 felter, så er den fuld, og går til skaldepanden.

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.

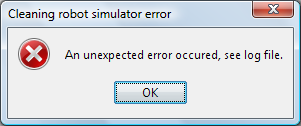
## Fejlhåndtering

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

I 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

Formålet med opgaven var at lave en rengørings robot simulator i java, som visulet viser hvordan 3 robotter holder et areal, bestående af 10 gange 10 felter rent. Jeg har måske brugt lidt meget 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 interagere med arealet (Boardet). Som nævt tidliger at det ikke et ægte simuleringsprogram, fordi de enkelte robotter ikke er autonome. Det er Boardet 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 tråd programmering, samtidighed, mutual exclusion, semaphore og monitors.

Jeg er ikke løbet ind i de store problemer med 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 lybsk. Og deadlock undgås ved at der kun er en robot som kan står på et felt af gangen. Hvis der var mere en 3 robotter kunne en robot eksempelvis, blive låst inde i et hjørne. men det betragter jeg ikke som en deadlock situation, da denne robot så bare må vente til de andre robotter har flyttet sig. Så som det også fremgår af bogen, så er designet vigtigt når man udvikler programmer, med flere tråde, kaldet ”Deadlock prevention” i bogen.

# 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\Constants.java |

1 package dk.jsh.cleaningrobotsimulator.concurrent;

2

3 import java.text.SimpleDateFormat;

4

5 /\*\*

6 \* Cleaning robot simulator constants.

7 \* @author Jan S. Hansen

8 \*/

9 public class Constants {

10

11 public final static int MAX\_ROWS = 10;

12 public final static int MAX\_COLUMNS = 10;

13 public final static int MAX\_DIRTY\_FIELDS = 10;

14 public final static int MAX\_CLEANED\_FIELDS = 5;

15 public final static SimpleDateFormat timeFormat =

16 new SimpleDateFormat("HH:mm:ss");

17

18 //Empty private constructor to prevent that this class can be instantiated.

19 private Constants() {

20 }

21 }

22

|  |
| --- |
| \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

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

1 /\*

2 \* To change this template, choose Tools | Templates

3 \* and open the template in the editor.

4 \*/

5

6 package dk.jsh.cleaningrobotsimulator.concurrent;

7

8 import javax.swing.JLabel;

9

10 /\*\*

11 \* A read only Field interface. Used by Board, so it can return a read only

12 \* field.

13 \* @author Jan S. Hansen

14 \*/

15 public interface ReadOnlyField {

16 public enum Status {CLEAN, DIRTY, DUSTBIN}

17 public enum UsedBy {BENDER, WALL\_E, ANDROID, EMPTY}

18

19 public int getColumn();

20 public int getRow();

21 public Status getStatus();

22 public UsedBy getUsedBy();

23 public boolean isEmpty();

24 public boolean isDirty();

25 public JLabel getLabel();

26 }

27

28

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

1 package dk.jsh.cleaningrobotsimulator.concurrent;

2

3 import javax.swing.JLabel;

4

5 /\*\*

6 \* Field value object.<br>

7 \* A field has a row and a column, both starting from 0.<br>

8 \* A field has a status that can be CLEAR, DIRTY or DUSTBIN, and <br>

9 \* a field can be used by either robot BENDER, WALL-E of ANDROID or else the

10 \* field is EMPTY.

11 \* @author Jan S. Hansen

12 \*/

13 public class Field implements ReadOnlyField {

14

15 private Status status;

16 private UsedBy usedBy;

17 private int column;

18 private int row;

19 public JLabel jLabel;

20

21 /\*\*

22 \* Constructor.

23 \* @param column Fields column

24 \* @param row Fields row

25 \* @param status Fields Status

26 \* @param usedBy Fields UsedBy

27 \*/

28 public Field(int column, int row, Status status, UsedBy usedBy) {

29 this.column = column;

30 this.row = row;

31 this.status = status;

32 this.usedBy = usedBy;

33 this.jLabel = new JLabel();

34 }

35

36 /\*\*

37 \* Gets Fields column.

38 \* @return column number

39 \*/

40 @Override

41 public int getColumn() {

42 return column;

43 }

44

45 /\*\*

46 \* Gets Fields row.

47 \* @return row number

48 \*/

49 @Override

50 public int getRow() {

51 return row;

52 }

53

54 /\*\*

55 \* Gets Fields Status.

56 \* @return Status

57 \*/

58 @Override

59 public Status getStatus() {

60 return status;

61 }

62

63 /\*\*

64 \* Sets Fields Status

65 \* @param status Status

66 \*/

67 public void setStatus(Status status) {

68 this.status = status;

69 }

70

71 /\*\*

72 \* Gets Fields UsedBt

73 \* @return UsedBy

74 \*/

75 @Override

76 public UsedBy getUsedBy() {

77 return usedBy;

78 }

79

80 /\*\*

81 \* Sets Fields UsedBy

82 \* @param usedBy UsedBy

83 \*/

84 public void setUsedBy(UsedBy usedBy) {

85 this.usedBy = usedBy;

86 }

87

88 /\*\*

89 \* Returns true if Field is empty.

90 \* @return true if empty

91 \*/

92 @Override

93 public boolean isEmpty() {

94 return usedBy == UsedBy.EMPTY;

95 }

96

97 /\*\*

98 \* Returns true if Field is dirty.

99 \* @return true if dirty

100 \*/

101 @Override

102 public boolean isDirty() {

103 return status == Status.DIRTY;

104 }

105

106 /\*\*

107 \* Returns JLabel.

108 \* @return JLabel

109 \*/

110 @Override

111 public JLabel getLabel() {

112 return jLabel;

113 }

114

115 /\*\*

116 \* Test if this field is equal to a given object.</br>

117 \* Row and column is tested.

118 \* @param obj object to Test

119 \* @return true if equal.

120 \*/

121 @Override

122 public boolean equals(Object obj) {

123 if (obj == null) {

124 return false;

125 }

126 if (getClass() != obj.getClass()) {

127 return false;

128 }

129 final Field other = (Field) obj;

130 if (this.column != other.column) {

131 return false;

132 }

133 if (this.row != other.row) {

134 return false;

135 }

136 return true;

137 }

138

139 @Override

140 public int hashCode() {

141 int hash = 5;

142 hash = 61 \* hash + this.column;

143 hash = 61 \* hash + this.row;

144 return hash;

145 }

146 }

147

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

1 package dk.jsh.cleaningrobotsimulator.concurrent;

2

3 import java.util.Date;

4 import java.util.logging.Logger;

5 import javax.swing.JTextArea;

6 import org.jdesktop.application.ResourceMap;

7

8 /\*\*

9 \* Abstract class with common thread functions

10 \* @author Jan S. Hansen

11 \*/

12 abstract public class BaseThread extends Thread {

13 protected JTextArea jTextArea;

14 protected Board board;

15 protected ResourceMap resourceMap;

16 protected Logger exceptionLogger; //Logging of exceptions in a log file.

17

18 /\*\*

19 \* Constructor.

20 \* @param threadName thread name

21 \* @param board A Board object

22 \* @param jTextArea A JTextArea to use as log for this thread

23 \* @param resourceMap A ResourceMap

24 \*/

25 public BaseThread(String threadName, Board board, JTextArea jTextArea,

26 ResourceMap resourceMap) {

27 this.board = board;

28 this.jTextArea = jTextArea;

29 this.resourceMap = resourceMap;

30 //If an exceptions occurs, the this name will be part of the exception

31 //stacktrace.

32 this.setName(threadName);

33 exceptionLogger = Logger.getLogger(Robot.class.getName());

34 setUncaughtExceptionHandler(new SimpleThreadExceptionHandler());

35 }

36

37 /\*\*

38 \* Log a message a the JTestArea. See constructor.

39 \* @param message message to log.

40 \*/

41 protected void log(String message) {

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

43 if (jTextArea.getLineCount() > 2000) {

44 jTextArea.setText("");

45 }

46

47 StringBuilder timeAndMessage =

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

49 timeAndMessage.append(" ").append(message).append("\n");

50 jTextArea.append(timeAndMessage.toString());

51 }

52

53 /\*\*

54 \* Log that an exception has occured in the thread.

55 \*/

56 protected void logException() {

57 log("The thread is stopped, due to an exception, see log file.");

58 }

59 }

60

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

1 package dk.jsh.cleaningrobotsimulator.concurrent;

2

3 import java.util.ArrayList;

4 import java.util.Date;

5 import java.util.List;

6 import java.util.Random;

7 import java.util.logging.Level;

8 import javax.swing.JTextArea;

9 import org.jdesktop.application.ResourceMap;

10

11 /\*\*

12 \* Robot thread.

13 \* @author Jan S. Hansen

14 \*/

15 public class Robot extends BaseThread {

16

17 private boolean stopRequested = false;

18 private boolean pauseRequested = false;

19 private String resource;

20 private String fullResource;

21 private int column;

22 private int row;

23 private ReadOnlyField[] prevFields =

24 new ReadOnlyField[]{null, null, null, null, null, null};

25 private int nextPrevField;

26 private int fieldsCleaned;

27 Random randomGenerator = new Random();

28

29 /\*\*

30 \* Constructor.

31 \* @param threadName Thread name

32 \* @param board A Board object

33 \* @param jTextArea A JTextArea to use as log for this thread

34 \* @param resourceMap A ResourceMap

35 \* @param resource Robots normal icon resource

36 \* @param fullResource Robots full icon resource

37 \* @param row Robots start row position

38 \* @param column Robots start column position

39 \*/

40 public Robot(String threadName, Board board, JTextArea jTextArea,

41 ResourceMap resourceMap,

42 String resource, String fullResource,

43 int row, int column) {

44 super(threadName, board, jTextArea, resourceMap);

45 this.resource = resource;

46 this.fullResource = fullResource;

47 this.column = column;

48 this.row = row;

49 this.resourceMap = resourceMap;

50 }

51

52 /\*\*

53 \* The threads run method.

54 \*/

55 @Override

56 public void run() {

57 log("Thread for robot is now running.");

58 while (!isStopRequested()) {

59 if (isPauseRequested()) {

60 paused();

61 } else {

62 cleaning();

63 }

64 }

65 log("Thread for robot is now stopped");

66 }

67

68 /\*\*

69 \* Robot is in cleaning mode

70 \*/

71 private void cleaning() {

72 logPrevFields();

73 addToPrevFields(board.getReadOnlyField(column, row));

74 if (fieldsCleaned >= Constants.MAX\_CLEANED\_FIELDS) { //Goto bin

75 gotoDustbinMode();

76 } else { //Search and clean

77 cleaningMode();

78 }

79 sleepForSecs(1);

80 }

81

82 /\*\*

83 \* Search for dirty nearby fields. If found clean a random

84 \* dirty field, else goto randon a nearby empty and clean field.

85 \*/

86 private void cleaningMode() {

87 //Search and clean

88 ReadOnlyField moveToField = getNextField();

89 if (moveToField == null) {

90 clearPrevFields();

91 } else {

92 int toColumn = moveToField.getColumn();

93 int toRow = moveToField.getRow();

94 logMove("Try move", row, column, toRow, toColumn);

95 if (board.tryMove(column, row, toColumn, toRow, resource)) {

96 logMove("Moved from", row, column, toRow, toColumn);

97 if (moveToField.isDirty()) {

98 if (board.tryCleanField(toColumn, toRow)) {

99 fieldsCleaned++;

100 log("Number of fields cleaned: " + fieldsCleaned + ".");

101 if (fieldsCleaned >= Constants.MAX\_CLEANED\_FIELDS) {

102 log("Robot is full.");

103 }

104 } else {

105 log("\*\*\* The field is no longer dirty, after moving " + "robot.");

106 }

107 }

108 row = toRow;

109 column = toColumn;

110 } else {

111 log("\*\*\* Move failed.");

112 }

113 }

114 }

115

116 /\*\*

117 \* Move robot closer to the dustbin. If robot is on the dustbin field, the

118 \* robot is emptied.

119 \*/

120 private void gotoDustbinMode() {

121 //Goto bin

122 int toRow = row > 0 ? row - 1 : 0;

123 int toColumn = column > 0 ? column - 1 : 0;

124 if (board.tryMove(column, row, toColumn, toRow, fullResource)) {

125 logMove("Move to dustbin", row, column, toRow, toColumn);

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

127 fieldsCleaned = 0;

128 board.emptyRobot(this.getName());

129 clearPrevFields();

130 log("Robot is emptied.");

131 }

132 row = toRow;

133 column = toColumn;

134 } else {

135 log("\*\*\* Move to dustbin failed.");

136 }

137 }

138

139 /\*\*

140 \* Paused this thread for 1 second.

141 \*/

142 private void paused() {

143 sleepForSecs(1);

144 }

145

146 /\*\*

147 \* Makes this thread goto sleep for a given number of seconds.

148 \* @param secs seconds

149 \*/

150 private void sleepForSecs(int secs) {

151 try {

152 int msecs = secs \* 1000;

153 int i = 0;

154 while (i < (msecs / 100) && !isStopRequested()) {

155 sleep(100);

156 i++;

157 }

158 } catch (InterruptedException ex) {

159 exceptionLogger.log(Level.SEVERE, null, ex);

160 logException();

161 requestStop();

162 }

163 }

164

165 /\*\*

166 \* Request this thread to stop

167 \*/

168 public synchronized void requestStop() {

169 log("Stop requested for robot.");

170 stopRequested = true;

171 }

172

173 /\*\*'

174 \* Returns true if this thread is requested to stop.

175 \* @return true if this thread is requested to stop

176 \*/

177 private synchronized boolean isStopRequested() {

178 return stopRequested;

179 }

180

181 /\*\*

182 \* Request this thread to go into pause mode.

183 \*/

184 public synchronized void requestPause() {

185 log("Pause requested for robot.");

186 pauseRequested = true;

187 }

188

189 /\*\*

190 \* Request this thread to go into running mode.

191 \*/

192 public synchronized void continueAfterPause() {

193 log("Continue requested for robot.");

194 pauseRequested = false;

195 }

196

197 /\*\*

198 \* Returns true if this thread is requested to go into pause mode.

199 \* @return true if this thread is requested to go into pause mode

200 \*/

201 private synchronized boolean isPauseRequested() {

202 return pauseRequested;

203 }

204

205 /\*\*

206 \* Returns the next field the Robot should try to go to. Dirty Fields has

207 \* priority.

208 \* @return A Field or null if no move is possible

209 \*/

210 private ReadOnlyField getNextField() {

211 List<ReadOnlyField> moveToCleanFieldOptions =

212 new ArrayList<ReadOnlyField>();

213 List<ReadOnlyField> moveToDirtyFieldOptions =

214 new ArrayList<ReadOnlyField>();

215 //Test fields above

216 int testColumn = column - 1;

217 int testRow = row - 1;

218 for (testColumn = column - 1; testColumn <= column + 1; testColumn++) {

219 if (validRowColumn(testColumn, testRow)) {

220 ReadOnlyField field = board.getReadOnlyField(testColumn, testRow);

221 if (field.isEmpty()) {

222 if (field.isDirty()) {

223 moveToDirtyFieldOptions.add(field);

224 } else {

225 if(!isFieldInPrevFields(field)) {

226 moveToCleanFieldOptions.add(field);

227 }

228 }

229 }

230 }

231 }

232 //Test field to the left

233 testRow = row;

234 testColumn = column - 1;

235 if (validRowColumn(testColumn, testRow)) {

236 ReadOnlyField field = board.getReadOnlyField(testColumn, testRow);

237 if (field.isEmpty()) {

238 if (field.isDirty()) {

239 moveToDirtyFieldOptions.add(field);

240 } else {

241 if(!isFieldInPrevFields(field)) {

242 moveToCleanFieldOptions.add(field);

243 }

244 }

245 }

246 }

247 //Test field to the right

248 testColumn = column + 1;

249 if (validRowColumn(testColumn, testRow)) {

250 ReadOnlyField field = board.getReadOnlyField(testColumn, testRow);

251 if (field.isEmpty()) {

252 if (field.isDirty()) {

253 moveToDirtyFieldOptions.add(field);

254 } else {

255 if(!isFieldInPrevFields(field)) {

256 moveToCleanFieldOptions.add(field);

257 }

258 }

259 }

260 }

261 //Test fields below

262 testColumn = column - 1;

263 testRow = row + 1;

264 for (testColumn = column - 1; testColumn <= column + 1; testColumn++) {

265 if (validRowColumn(testColumn, testRow)) {

266 ReadOnlyField field =

267 board.getReadOnlyField(testColumn, testRow);

268 if (field.isEmpty()) {

269 if (field.isDirty()) {

270 moveToDirtyFieldOptions.add(field);

271 } else {

272 if(!isFieldInPrevFields(field)) {

273 moveToCleanFieldOptions.add(field);

274 }

275 }

276 }

277 }

278 }

279 ReadOnlyField field = null;

280 if (!moveToDirtyFieldOptions.isEmpty()) {

281 logMoveToOptions("Move to dirty field options",

282 moveToDirtyFieldOptions);

283 //Return random

284 int index = randomGenerator.nextInt(moveToDirtyFieldOptions.size());

285 field = moveToDirtyFieldOptions.get(index);

286 } else { //No dirty fields to move to, try clean fields.

287 log("No dirty fields nearby.");

288 if (!moveToCleanFieldOptions.isEmpty()) {

289 logMoveToOptions("Move to clean field options",

290 moveToCleanFieldOptions);

291 int index = randomGenerator.nextInt(

292 moveToCleanFieldOptions.size());

293 field = moveToCleanFieldOptions.get(index);

294 } else {

295 log("\*\*\* Robot is locked, no move is possible!");

296 }

297 }

298 return field;

299 }

300

301 /\*\*

302 \* Test if a given column and row is valid.

303 \* @param column Column

304 \* @param row Row

305 \* @return true if valid pair of column and row

306 \*/

307 private boolean validRowColumn(int column, int row) {

308 boolean ok = true;

309 if (row < 0 || row >= Constants.MAX\_ROWS

310 || column < 0 || column >= Constants.MAX\_COLUMNS) {

311 ok = false;

312 }

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

314 ok = false;

315 }

316 return ok;

317 }

318

319 /\*\*

320 \* Add a Field to a circular buffer with previous fields this Robot has

321 \* visited.

322 \* @param field ReadOnlyField to add to buffer

323 \*/

324 private void addToPrevFields(ReadOnlyField field) {

325 prevFields[nextPrevField] = field;

326 nextPrevField++;

327 if (nextPrevField > prevFields.length - 1) {

328 nextPrevField = 0;

329 }

330 }

331

332 /\*\*

333 \* Clear a circular buffer with previous fields this Robot has

334 \* visited.

335 \*/

336 private void clearPrevFields() {

337 log("Clear prev. fields.");

338 for (int i = 0; i < prevFields.length; i++) {

339 prevFields[i] = null;

340 }

341 nextPrevField = 0;

342 }

343

344 /\*\*

345 \* Returns true if this field is in the circular buffer with previous

346 \* fields.

347 \* @param field ReadOnlyField to test

348 \* @return true if this field is in the circular buffer with previous

349 \* fields.

350 \*/

351 private boolean isFieldInPrevFields(ReadOnlyField field) {

352 int i = 0;

353 boolean fieldFound = false;

354 while (!fieldFound && i < prevFields.length) {

355 if (field.equals(prevFields[i])) {

356 fieldFound = true;

357 } else {

358 i++;

359 }

360 }

361 return fieldFound;

362 }

363

364 /\*\*

365 \* Log a move.

366 \* @param message Message before from and to text.

367 \* @param fromRow from row

368 \* @param fromColumn from column

369 \* @param toRow to row

370 \* @param toColumn to column

371 \*/

372 private void logMove(String message,

373 int fromRow, int fromColumn,

374 int toRow, int toColumn) {

375 StringBuilder timeAndMessage =

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

377 timeAndMessage.append(" ").append(message).append(" ");

378 timeAndMessage.append((char) (fromColumn + 65));

379 timeAndMessage.append(++fromRow).append(" to ");

380 timeAndMessage.append((char) (toColumn + 65));

381 timeAndMessage.append(++toRow).append(".\n");

382 jTextArea.append(timeAndMessage.toString());

383 }

384

385 /\*\*

386 \* Log all move to options.

387 \* @param message Message before options

388 \* @param fields A List of Fields

389 \*/

390 private void logMoveToOptions(String message, List<ReadOnlyField> fields) {

391 StringBuilder timeAndMessage =

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

393 timeAndMessage.append(" ").append(message);

394 String before = ": ";

395 for (ReadOnlyField field : fields) {

396 timeAndMessage.append(before);

397 timeAndMessage.append((char) (field.getColumn() + 65));

398 timeAndMessage.append(field.getRow() + 1);

399 before = ", ";

400 }

401 timeAndMessage.append(".\n");

402 jTextArea.append(timeAndMessage.toString());

403 }

404

405 /\*\*

406 \* Log prev. fields.

407 \*/

408 private void logPrevFields() {

409 StringBuilder timeAndMessage =

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

411 timeAndMessage.append(" Previous fields: ");

412 String before = null;

413 int i = nextPrevField;

414 boolean noPrevFields = true;

415 for (int c = 0; c < prevFields.length; c++) {

416 ReadOnlyField field = prevFields[i];

417 if (field != null) {

418 noPrevFields = false;

419 if (before != null) {

420 timeAndMessage.append(before);

421 }

422 timeAndMessage.append((char) (field.getColumn() + 65));

423 timeAndMessage.append(field.getRow() + 1);

424 before = ", ";

425 }

426 i++;

427 if (i > prevFields.length - 1) {

428 i = 0;

429 }

430 }

431 if (noPrevFields) {

432 timeAndMessage.append("No previous fields");

433 }

434 timeAndMessage.append(".\n");

435 jTextArea.append(timeAndMessage.toString());

436 }

437 }

438

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

1 package dk.jsh.cleaningrobotsimulator.concurrent;

2

3 import java.util.Date;

4 import java.util.Random;

5 import java.util.logging.Level;

6 import javax.swing.JTextArea;

7 import org.jdesktop.application.ResourceMap;

8

9 /\*\*

10 \* Dust creator thread.

11 \* @author Jan S. Hansen

12 \*/

13 public class DustCreator extends BaseThread {

14 Random randomGenerator = new Random();

15

16 /\*\*

17 \* Constructor.

18 \* @param threadName Thread name

19 \* @param board A Board object

20 \* @param jTextArea A JTextArea to use as log for this thread.

21 \* @param resourceMap A ResourceMap

22 \*/

23 public DustCreator(String threadName, Board board, JTextArea jTextArea,

24 ResourceMap resourceMap) {

25 super(threadName, board, jTextArea, resourceMap);

26 }

27

28 /\*\*

29 \* The threads run method.

30 \*/

31 @Override

32 public void run() {

33 log("Thread for dust creator is now running.");

34 int dirtyFields = board.getDirtyFieldsCounter();

35 log("Dirty fields on board: " + dirtyFields);

36 while (dirtyFields < Constants.MAX\_DIRTY\_FIELDS) {

37 int row = randomGenerator.nextInt(Constants.MAX\_ROWS);

38 int column = randomGenerator.nextInt(Constants.MAX\_COLUMNS);

39 if (row != 0 || column != 0) { //Dustbin

40 logTrySetFieldDirty(row, column);

41 if (board.tryMakeFieldDirty(column, row)) {

42 dirtyFields++;

43 log("Dirt added.");

44 }

45 else {

46 log("Failed.");

47 }

48 }

49 sleepForSecs(1);

50 }

51 log("Thread for dust creator is now finished.");

52 }

53

54 /\*\*

55 \* Log a "Try put dirt on field" message.

56 \* @param row fields row, used in log message, converted to row + 1

57 \* @param column fields column, used in log message, converted to A, B, C

58 \* etc.

59 \*/

60 private void logTrySetFieldDirty(int row, int column) {

61 StringBuilder timeAndMessage =

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

63 timeAndMessage.append(" Try put dirt on field ");

64 timeAndMessage.append((char)(column + 65));

65 timeAndMessage.append(++row).append(".\n");;

66 jTextArea.append(timeAndMessage.toString());

67 }

68

69 /\*\*

70 \* Makes this thread goto sleep for a given number of seconds.

71 \* @param secs seconds

72 \*/

73 private void sleepForSecs(int secs) {

74 try {

75 sleep(secs \* 1000);

76 } catch (InterruptedException ex) {

77 exceptionLogger.log(Level.SEVERE, null, ex);

78 logException();

79 }

80 }

81 }

82

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

1 package dk.jsh.cleaningrobotsimulator.concurrent;

2

3 import java.io.PrintWriter;

4 import java.io.StringWriter;

5 import java.util.logging.Level;

6 import java.util.logging.Logger;

7

8 /\*\*

9 \* This class is used to handle uncaught exceptions in threads.

10 \* @author Jan S. Hansen

11 \*/

12 public class SimpleThreadExceptionHandler

13 implements Thread.UncaughtExceptionHandler {

14 private Logger logger;

15

16 /\*\*

17 \* Constructor.

18 \*/

19 public SimpleThreadExceptionHandler() {

20 logger = Logger.getLogger(getClass().getName());

21 }

22

23 /\*\*

24 \* Log uncaugth exceptions to a log file and to the standard error stream.

25 \* @param thread The thread that throw the exception

26 \* @param exception Exception.

27 \*/

28 @Override

29 public void uncaughtException(Thread thread, Throwable exception) {

30 exception.printStackTrace();

31 StringWriter sw = new StringWriter();

32 exception.printStackTrace(new PrintWriter(sw));

33 logger.log(Level.SEVERE, "Uncaught exception in thread",

34 thread.getName());

35 logger.log(Level.SEVERE, "Uncaught exception in thread", sw.toString());

36 if (thread instanceof Robot) {

37 Robot robot = (Robot)thread;

38 robot.logException();

39 }

40 }

41 }

42

#### dk.jsh.cleaningrobotsimulator.ui.swing

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

1 package dk.jsh.cleaningrobotsimulator.ui.swing;

2

3 import java.util.logging.FileHandler;

4 import java.util.logging.Handler;

5 import java.util.logging.Level;

6 import java.util.logging.Logger;

7 import org.jdesktop.application.Application;

8 import org.jdesktop.application.SingleFrameApplication;

9

10 /\*\*

11 \* The main class of the application.

12 \* @author Jan S. Hansen

13 \*/

14 public class CleaningRobotSimulator extends SingleFrameApplication {

15

16 /\*\*

17 \* At startup create and show the main frame of the application.

18 \*/

19 @Override protected void startup() {

20 show(new View(this));

21 }

22

23 /\*\*

24 \* Setup log file.

25 \*/

26 private static void setupLog() {

27 try {

28 //%t - Means that the log is located in the Systems Temp directory

29 Handler fh = new FileHandler("%t/cleaning-robot-simulator.log",

30 10000, 5);

31 Logger logger = Logger.getLogger("");

32 logger.addHandler(fh);

33 logger.setLevel(Level.INFO);

34 logger.info("Application started.");

35 } catch (Exception ex) {

36 ex.printStackTrace();

37 }

38 }

39

40 /\*\*

41 \* This method is to initialize the specified window by injecting resources.

42 \* Windows shown in our application come fully initialized from the GUI

43 \* builder, so this additional configuration is not needed.

44 \*/

45 @Override protected void configureWindow(java.awt.Window root) {

46 }

47

48 /\*\*

49 \* A convenient static getter for the application instance.

50 \* @return the instance of CleaningRobotSimulator

51 \*/

52 public static CleaningRobotSimulator getApplication() {

53 return Application.getInstance(CleaningRobotSimulator.class);

54 }

55

56 /\*\*

57 \* Main method launching the application.

58 \*/

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

60 setupLog();

61 Thread.setDefaultUncaughtExceptionHandler(

62 new SimpleMainThreadExceptionHandler());

63 launch(CleaningRobotSimulator.class, args);

64 }

65 }

66

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

1 package dk.jsh.cleaningrobotsimulator.ui.swing;

2

3 import dk.jsh.cleaningrobotsimulator.concurrent.Board;

4 import dk.jsh.cleaningrobotsimulator.concurrent.Constants;

5 import dk.jsh.cleaningrobotsimulator.concurrent.DustCreator;

6 import dk.jsh.cleaningrobotsimulator.concurrent.ReadOnlyField;

7 import dk.jsh.cleaningrobotsimulator.concurrent.Robot;

8 import java.awt.GridBagConstraints;

9 import java.awt.Insets;

10 import java.awt.event.ComponentAdapter;

11 import java.awt.event.ComponentEvent;

12 import java.awt.event.WindowEvent;

13 import java.util.concurrent.Executors;

14 import java.util.concurrent.ScheduledExecutorService;

15 import java.util.concurrent.TimeUnit;

16 import java.util.logging.Level;

17 import java.util.logging.Logger;

18 import javax.swing.ImageIcon;

19 import org.jdesktop.application.Action;

20 import org.jdesktop.application.ResourceMap;

21 import org.jdesktop.application.SingleFrameApplication;

22 import org.jdesktop.application.FrameView;

23 import javax.swing.JDialog;

24 import javax.swing.JFrame;

25 import javax.swing.JLabel;

26

27 /\*\*

28 \* The application's main frame.

29 \* @author Jan S. Hansen

30 \*/

31 public class View extends FrameView {

32

33 private Board board;

34 private ResourceMap resourceMap;

35 private Robot bender;

36 private Robot android;

37 private Robot wallE;

38 private Logger logger;

39

40 /\*\*

41 \* Constructor.

42 \*/

43 public View(SingleFrameApplication app) {

44 super(app);

45 logger = Logger.getLogger(View.class.getName());

46

47 resourceMap = getResourceMap();

48

49 //Cacth windowClosing event

50 JFrame jFrame = this.getFrame();

51 jFrame.addWindowListener(new java.awt.event.WindowAdapter() {

52

53 @Override

54 public void windowClosing(WindowEvent winEvt) {

55 quit();

56 }

57 });

58

59 //Set icon in upper left corner

60 ImageIcon image = resourceMap.getImageIcon("RobotSimulator.recycle");

61 jFrame.setIconImage(image.getImage());

62

63 //Initialize UI

64 initComponents();

65

66 //Create board

67 board = new Board(resourceMap, jTextAreaDustbin);

68

69 //Set tab icons

70 jTabbedPane1.setIconAt(0, resourceMap.getIcon("RobotSimulator.bender"));

71 jTabbedPane1.setIconAt(1, resourceMap.getIcon("RobotSimulator.wall-e"));

72 jTabbedPane1.setIconAt(2, resourceMap.getIcon("RobotSimulator.android"));

73 jTabbedPane1.setIconAt(3, resourceMap.getIcon("RobotSimulator.dirt"));

74 jTabbedPane1.setIconAt(4, resourceMap.getIcon("RobotSimulator.dustbin"));

75 jTabbedPane1.setSelectedIndex(0);

76

77 jButtonContinue.setEnabled(false);

78

79 //Set JFrame's min. hight and width.

80 jFrame.addComponentListener(new ComponentAdapter() {

81 private final static int MIN\_WIDTH = 855;

82 private final static int MIN\_HIGHT = 450;

83 @Override

84 public void componentResized(ComponentEvent e) {

85 JFrame frame = (JFrame) e.getSource();

86 int width = frame.getWidth() < MIN\_WIDTH

87 ? MIN\_WIDTH : frame.getWidth();

88 int hight = frame.getHeight() < MIN\_HIGHT

89 ? MIN\_HIGHT : frame.getHeight();

90 frame.setSize(width, hight);

91 }

92 });

93

94 createUIBoard();

95

96 //Start robot threads

97 bender = new Robot("Bender", board, jTextAreaBender, resourceMap,

98 "RobotSimulator.bender", "RobotSimulator.bender-full", 0, 9);

99 bender.start();

100

101 android = new Robot("Android", board, jTextAreaAndroid, resourceMap,

102 "RobotSimulator.android", "RobotSimulator.android-full", 9, 0);

103 android.start();

104

105 wallE = new Robot("Wall-E", board, jTextAreaWallE, resourceMap,

106 "RobotSimulator.wall-e", "RobotSimulator.wall-e-full", 9, 9);

107 wallE.start();

108

109 //Get a scheduler

110 ScheduledExecutorService scheduler =

111 Executors.newSingleThreadScheduledExecutor();

112 //Run DustCreator with a 30 secs. delay between each run.

113 scheduler.scheduleWithFixedDelay(

114 new DustCreator("DustCreator", board, jTextAreaDust,

115 resourceMap), 0, 30, TimeUnit.SECONDS);

116 }

117

118 /\*\*

119 \* Adds fields from board to UI.

120 \*/

121 private void createUIBoard() {

122 GridBagConstraints gridBagConstraints = new java.awt.GridBagConstraints();

123 Insets insets = new Insets(1, 1, 1, 1);

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

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

126 ReadOnlyField field = board.getReadOnlyField(column, row);

127 JLabel jLabel = field.getLabel();

128 gridBagConstraints.gridx = column + 1;

129 gridBagConstraints.gridy = row + 1;

130 gridBagConstraints.insets = insets;

131 mainPanel.add(jLabel, gridBagConstraints);

132 }

133 }

134 }

135

136 /\*\*

137 \* Show about box action.

138 \*/

139 @Action

140 public void showAboutBox() {

141 if (aboutBox == null) {

142 JFrame mainFrame = CleaningRobotSimulator.getApplication().getMainFrame();

143 aboutBox = new AboutBox(mainFrame);

144 aboutBox.setLocationRelativeTo(mainFrame);

145 }

146 CleaningRobotSimulator.getApplication().show(aboutBox);

147 }

148

149 /\*\*

150 \* Pause button action.

151 \*/

152 @Action

153 public void pause() {

154 bender.requestPause();

155 android.requestPause();

156 wallE.requestPause();

157 jButtonPause.setEnabled(false);

158 jButtonContinue.setEnabled(true);

159 }

160

161 /\*\*

162 \* Continue button action.

163 \*/

164 @Action

165 public void cont() {

166 bender.continueAfterPause();

167 android.continueAfterPause();

168 wallE.continueAfterPause();

169 jButtonPause.setEnabled(true);

170 jButtonContinue.setEnabled(false);

171 }

172

173 /\*\*

174 \* Quit application action.

175 \*/

176 @Action

177 public void quit() {

178 bender.requestStop();

179 android.requestStop();

180 wallE.requestStop();

181 while (bender.isAlive() || android.isAlive() || wallE.isAlive()) {

182 try {

183 Thread.sleep(50);

184 } catch (InterruptedException ex) {

185 logger.log(Level.SEVERE, "Error waiting for robots to stop.",

186 ex);

187 }

188 }

189 logger.log(Level.INFO, "Application stopped.");

190 System.exit(0);

191 }

192

193 /\*\* This method is called from within the constructor to

194 \* initialize the form.

195 \* WARNING: Do NOT modify this code. The content of this method is

196 \* always regenerated by the Form Editor.

197 \*/

198 @SuppressWarnings("unchecked")

199 // <editor-fold defaultstate="collapsed" desc="Generated Code">

200 private void initComponents() {

201 java.awt.GridBagConstraints gridBagConstraints;

202

203 menuBar = new javax.swing.JMenuBar();

204 javax.swing.JMenu fileMenu = new javax.swing.JMenu();

205 javax.swing.JMenuItem exitMenuItem = new javax.swing.JMenuItem();

206 javax.swing.JMenu helpMenu = new javax.swing.JMenu();

207 javax.swing.JMenuItem aboutMenuItem = new javax.swing.JMenuItem();

208 mainPanel = new javax.swing.JPanel();

209 jLabel1 = new javax.swing.JLabel();

210 jLabel2 = new javax.swing.JLabel();

211 jLabel3 = new javax.swing.JLabel();

212 jLabel4 = new javax.swing.JLabel();

213 jLabel5 = new javax.swing.JLabel();

214 jLabel6 = new javax.swing.JLabel();

215 jLabel7 = new javax.swing.JLabel();

216 jLabel8 = new javax.swing.JLabel();

217 jLabel9 = new javax.swing.JLabel();

218 jLabel10 = new javax.swing.JLabel();

219 jLabel11 = new javax.swing.JLabel();

220 jLabel12 = new javax.swing.JLabel();

221 jLabel13 = new javax.swing.JLabel();

222 jLabel14 = new javax.swing.JLabel();

223 jLabel15 = new javax.swing.JLabel();

224 jLabel16 = new javax.swing.JLabel();

225 jLabel17 = new javax.swing.JLabel();

226 jLabel18 = new javax.swing.JLabel();

227 jLabel19 = new javax.swing.JLabel();

228 jLabel20 = new javax.swing.JLabel();

229 jLabel21 = new javax.swing.JLabel();

230 jTabbedPane1 = new javax.swing.JTabbedPane();

231 jScrollPane1 = new javax.swing.JScrollPane();

232 jTextAreaBender = new javax.swing.JTextArea();

233 jScrollPane2 = new javax.swing.JScrollPane();

234 jTextAreaWallE = new javax.swing.JTextArea();

235 jScrollPane3 = new javax.swing.JScrollPane();

236 jTextAreaAndroid = new javax.swing.JTextArea();

237 jScrollPane4 = new javax.swing.JScrollPane();

238 jTextAreaDust = new javax.swing.JTextArea();

239 jScrollPane5 = new javax.swing.JScrollPane();

240 jTextAreaDustbin = new javax.swing.JTextArea();

241 jButtonPause = new javax.swing.JButton();

242 jButtonContinue = new javax.swing.JButton();

243

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

245

246 org.jdesktop.application.ResourceMap resourceMap = org.jdesktop.application.Application.getInstance(dk.jsh.cleaningrobotsimulator.ui.swing.CleaningRobotSimulator.class).getContext().getResourceMap(View.class);

247 fileMenu.setText(resourceMap.getString("fileMenu.text")); // NOI18N

248 fileMenu.setName("fileMenu"); // NOI18N

249

250 javax.swing.ActionMap actionMap = org.jdesktop.application.Application.getInstance(dk.jsh.cleaningrobotsimulator.ui.swing.CleaningRobotSimulator.class).getContext().getActionMap(View.class, this);

251 exitMenuItem.setAction(actionMap.get("quit")); // NOI18N

252 exitMenuItem.setIcon(resourceMap.getIcon("exitMenuItem.icon")); // NOI18N

253 exitMenuItem.setName("exitMenuItem"); // NOI18N

254 fileMenu.add(exitMenuItem);

255

256 menuBar.add(fileMenu);

257

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

259 helpMenu.setName("helpMenu"); // NOI18N

260

261 aboutMenuItem.setAction(actionMap.get("showAboutBox")); // NOI18N

262 aboutMenuItem.setIcon(resourceMap.getIcon("aboutMenuItem.icon")); // NOI18N

263 aboutMenuItem.setDisabledIcon(resourceMap.getIcon("aboutMenuItem.disabledIcon")); // NOI18N

264 aboutMenuItem.setName("aboutMenuItem"); // NOI18N

265 helpMenu.add(aboutMenuItem);

266

267 menuBar.add(helpMenu);

268

269 mainPanel.setMinimumSize(new java.awt.Dimension(313, 240));

270 mainPanel.setName("mainPanel"); // NOI18N

271 mainPanel.setLayout(new java.awt.GridBagLayout());

272

273 jLabel1.setText(resourceMap.getString("jLabel1.text")); // NOI18N

274 jLabel1.setName("jLabel1"); // NOI18N

275 gridBagConstraints = new java.awt.GridBagConstraints();

276 gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;

277 mainPanel.add(jLabel1, gridBagConstraints);

278

279 jLabel2.setText(resourceMap.getString("jLabel2.text")); // NOI18N

280 jLabel2.setName("jLabel2"); // NOI18N

281 mainPanel.add(jLabel2, new java.awt.GridBagConstraints());

282

283 jLabel3.setText(resourceMap.getString("jLabel3.text")); // NOI18N

284 jLabel3.setName("jLabel3"); // NOI18N

285 mainPanel.add(jLabel3, new java.awt.GridBagConstraints());

286

287 jLabel4.setText(resourceMap.getString("jLabel4.text")); // NOI18N

288 jLabel4.setName("jLabel4"); // NOI18N

289 mainPanel.add(jLabel4, new java.awt.GridBagConstraints());

290

291 jLabel5.setText(resourceMap.getString("jLabel5.text")); // NOI18N

292 jLabel5.setName("jLabel5"); // NOI18N

293 mainPanel.add(jLabel5, new java.awt.GridBagConstraints());

294

295 jLabel6.setText(resourceMap.getString("jLabel6.text")); // NOI18N

296 jLabel6.setName("jLabel6"); // NOI18N

297 mainPanel.add(jLabel6, new java.awt.GridBagConstraints());

298

299 jLabel7.setText(resourceMap.getString("jLabel7.text")); // NOI18N

300 jLabel7.setName("jLabel7"); // NOI18N

301 mainPanel.add(jLabel7, new java.awt.GridBagConstraints());

302

303 jLabel8.setText(resourceMap.getString("jLabel8.text")); // NOI18N

304 jLabel8.setName("jLabel8"); // NOI18N

305 mainPanel.add(jLabel8, new java.awt.GridBagConstraints());

306

307 jLabel9.setText(resourceMap.getString("jLabel9.text")); // NOI18N

308 jLabel9.setName("jLabel9"); // NOI18N

309 mainPanel.add(jLabel9, new java.awt.GridBagConstraints());

310

311 jLabel10.setText(resourceMap.getString("jLabel10.text")); // NOI18N

312 jLabel10.setName("jLabel10"); // NOI18N

313 mainPanel.add(jLabel10, new java.awt.GridBagConstraints());

314

315 jLabel11.setText(resourceMap.getString("jLabel11.text")); // NOI18N

316 jLabel11.setName("jLabel11"); // NOI18N

317 mainPanel.add(jLabel11, new java.awt.GridBagConstraints());

318

319 jLabel12.setText(resourceMap.getString("jLabel12.text")); // NOI18N

320 jLabel12.setName("jLabel12"); // NOI18N

321 gridBagConstraints = new java.awt.GridBagConstraints();

322 gridBagConstraints.gridx = 0;

323 gridBagConstraints.gridy = 1;

324 mainPanel.add(jLabel12, gridBagConstraints);

325

326 jLabel13.setText(resourceMap.getString("jLabel13.text")); // NOI18N

327 jLabel13.setName("jLabel13"); // NOI18N

328 gridBagConstraints = new java.awt.GridBagConstraints();

329 gridBagConstraints.gridx = 0;

330 gridBagConstraints.gridy = 2;

331 mainPanel.add(jLabel13, gridBagConstraints);

332

333 jLabel14.setText(resourceMap.getString("jLabel14.text")); // NOI18N

334 jLabel14.setName("jLabel14"); // NOI18N

335 gridBagConstraints = new java.awt.GridBagConstraints();

336 gridBagConstraints.gridx = 0;

337 gridBagConstraints.gridy = 3;

338 mainPanel.add(jLabel14, gridBagConstraints);

339

340 jLabel15.setText(resourceMap.getString("jLabel15.text")); // NOI18N

341 jLabel15.setName("jLabel15"); // NOI18N

342 gridBagConstraints = new java.awt.GridBagConstraints();

343 gridBagConstraints.gridx = 0;

344 gridBagConstraints.gridy = 4;

345 mainPanel.add(jLabel15, gridBagConstraints);

346

347 jLabel16.setText(resourceMap.getString("jLabel16.text")); // NOI18N

348 jLabel16.setName("jLabel16"); // NOI18N

349 gridBagConstraints = new java.awt.GridBagConstraints();

350 gridBagConstraints.gridx = 0;

351 gridBagConstraints.gridy = 5;

352 mainPanel.add(jLabel16, gridBagConstraints);

353

354 jLabel17.setText(resourceMap.getString("jLabel17.text")); // NOI18N

355 jLabel17.setName("jLabel17"); // NOI18N

356 gridBagConstraints = new java.awt.GridBagConstraints();

357 gridBagConstraints.gridx = 0;

358 gridBagConstraints.gridy = 6;

359 mainPanel.add(jLabel17, gridBagConstraints);

360

361 jLabel18.setText(resourceMap.getString("jLabel18.text")); // NOI18N

362 jLabel18.setName("jLabel18"); // NOI18N

363 gridBagConstraints = new java.awt.GridBagConstraints();

364 gridBagConstraints.gridx = 0;

365 gridBagConstraints.gridy = 7;

366 mainPanel.add(jLabel18, gridBagConstraints);

367

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

369 jLabel19.setName("jLabel19"); // NOI18N

370 gridBagConstraints = new java.awt.GridBagConstraints();

371 gridBagConstraints.gridx = 0;

372 gridBagConstraints.gridy = 8;

373 mainPanel.add(jLabel19, gridBagConstraints);

374

375 jLabel20.setText(resourceMap.getString("jLabel20.text")); // NOI18N

376 jLabel20.setName("jLabel20"); // NOI18N

377 gridBagConstraints = new java.awt.GridBagConstraints();

378 gridBagConstraints.gridx = 0;

379 gridBagConstraints.gridy = 9;

380 mainPanel.add(jLabel20, gridBagConstraints);

381

382 jLabel21.setText(resourceMap.getString("jLabel21.text")); // NOI18N

383 jLabel21.setName("jLabel21"); // NOI18N

384 gridBagConstraints = new java.awt.GridBagConstraints();

385 gridBagConstraints.gridx = 0;

386 gridBagConstraints.gridy = 10;

387 mainPanel.add(jLabel21, gridBagConstraints);

388

389 jTabbedPane1.setBorder(javax.swing.BorderFactory.createEmptyBorder(1, 1, 1, 1));

390 jTabbedPane1.setName("jTabbedPane1"); // NOI18N

391

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

393

394 jTextAreaBender.setColumns(20);

395 jTextAreaBender.setEditable(false);

396 jTextAreaBender.setRows(5);

397 jTextAreaBender.setName("jTextAreaBender"); // NOI18N

398 jScrollPane1.setViewportView(jTextAreaBender);

399

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

401

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

403

404 jTextAreaWallE.setColumns(20);

405 jTextAreaWallE.setEditable(false);

406 jTextAreaWallE.setRows(5);

407 jTextAreaWallE.setName("jTextAreaWallE"); // NOI18N

408 jScrollPane2.setViewportView(jTextAreaWallE);

409

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

411

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

413

414 jTextAreaAndroid.setColumns(20);

415 jTextAreaAndroid.setEditable(false);

416 jTextAreaAndroid.setRows(5);

417 jTextAreaAndroid.setName("jTextAreaAndroid"); // NOI18N

418 jScrollPane3.setViewportView(jTextAreaAndroid);

419

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

421

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

423

424 jTextAreaDust.setColumns(20);

425 jTextAreaDust.setEditable(false);

426 jTextAreaDust.setRows(5);

427 jTextAreaDust.setName("jTextAreaDust"); // NOI18N

428 jScrollPane4.setViewportView(jTextAreaDust);

429

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

431

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

433

434 jTextAreaDustbin.setColumns(20);

435 jTextAreaDustbin.setEditable(false);

436 jTextAreaDustbin.setRows(5);

437 jTextAreaDustbin.setName("jTextAreaDustbin"); // NOI18N

438 jScrollPane5.setViewportView(jTextAreaDustbin);

439

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

441

442 gridBagConstraints = new java.awt.GridBagConstraints();

443 gridBagConstraints.gridwidth = 15;

444 gridBagConstraints.gridheight = 12;

445 gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;

446 gridBagConstraints.weightx = 1.0;

447 gridBagConstraints.weighty = 1.0;

448 gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);

449 mainPanel.add(jTabbedPane1, gridBagConstraints);

450

451 jButtonPause.setAction(actionMap.get("pause")); // NOI18N

452 jButtonPause.setText(resourceMap.getString("jButtonPause.text")); // NOI18N

453 jButtonPause.setName("jButtonPause"); // NOI18N

454 gridBagConstraints = new java.awt.GridBagConstraints();

455 gridBagConstraints.gridx = 0;

456 gridBagConstraints.gridy = 11;

457 gridBagConstraints.gridwidth = 4;

458 gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;

459 gridBagConstraints.anchor = java.awt.GridBagConstraints.SOUTH;

460 gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 5);

461 mainPanel.add(jButtonPause, gridBagConstraints);

462

463 jButtonContinue.setAction(actionMap.get("cont")); // NOI18N

464 jButtonContinue.setText(resourceMap.getString("jButtonContinue.text")); // NOI18N

465 jButtonContinue.setName("jButtonContinue"); // NOI18N

466 gridBagConstraints = new java.awt.GridBagConstraints();

467 gridBagConstraints.gridx = 7;

468 gridBagConstraints.gridy = 11;

469 gridBagConstraints.gridwidth = 4;

470 gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;

471 gridBagConstraints.anchor = java.awt.GridBagConstraints.SOUTH;

472 gridBagConstraints.insets = new java.awt.Insets(5, 5, 5, 0);

473 mainPanel.add(jButtonContinue, gridBagConstraints);

474

475 setComponent(mainPanel);

476 setMenuBar(menuBar);

477 }// </editor-fold>

478 // Variables declaration - do not modify

479 private javax.swing.JButton jButtonContinue;

480 private javax.swing.JButton jButtonPause;

481 private javax.swing.JLabel jLabel1;

482 private javax.swing.JLabel jLabel10;

483 private javax.swing.JLabel jLabel11;

484 private javax.swing.JLabel jLabel12;

485 private javax.swing.JLabel jLabel13;

486 private javax.swing.JLabel jLabel14;

487 private javax.swing.JLabel jLabel15;

488 private javax.swing.JLabel jLabel16;

489 private javax.swing.JLabel jLabel17;

490 private javax.swing.JLabel jLabel18;

491 private javax.swing.JLabel jLabel19;

492 private javax.swing.JLabel jLabel2;

493 private javax.swing.JLabel jLabel20;

494 private javax.swing.JLabel jLabel21;

495 private javax.swing.JLabel jLabel3;

496 private javax.swing.JLabel jLabel4;

497 private javax.swing.JLabel jLabel5;

498 private javax.swing.JLabel jLabel6;

499 private javax.swing.JLabel jLabel7;

500 private javax.swing.JLabel jLabel8;

501 private javax.swing.JLabel jLabel9;

502 private javax.swing.JScrollPane jScrollPane1;

503 private javax.swing.JScrollPane jScrollPane2;

504 private javax.swing.JScrollPane jScrollPane3;

505 private javax.swing.JScrollPane jScrollPane4;

506 private javax.swing.JScrollPane jScrollPane5;

507 private javax.swing.JTabbedPane jTabbedPane1;

508 private javax.swing.JTextArea jTextAreaAndroid;

509 private javax.swing.JTextArea jTextAreaBender;

510 private javax.swing.JTextArea jTextAreaDust;

511 private javax.swing.JTextArea jTextAreaDustbin;

512 private javax.swing.JTextArea jTextAreaWallE;

513 private javax.swing.JPanel mainPanel;

514 private javax.swing.JMenuBar menuBar;

515 // End of variables declaration

516 private JDialog aboutBox;

517 }

518

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

1 package dk.jsh.cleaningrobotsimulator.ui.swing;

2

3 import org.jdesktop.application.Action;

4

5 /\*\*

6 \* About box dialog.

7 \* @author Jan S. Hansen

8 \*/

9 public class AboutBox extends javax.swing.JDialog {

10

11 /\*\*

12 \* Constructor.

13 \* @param parent parent frame

14 \*/

15 public AboutBox(java.awt.Frame parent) {

16 super(parent);

17 initComponents();

18 getRootPane().setDefaultButton(closeButton);

19 }

20

21 /\*\*

22 \* Close about box action.

23 \*/

24 @Action

25 public void closeAboutBox() {

26 dispose();

27 }

28

29 /\*\* This method is called from within the constructor to

30 \* initialize the form.

31 \* WARNING: Do NOT modify this code. The content of this method is

32 \* always regenerated by the Form Editor.

33 \*/

34 // <editor-fold defaultstate="collapsed" desc="Generated Code">

35 private void initComponents() {

36

37 closeButton = new javax.swing.JButton();

38 javax.swing.JLabel appTitleLabel = new javax.swing.JLabel();

39 javax.swing.JLabel versionLabel = new javax.swing.JLabel();

40 javax.swing.JLabel appVersionLabel = new javax.swing.JLabel();

41 javax.swing.JLabel vendorLabel = new javax.swing.JLabel();

42 javax.swing.JLabel appVendorLabel = new javax.swing.JLabel();

43 javax.swing.JLabel homepageLabel = new javax.swing.JLabel();

44 javax.swing.JLabel appHomepageLabel = new javax.swing.JLabel();

45 javax.swing.JLabel appDescLabel = new javax.swing.JLabel();

46 javax.swing.JLabel imageLabel = new javax.swing.JLabel();

47

48 setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE\_ON\_CLOSE);

49 org.jdesktop.application.ResourceMap resourceMap = org.jdesktop.application.Application.getInstance(dk.jsh.cleaningrobotsimulator.ui.swing.CleaningRobotSimulator.class).getContext().getResourceMap(AboutBox.class);

50 setTitle(resourceMap.getString("title")); // NOI18N

51 setModal(true);

52 setName("aboutBox"); // NOI18N

53 setResizable(false);

54

55 javax.swing.ActionMap actionMap = org.jdesktop.application.Application.getInstance(dk.jsh.cleaningrobotsimulator.ui.swing.CleaningRobotSimulator.class).getContext().getActionMap(AboutBox.class, this);

56 closeButton.setAction(actionMap.get("closeAboutBox")); // NOI18N

57 closeButton.setName("closeButton"); // NOI18N

58

59 appTitleLabel.setFont(appTitleLabel.getFont().deriveFont(appTitleLabel.getFont().getStyle() | java.awt.Font.BOLD, appTitleLabel.getFont().getSize()+4));

60 appTitleLabel.setText(resourceMap.getString("Application.title")); // NOI18N

61 appTitleLabel.setName("appTitleLabel"); // NOI18N

62

63 versionLabel.setFont(versionLabel.getFont().deriveFont(versionLabel.getFont().getStyle() | java.awt.Font.BOLD));

64 versionLabel.setText(resourceMap.getString("versionLabel.text")); // NOI18N

65 versionLabel.setName("versionLabel"); // NOI18N

66

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

68 appVersionLabel.setName("appVersionLabel"); // NOI18N

69

70 vendorLabel.setFont(vendorLabel.getFont().deriveFont(vendorLabel.getFont().getStyle() | java.awt.Font.BOLD));

71 vendorLabel.setText(resourceMap.getString("vendorLabel.text")); // NOI18N

72 vendorLabel.setName("vendorLabel"); // NOI18N

73

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

75 appVendorLabel.setName("appVendorLabel"); // NOI18N

76

77 homepageLabel.setFont(homepageLabel.getFont().deriveFont(homepageLabel.getFont().getStyle() | java.awt.Font.BOLD));

78 homepageLabel.setText(resourceMap.getString("homepageLabel.text")); // NOI18N

79 homepageLabel.setName("homepageLabel"); // NOI18N

80

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

82 appHomepageLabel.setName("appHomepageLabel"); // NOI18N

83

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

85 appDescLabel.setName("appDescLabel"); // NOI18N

86

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

88 imageLabel.setName("imageLabel"); // NOI18N

89

90 org.jdesktop.layout.GroupLayout layout = new org.jdesktop.layout.GroupLayout(getContentPane());

91 getContentPane().setLayout(layout);

92 layout.setHorizontalGroup(

93 layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

94 .add(layout.createSequentialGroup()

95 .add(imageLabel)

96 .add(18, 18, 18)

97 .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING)

98 .add(org.jdesktop.layout.GroupLayout.LEADING, layout.createSequentialGroup()

99 .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

100 .add(versionLabel)

101 .add(vendorLabel)

102 .add(homepageLabel))

103 .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

104 .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

105 .add(appVersionLabel)

106 .add(appVendorLabel)

107 .add(appHomepageLabel)))

108 .add(org.jdesktop.layout.GroupLayout.LEADING, appTitleLabel)

109 .add(org.jdesktop.layout.GroupLayout.LEADING, appDescLabel, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, 346, Short.MAX\_VALUE)

110 .add(closeButton))

111 .addContainerGap())

112 );

113 layout.setVerticalGroup(

114 layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

115 .add(imageLabel, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

116 .add(layout.createSequentialGroup()

117 .addContainerGap()

118 .add(appTitleLabel)

119 .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

120 .add(appDescLabel, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE)

121 .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

122 .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)

123 .add(versionLabel)

124 .add(appVersionLabel))

125 .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

126 .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)

127 .add(vendorLabel)

128 .add(appVendorLabel))

129 .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

130 .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)

131 .add(homepageLabel)

132 .add(appHomepageLabel))

133 .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED, 33, Short.MAX\_VALUE)

134 .add(closeButton)

135 .addContainerGap())

136 );

137

138 pack();

139 }// </editor-fold>

140

141 // Variables declaration - do not modify

142 private javax.swing.JButton closeButton;

143 // End of variables declaration

144

145 }

146

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

1 package dk.jsh.cleaningrobotsimulator.ui.swing;

2

3 import java.io.PrintWriter;

4 import java.io.StringWriter;

5 import java.util.logging.Level;

6 import java.util.logging.Logger;

7 import javax.swing.JOptionPane;

8 import javax.swing.SwingUtilities;

9

10 /\*\*

11 \* Main thread uncaught exception handler.

12 \* @author Jan S. Hansen

13 \*/

14 public class SimpleMainThreadExceptionHandler

15 implements Thread.UncaughtExceptionHandler {

16 private Logger logger;

17

18 /\*\*

19 \* Constructor.

20 \*/

21 public SimpleMainThreadExceptionHandler() {

22 logger = Logger.getLogger(getClass().getName());

23 }

24

25 /\*\*

26 \* Log uncaugth exceptions to a log file and show an error dialog.

27 \* @param thread The thread that throw the exception

28 \* @param exception Exception to log.

29 \*/

30 @Override

31 public void uncaughtException(final Thread thread,

32 final Throwable exception) {

33 if (SwingUtilities.isEventDispatchThread()) {

34 showAndLogException(thread, exception);

35 } else {

36 SwingUtilities.invokeLater(new Runnable() {

37 @Override

38 public void run() {

39 showAndLogException(thread, exception);

40 }

41 });

42 }

43 }

44

45 /\*\*

46 \* Log exception in log file and show an error dialog.

47 \* @param thread The thread that throw the exception

48 \* @param exception Exception to log.

49 \*/

50 private void showAndLogException(Thread thread, Throwable exception) {

51 exception.printStackTrace();

52 StringWriter sw = new StringWriter();

53 exception.printStackTrace(new PrintWriter(sw));

54 logger.log(Level.SEVERE, "Uncaught exception in main thread",

55 sw.toString());

56 JOptionPane.showMessageDialog(null,

57 "An unexpected error occured, see log file.",

58 "Cleaning robot simulator error",

59 JOptionPane.ERROR\_MESSAGE);

60 System.exit(1);

61 }

62 }

63

## Indhold på vedlagte CD

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

* Løsning – Indeholder java kode, resourcer m.m samt Maven projektfil.
* Program – Indeholder en todo.exer 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. Java Concurrency in Practice, af Brian Goetz m.fl., se [www.javaconcurrencyinpractice.com](http://www.javaconcurrencyinpractice.com/) [↑](#footnote-ref-3)
3. NetBeans IDE, se [netbeans.org](http://netbeans.org/) [↑](#footnote-ref-4)
4. Maven2, se [maven.apache.org/](http://maven.apache.org/) [↑](#footnote-ref-5)
5. SubVersion, se [subversion.tigris.org](http://subversion.tigris.org/) [↑](#footnote-ref-6)
6. GoogleCode, se [code.google.com](http://code.google.com) [↑](#footnote-ref-7)
7. Open source, se [da.wikipedia.org/wiki/Open\_source](http://da.wikipedia.org/wiki/Open_source) [↑](#footnote-ref-8)
8. Dia, se [projects.gnome.org/dia](http://projects.gnome.org/dia/) [↑](#footnote-ref-9)