Concurrent Programming

Name: Danny Rene Jensen
D1
Supervisor: Peter Schneider-Kamp
Teaching assistant: Christian Damsgaard Jørgensen
DM519

12. april 2015

Indhold

1 Modelling

The first thing to do was to create an elevator design with seperate modules. The buttons to call the elevator was a good starting point. It would be easier to test if the elevator could move, as the first thing. To do that the elevator itself, with all the buttons, was needed. All the buttons are created the same way without any restrictions so the elevator can move freely and take as many agents in and even let more agents out than actually entered. The in and out buttons have a variable f on them, this variable is the current floor the eleavator is standing on. It is designed like this to prevent the agents to be able to exit on other floors than the elevator is on. the [g:FLOORS] on both the call buttons and the inside buttons are designed so the buttons for all floors are active and when, one of them is pressed the floors number is stored in g and given the next elevator runthrough, for the in and out buttons to use.

```
1 ELEVATOR = ELEVATOR[G],
2 ELEVATOR[f:FLOORS] =
3        ([f].in -> ELEVATOR[f] |
4        [f].out -> ELEVATOR[f] |
5        [g:FLOORS].callButton -> ELEVATOR[g] |
6        [r:FLOORS].insideButton -> ELEVATOR[r]).
```

After this the buttons on the elevator should work so the passengers could get in and out on all the floors. when that is possible the in and out funktions needs to be restricted to only letting 1 person in and out. it would be pretty bad if an unlimited number of people could get in but only 1 is allowed out again, and vice versa if only 1 enters and passengers just keeps exitting the elevator. (The elevator is for AREA 51 but it is doubtfull they want to clone their workers.) All these restrictions happens in AREA_ELEVATOR, and is based on the number of agents in the elevator. The assumption is taken that the elevator is not big enough to hold more than one agent at a time, that means CAPACITY is 1. When the elevator is emty i becomes 0. When i is smaller than CAPASITY two things happens here, first thing is all the in buttons can be activated this can't happen if i is 1. Second thing that happens is all the call buttons are active only when i is 0. Else if i is 1 the out buttons, inside buttons and the scan buttons can be active, of cause if they are resticted in other places all the criterias needs to be fullfiled for them to be active.

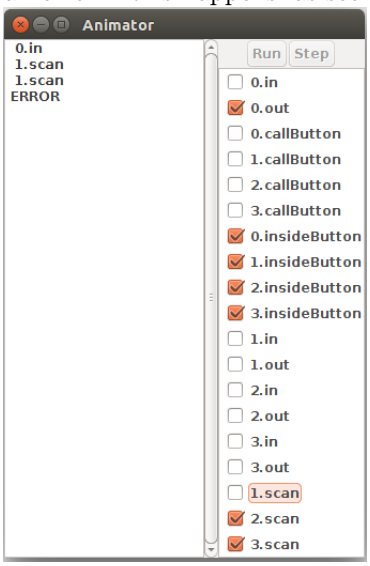
```
1 AREA_ELEVATOR = NORMAL[0],
2 NORMAL[i:NUMBER] = (when (i < CAPACITY) [G..T2].in -> NORMAL[i+1] |
3                when (i < CAPACITY) [FLOORS].callButton -> NORMAL[i] |
4                when (i == CAPACITY) [G..T2].out -> NORMAL[i-1]|
5                when (i == CAPACITY) [FLOORS].insideButton-> NORMAL[i] |
6                when (i == CAPACITY) [LEVEL].scan -> NORMAL[i]).
```

When the restrictions are done it still needs buttons on the inside of the elevator, these were created in the ELEVATOR. And of cause the retina scanner.
The retina scanner is set up so the passenger needs to scan his or her eye to even get the possibility to go to the other floors. the restriction on the out fuction, will only let the agent out on the floor that is less or equal to the permitted floors. The agent can't leave the elevator before he has gotten a clearence level.

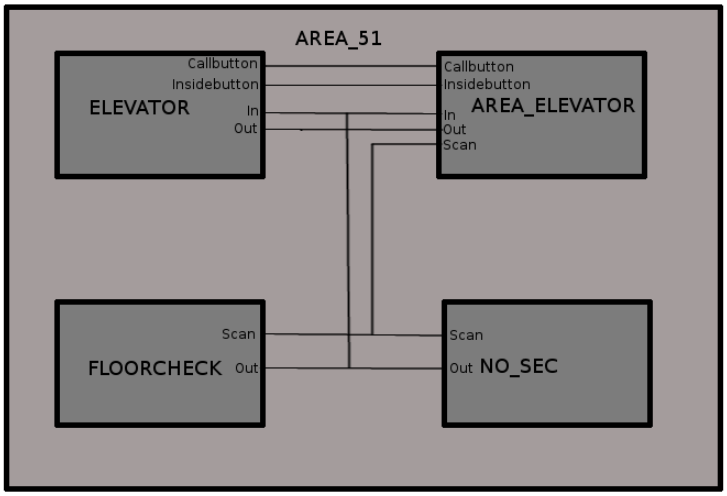
Here out is restricted yet again that means that both this and the previous reqirements needs to be fullfilled, to let an out button become active. Here the requirement is that the scan needs to be of a certain number or higher to activate the out button.

```
1 FLOORCHECK = FLOORCHECK[1],
2 FLOORCHECK[d:LEVEL] = ([h:LEVEL].scan -> SCAN[h]),
3 SCAN = SCAN[1],
4 SCAN[i:LEVEL] = (when (i > 0) [0].out -> FLOORCHECK[i] |
5                  when (i > 1) [1].out -> FLOORCHECK[i] |
6                  when (i > 2) [2..3].out -> FLOORCHECK[i]).
```

If both the AREA_ELEVATOR end FLOORCHECK requirements are put together with the ELEVATOR, then the out buttons can only be activated on the floor the elevator is on, only if there is an agent inside and he has the right securety clearence.
The securety property is just a copy of FLOORCHECK with the name altered to NO_SEC. Though the property lets the agent get scanned again, it creates an error if this happens. as seen below.



1.1 LTSA model diagram



2 Analysis

to prevent deadlocks the assumption is taken that the elevator is not big enough to hold more than one agent at a time. This assumtoin helps to avoid a deadlock where two agents enter the elevator that have different clearence levels. By only having one agent in the elevator at a time there are no chance of any resources being taken by another method or prgram, because there is only 1 agent prgram running at a time. Another deadlock could happen if an agent, or some other person, where to enter that does not have a clearence level at all. But the securety at AREA 51 should be so tight that no outsider would have access to the elevator, and even if he did the elevator wouldn't move. To avoid a deadlock where an agent whith a low clearnce level is trying to enter to high securety floor he is not allowed out but can ride the elevator to all the other floors until he is able to, and chooses, to leave.

3 Implementation

First off the elevator should be created so it could move with an agent inside. After this the elevator needs some restrictions, like only 1 agent would be allowed in the elevator at a time.
To make sure an agent with to low clearence level wouldn't be able to exit the wrong floor, TjeckClerance takes their level and returns the number on which floor they have the maximum clearnce for. Then in FloorCheck the max floor is used to check against the floor the agent wants to leave on, if it is lover or equal it return true, letting the agent know the doors are open and he is free to leave.

```
public int TjeckClerance(int level) {
	if (level == 1){
		return 0;
	}else if(level == 2) {
		return 1;
	}else {
		return 3;
	}
}
public boolean FloorCheck(int level, int floor) {
	if (TjeckClerance(level) >= floor) {
		return true;
	}else{
		return false;
	}
}
```

In this elevator there are 4 differnt agent threads, 2 are specefecly designed to move in a predifined pattern. The 3rd is a random agent he can spawn on any floor, have any clearence level and he also wants to go to a random floor. If he can't get out on that floor he tries another random floor until he gets out. This random agent is created by using the random numbergenerator functions in java.

```
public int randClerance() {
	Random rn = new Random();
	int num = rn.nextInt(3) + 1;
	return num;
}
```

The last agent is a user created agent, the user can define this agent with all the criteria.
The monitor a.k. GUI is starting the agents by clicking the buttons. The GUI can handle all 4 agents at a time. The elevator.java is just the one that sets restrictions to the agents. It looks if they can enter or not, this enter function has a random time before checking if the elevator is empty which prevents multiple agents to enter at a time. this random time is only 10 milliseconds but is eneough time to make the chacne of two or more agents enter at the same time allmost impossible.

```
public boolean EnterElevator() {
	Random rn = new Random();
	int time = rn.nextInt(10) + 1;
	try {
		Thread.sleep(time);
	} catch(InterruptedException ex) {
		Thread.currentThread().interrupt();
	}
	if (inElevator == 0){
		return true;
	}else{
		return false;
	}
}
```

The Elevator.java also stores all the info on the agents, like their clearence level, the start floor and the floor the agent wants to go to.
All agents has an if satament that looks if it is possible to exit the elevator on the desired floor, if not a new floor is selected.

```
if (!elevator.FloorCheck(elevator.level, elevator.floor)){
	elevator.SelectedFloor(0, 1);
}
```

The last int, in this case 1, in elevator.SelectedFloor is the umber on the agent and is only used to update the right image in the GUI. these images are there as a visual representation for the user to see what happens and would not be in a real elevator program. The elevator and floors are also represented by images and the elevator will also be updated as the agent chooses which floor he starts on and which floor he wants to leave. To keep track of the agent inside the elevator, e.g. which clearence level he has and if he is allowed to leave, there are three textfields in the upper right corner. These textfield shows, from top, the clearnce level, if the agent enters or exit if and he cant, e.g. if an agent with confidential clearence level tries to exit on secret floor 2, the experimental wepons floor, (where they are trying to create their prototype nr 42 which is the point of view gun) the label will say "no access". The last displays on which floor the elevator curently is.
The sleep function is also only for show. All it does is slow down the movements of the agents so it is possible to follow them on the screen. Without this sleep function the agents would move allmost instantly around and it would be impossible for any human to check if the program is working like it should or not.

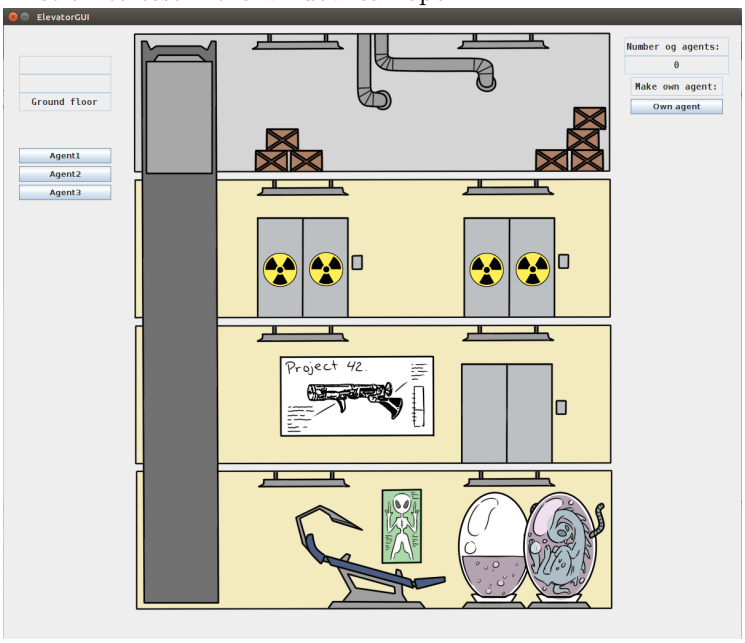
```
public void sleep(){
	try {
		Thread.sleep(1000);
	} catch(InterruptedException ex) {
		Thread.currentThread().interrupt();
	}
}
```

ElevatorRun in the GUI is the method that performes the movements and chechks of the agent, after he entered the elevator. It moves the agent and the elevator to the desired floor. After this it checks if the agent has the possibility to exit the elevator at that floor, if not it writes this to a label. The else statement is designed to run through, if the agent can exit at the desired floor. The agent is then moved out of the elevator, and the label says that the agent is exitting. When this is done, it resets some variables like the elevator.InElevator. the InElevator variable is the variable that keeps track of if the elevator is empty or not.

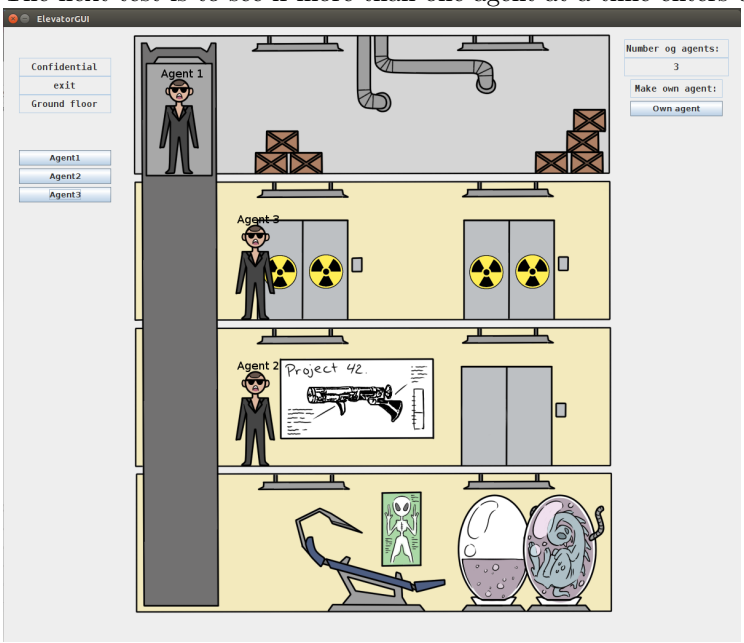
```
public void ElevatorRun(int agent) {
	if (!elevator.FloorCheck(elevator.level, elevator.floor)){
		FloorLevel(elevator.floor);
		agentEntered(agentNum(elevator.agentNr), elevator.floor);
		sleep();
		 enterExitlabel.setText("no access");
		sleep();
		enterExitlabel.setText(" ");
	}else{
		FloorLevel(elevator.floor);
		agentEntered(agentNum(elevator.agentNr), elevator.floor);
		sleep();
		enterExitlabel.setText("exit");
		sleep();
		agentExit(agentNum(elevator.agentNr), elevator.floor);
		sleep();
		enterExitlabel.setText(" ");
		clearencelabel.setText(" ");
		agentNum(elevator.agentNr).setVisible(false);
		agentsIn -= 1;
		agentnumlabel.setText(agentsIn + "");
		if (agent == 1) {
			agentcheck1 = 0;
		}else if (agent == 2) {
			agentcheck2 = 0;
		}else if (agent == 3) {
			agentcheck3 = 0;
		}
		elevator.inElevator = 0;
	}
}
```

4 Testing

First off to test if the window can open:

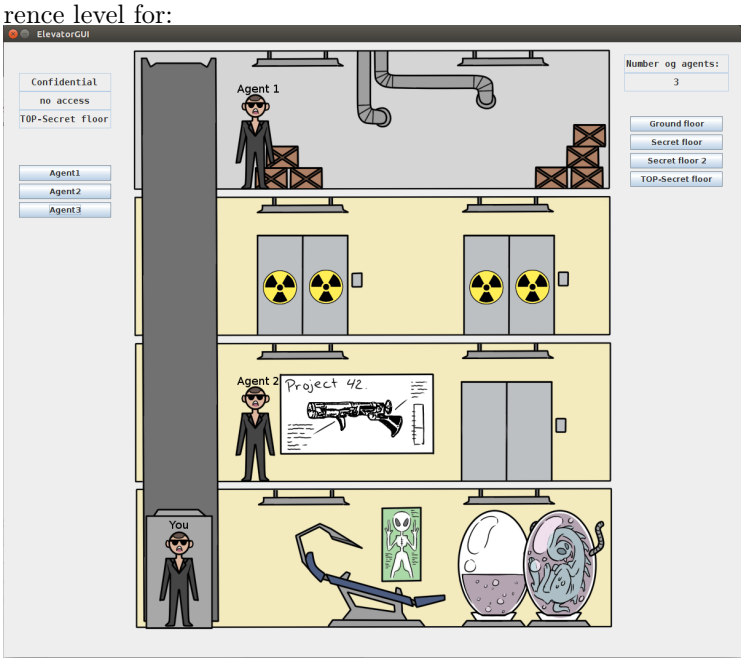


It opens the window nicely and displays the floors, buttons, labels and the elevator as it should.
The next test is to see if more than one agent at a time enters the elevator:

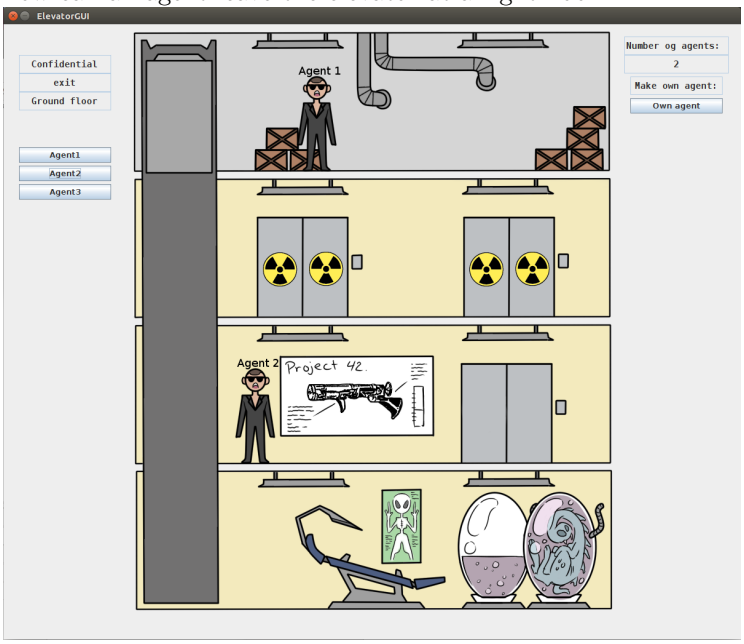


This looks good too, agent 2 and 3 are waiting for agent 1 to exit the elevator befor using it, and thanks to the random time, on the check to see if it is empty, the agents wont try and ride it at the same time.
What will happen if an agent tries to exit a floor he dosn't have the right clea-

rence level for:



The label will display "no access"and not let the agent exit on that floor.
now can an agent leave the elevator at a right floor:



It looks like agnet 1 was able to leave the elevator at the floor he has permission to leave. The top right label is displaying that he has a clearnece level of Confidential, which means he is only allowed to leave on the ground floor.

5 Appendix (source code)

5.1 LTSA code

```
const G = 0
const T2 = 3
const CAPACITY = 1
range LEVEL = 1..3
range FLOORS = G..T2
range NUMBER = 0..1

FLOORCHECK = FLOORCHECK[1],
FLOORCHECK[d:LEVEL] = ([h:LEVEL].scan -> SCAN[h]),
SCAN = SCAN[1],
SCAN[i:LEVEL] = (when (i > 0) [0].out -> FLOORCHECK[i] |
                            when (i > 1) [1].out -> FLOORCHECK[i] |
                            when (i > 2) [2..3].out -> FLOORCHECK[i]).

property NO_SEC = NO_SEC[1],
NO_SEC[d:LEVEL] = ([h:LEVEL].scan -> SCAN[h]),
SCAN = SCAN[1],
SCAN[i:LEVEL] = (when (i > 0) [0].out -> NO_SEC[i] |
                            when (i > 1) [1].out -> NO_SEC[i] |
                            when (i > 2) [2..3].out -> NO_SEC[i]).

ELEVATOR = ELEVATOR[G],
ELEVATOR[f:FLOORS] =
       ([f].in -> ELEVATOR[f] |
       [f].out -> ELEVATOR[f] |
       [g:FLOORS].callButton -> ELEVATOR[g] |
       [r:FLOORS].insideButton -> ELEVATOR[r]).

AREA_ELEVATOR = NORMAL[0],
NORMAL[i:NUMBER] = (when (i < CAPACITY) [G..T2].in -> NORMAL[i+1] |
                                    when (i < CAPACITY) [FLOORS].
                                        callButton -> NORMAL[i] |
                  when (i == CAPACITY) [G..T2].out -> NORMAL[i-1]|
                                    when (i == CAPACITY) [FLOORS].
                                        insideButton-> NORMAL[i] |
                                    when (i == CAPACITY) [LEVEL].scan ->
                                        NORMAL[i]).

||AREA_51 = (ELEVATOR || AREA_ELEVATOR || FLOORCHECK || NO_SEC).
```

5.2 Elevator.java

```
import java.util.*;

public class Elevator {
       ElevatorGUI gui;
       public int start;
```

```
6         public int level;
7         public int floor;
8         public int inElevator = 0;
9         public int okToExit;
10        public int agentNr;
11
12        public void setgui(ElevatorGUI gui){
13                this.gui = gui;
14        }
15
16        public void startFloor(int startfloor, int agentNr) {
17                gui.ElevatorPre(agentNr, startfloor);
18        }
19
20        public void floorstart(int startfloor) {
21                start = startfloor;
22        }
23
24        public boolean EnterElevator() {
25                Random rn = new Random();
26                int time = rn.nextInt(10) + 1;
27                try {
28                        Thread.sleep(time);
29                } catch(InterruptedException ex) {
30                        Thread.currentThread().interrupt();
31                }
32                if (inElevator == 0){
33                        return true;
34                }else{
35                        return false;
36                }
37        }
38
39        public void ChooseClearence(int clearenceLevel, int agentnr){
40                level = clearenceLevel;
41                agentNr = agentnr;
42                gui.ElevatorEnter(agentNr);
43        }
44
45        public void SelectedFloor(int floorSelected, int agent) {
46                floor = floorSelected;
47                if (inElevator != 0 && level != 0){
48                        gui.ElevatorRun(agent);
49                }
50        }
51
52        public int TjeckClerance(int level) {
53                if (level == 1){
54                        return 0;
55                }else if(level == 2) {
56                        return 1;
57                }else {
58                        return 3;
59                }
```

```
60          }
61
62          public boolean FloorCheck(int level, int floor) {
63                  if (TjeckClerance(level) >= floor) {
64                          return true;
65                  }else{
66                          return false;
67                  }
68          }
69
70          public void ExitOk(int possible) {
71                          okToExit = possible;
72          }
73  }
```

5.3 ElevatorGUI.java

```
1   import java.awt.image.BufferedImage;
2   import javax.imageio.ImageIO;
3   import java.io.IOException;
4   import java.awt.*;
5   import java.awt.event.*;
6   import javax.swing.*;
7   import java.io.*;
8
9   public class ElevatorGUI extends JPanel{
10          private JFrame Frame;
11          private JTextField floorlabel = new JTextField();
12          private JTextField clearencelabel = new JTextField();
13          private JTextField enterExitlabel = new JTextField();
14          private JTextField agentlabel = new JTextField();
15          private JTextField agentnumlabel = new JTextField();
16          private JTextField useragentlabel = new JTextField();
17          Elevator elevator;
18          private int check = 0;
19          private int agentsIn = 0;
20          private int agentcheck1 = 0;
21          private int agentcheck2 = 0;
22          private int agentcheck3 = 0;
23          Agent0 agent0;
24          Agent1 agent1;
25          Agent2 agent2;
26          Agent3 agent3;
27
28          JLabel floorsPic = new JLabel(new ImageIcon("floors.png"));
29          JLabel elevatorPic = new JLabel(new ImageIcon("elevator.png"));
30          JLabel agent0Pic = new JLabel(new ImageIcon("agent0.png"));
31          JLabel agent1Pic = new JLabel(new ImageIcon("agent1.png"));
32          JLabel agent2Pic = new JLabel(new ImageIcon("agent2.png"));
33          JLabel agent3Pic = new JLabel(new ImageIcon("agent3.png"));
34
35          JButton agent1Btn = new JButton("Agent1");
```

```
36          JButton agent2Btn = new JButton("Agent2");
37          JButton agent3Btn = new JButton("Agent3");
38
39          JButton userAgentBtn = new JButton("Own agent");
40          JButton gBtn = new JButton("Ground floor");
41          JButton sfBtn = new JButton("Secret floor");
42          JButton t1Btn = new JButton("Secret floor 2");
43          JButton t2Btn = new JButton("TOP-Secret floor");
44          JButton cBtn = new JButton("Confidential");
45          JButton sBtn = new JButton("Secret");
46          JButton tBtn = new JButton("TOP-Secret");
47          JButton rBtn = new JButton("Run agent");
48
49      public ElevatorGUI(Elevator elevator){
50                  this.elevator = elevator;
51          PrepareGUI();
52          display();
53      }
54
55      public void setagent0(Agent0 agent0){
56                  this.agent0 = agent0;
57                  }
58      public void setagent1(Agent1 agent1){
59                  this.agent1 = agent1;
60                  }
61          public void setagent2(Agent2 agent2){
62                  this.agent2 = agent2;
63                  }
64          public void setagent3(Agent3 agent3){
65                  this.agent3 = agent3;
66                  }
67
68          private void PrepareGUI(){
69                  Frame = new JFrame("ElevatorGUI");
70                  Frame.setResizable(false);
71                  Frame.setLayout(null);
72
73                  Insets insets = Frame.getInsets();
74          Frame.setSize(1200 + insets.left + insets.right,
75                        1000+ insets.top + insets.bottom);
76
77                  floorlabel.setHorizontalAlignment(JTextField.CENTER);
78                  floorlabel.setFont(new Font("Monospaced", Font.BOLD, 15));
79                  floorlabel.setEditable(false);
80                  floorlabel.setText("Ground floor");
81
82                  clearencelabel.setHorizontalAlignment(JTextField.CENTER);
83                  clearencelabel.setFont(new Font("Monospaced", Font.BOLD,
                        15));
84                  clearencelabel.setEditable(false);
85
86                  enterExitlabel.setHorizontalAlignment(JTextField.CENTER);
87                  enterExitlabel.setFont(new Font("Monospaced", Font.BOLD,
                        15));
```

```
88                enterExitlabel.setEditable(false);
89
90                agentlabel.setHorizontalAlignment(JTextField.CENTER);
91                agentlabel.setFont(new Font("Monospaced", Font.BOLD, 15));
92                agentlabel.setEditable(false);
93
94                agentnumlabel.setHorizontalAlignment(JTextField.CENTER);
95                agentnumlabel.setFont(new Font("Monospaced", Font.BOLD,
                      15));
96                agentnumlabel.setEditable(false);
97
98                useragentlabel.setHorizontalAlignment(JTextField.CENTER);
99                useragentlabel.setFont(new Font("Monospaced", Font.BOLD,
                      15));
100               useragentlabel.setEditable(false);
101
102               Frame.addWindowListener(new WindowAdapter() {
103                       public void windowClosing(WindowEvent windowEvent){
104                               System.exit(0);
105                       }
106               });
107
108               Frame.setVisible(true);
109       }
110
111       private void display(){
112
113               Agent1Event Agent1BtnEvent = new Agent1Event();
114               agent1Btn.addActionListener(Agent1BtnEvent);
115
116               Agent2Event Agent2BtnEvent = new Agent2Event();
117               agent2Btn.addActionListener(Agent2BtnEvent);
118
119               Agent3Event Agent3BtnEvent = new Agent3Event();
120               agent3Btn.addActionListener(Agent3BtnEvent);
121
122               userAgentEvent userAgentBtnEvent = new userAgentEvent();
123               userAgentBtn.addActionListener(userAgentBtnEvent);
124
125               gEvent gBtnEvent = new gEvent();
126               gBtn.addActionListener(gBtnEvent);
127
128               sfEvent sfBtnEvent = new sfEvent();
129               sfBtn.addActionListener(sfBtnEvent);
130
131               t1Event t1BtnEvent = new t1Event();
132               t1Btn.addActionListener(t1BtnEvent);
133
134               t2Event t2BtnEvent = new t2Event();
135               t2Btn.addActionListener(t2BtnEvent);
136
137               cEvent cBtnEvent = new cEvent();
138               cBtn.addActionListener(cBtnEvent);
139
```

```
140                 sEvent sBtnEvent = new sEvent();
141                 sBtn.addActionListener(sBtnEvent);
142 
143                 tEvent tBtnEvent = new tEvent();
144                 tBtn.addActionListener(tBtnEvent);
145 
146                 rEvent rBtnEvent = new rEvent();
147                 rBtn.addActionListener(rBtnEvent);
148 
149                 Frame.add(agentlabel);
150                 Frame.add(agentnumlabel);
151                 Frame.add(floorlabel);
152                 Frame.add(clearencelabel);
153                 Frame.add(enterExitlabel);
154                 Frame.add(useragentlabel);
155                 Frame.add(agent1Btn);
156                 Frame.add(agent2Btn);
157                 Frame.add(agent3Btn);
158 
159                 Frame.add(agent0Pic);
160                 Frame.add(agent1Pic);
161                 Frame.add(agent2Pic);
162                 Frame.add(agent3Pic);
163                 Frame.add(elevatorPic);
164                 Frame.add(floorsPic);
165 
166                 Frame.add(userAgentBtn);
167                 Frame.add(gBtn);
168                 Frame.add(sfBtn);
169                 Frame.add(t1Btn);
170                 Frame.add(t2Btn);
171                 Frame.add(cBtn);
172                 Frame.add(sBtn);
173                 Frame.add(tBtn);
174                 Frame.add(rBtn);
175 
176                 useragentlabel.setText("Make own agent:");
177                 agentlabel.setText("Number og agents: ");
178                 agentnumlabel.setText("" + agentsIn);
179 
180                 agent0Pic.setVisible(false);
181                 agent1Pic.setVisible(false);
182                 agent2Pic.setVisible(false);
183                 agent3Pic.setVisible(false);
184 
185                 gBtn.setVisible(false);
186                 sfBtn.setVisible(false);
187                 t1Btn.setVisible(false);
188                 t2Btn.setVisible(false);
189                 cBtn.setVisible(false);
190                 sBtn.setVisible(false);
191                 tBtn.setVisible(false);
192                 rBtn.setVisible(false);
193 
```

```
194                 Insets insets = Frame.getInsets();
195             //label:
196             agentlabel.setBounds(1010 + insets.left, 20 + insets.top, 170,
                    30);
197             agentnumlabel.setBounds(1010 + insets.left, 50 + insets.top, 170,
                     30);
198             clearencelabel.setBounds(25 + insets.left, 50 + insets.top, 150,
                    30);
199             enterExitlabel.setBounds(25 + insets.left, 80 + insets.top, 150,
                    30);
200             floorlabel.setBounds(25 + insets.left, 110 + insets.top, 150, 30)
                    ;
201                 //agent buttons:
202             agent1Btn.setBounds(25 + insets.left, 200 + insets.top, 150, 25);
203             agent2Btn.setBounds(25 + insets.left, 230 + insets.top, 150, 25);
204             agent3Btn.setBounds(25 + insets.left, 260 + insets.top, 150, 25);
205             elevatorPic.setBounds(226 + insets.left, 44 + insets.top, 119,
                    203);
206             floorsPic.setBounds(200 + insets.left, 10 + insets.top,800, 950);
207
208             useragentlabel.setBounds(1020 + insets.left, 85 + insets.top,
                    150, 30);
209             userAgentBtn.setBounds(1020 + insets.left, 120 + insets.top, 150,
                     25);
210             gBtn.setBounds(1020 + insets.left, 120 + insets.top, 150, 25);
211             sfBtn.setBounds(1020 + insets.left, 150 + insets.top, 150, 25);
212             t1Btn.setBounds(1020 + insets.left, 180 + insets.top, 150, 25);
213             t2Btn.setBounds(1020 + insets.left, 210 + insets.top, 150, 25);
214             cBtn.setBounds(1020 + insets.left, 120 + insets.top, 150, 25);
215             sBtn.setBounds(1020 + insets.left, 150 + insets.top, 150, 25);
216             tBtn.setBounds(1020 + insets.left, 180 + insets.top, 150, 25);
217             rBtn.setBounds(1020 + insets.left, 120 + insets.top, 150, 25);
218             }
219
220             public class Agent1Event implements ActionListener{
221                     public void actionPerformed(ActionEvent Agent1BtnEvent){
222                             System.out.println("1: " + agentcheck1);
223                             if (agentcheck1 == 0) {
224                                     Thread agent01 = new Thread(agent1);
225                                     agent01.start();
226                                     agentcheck1 += 1;
227                             }
228                     }
229             }
230
231             public class Agent2Event implements ActionListener{
232                     public void actionPerformed(ActionEvent Agent2BtnEvent){
233                             System.out.println("2: " + agentcheck2);
234                             if (agentcheck2 == 0) {
235                                     Thread agent02 = new Thread(agent2);
236                                     agent02.start();
237                                     agentcheck2 += 1;
238                             }
239                     }
```

```
240         }
241
242         public class Agent3Event implements ActionListener{
243                 public void actionPerformed(ActionEvent Agent3BtnEvent){
244                         System.out.println("3: " + agentcheck3);
245                         if (agentcheck3 == 0) {
246                                 Thread agent03 = new Thread(agent3);
247                                 agent03.start();
248                                 agentcheck3 += 1;
249                         }
250                 }
251         }
252
253         public class userAgentEvent implements ActionListener{
254                 public void actionPerformed(ActionEvent userAgentBtnEvent)
                        {
255                                 gBtn.setVisible(true);
256                                 sfBtn.setVisible(true);
257                                 t1Btn.setVisible(true);
258                                 t2Btn.setVisible(true);
259                                 userAgentBtn.setVisible(false);
260                                 useragentlabel.setVisible(false);
261                 }
262         }
263
264         public class gEvent implements ActionListener{
265                 public void actionPerformed(ActionEvent gBtnEvent){
266                         if(check == 0) {
267                                 agent0.startfloor = 0;
268                                 check = 1;
269                                 cBtn.setVisible(true);
270                                 sBtn.setVisible(true);
271                                 tBtn.setVisible(true);
272                                 gBtn.setVisible(false);
273                                 sfBtn.setVisible(false);
274                                 t1Btn.setVisible(false);
275                                 t2Btn.setVisible(false);
276                         }else if (check == 2){
277                                 agent0.selectedfloor = 0;
278                         } else {
279                                 agent0.selectedfloor = 0;
280                                 rBtn.setVisible(true);
281                                 gBtn.setVisible(false);
282                                 sfBtn.setVisible(false);
283                                 t1Btn.setVisible(false);
284                                 t2Btn.setVisible(false);
285                         }
286                 }
287         }
288
289         public class sfEvent implements ActionListener{
290                 public void actionPerformed(ActionEvent sfBtnEvent){
291                         if(check == 0) {
292                                 agent0.startfloor = 1;
```

```
293                             check = 1;
294                             cBtn.setVisible(true);
295                             sBtn.setVisible(true);
296                             tBtn.setVisible(true);
297                             gBtn.setVisible(false);
298                             sfBtn.setVisible(false);
299                             t1Btn.setVisible(false);
300                             t2Btn.setVisible(false);
301                     }else if (check == 2){
302                             agent0.selectedfloor = 1;
303                     } else {
304                             agent0.selectedfloor = 1;
305                             rBtn.setVisible(true);
306                             gBtn.setVisible(false);
307                             sfBtn.setVisible(false);
308                             t1Btn.setVisible(false);
309                             t2Btn.setVisible(false);
310                     }
311             }
312     }
313
314     public class t1Event implements ActionListener{
315             public void actionPerformed(ActionEvent t1BtnEvent){
316                     if(check == 0) {
317                             agent0.startfloor = 2;
318                             check = 1;
319                             cBtn.setVisible(true);
320                             sBtn.setVisible(true);
321                             tBtn.setVisible(true);
322                             gBtn.setVisible(false);
323                             sfBtn.setVisible(false);
324                             t1Btn.setVisible(false);
325                             t2Btn.setVisible(false);
326                     }else if (check == 2){
327                             agent0.selectedfloor = 2;
328                     } else {
329                             agent0.selectedfloor = 2;
330                             rBtn.setVisible(true);
331                             gBtn.setVisible(false);
332                             sfBtn.setVisible(false);
333                             t1Btn.setVisible(false);
334                             t2Btn.setVisible(false);
335                     }
336             }
337     }
338
339     public class t2Event implements ActionListener{
340             public void actionPerformed(ActionEvent t2BtnEvent){
341                     if(check == 0) {
342                             agent0.startfloor = 3;
343                             check = 1;
344                             cBtn.setVisible(true);
345                             sBtn.setVisible(true);
346                             tBtn.setVisible(true);
```

```
347                             gBtn.setVisible(false);
348                             sfBtn.setVisible(false);
349                             t1Btn.setVisible(false);
350                             t2Btn.setVisible(false);
351                     }else if (check == 2){
352                             agent0.selectedfloor = 3;
353                     } else {
354                             agent0.selectedfloor = 3;
355                             rBtn.setVisible(true);
356                             gBtn.setVisible(false);
357                             sfBtn.setVisible(false);
358                             t1Btn.setVisible(false);
359                             t2Btn.setVisible(false);
360                     }
361             }
362     }
363
364     public class cEvent implements ActionListener{
365             public void actionPerformed(ActionEvent cBtnEvent){
366                     agent0.clearence = 1;
367                     gBtn.setVisible(true);
368                     sfBtn.setVisible(true);
369                     t1Btn.setVisible(true);
370                     t2Btn.setVisible(true);
371                     cBtn.setVisible(false);
372                     sBtn.setVisible(false);
373                     tBtn.setVisible(false);
374             }
375     }
376
377     public class sEvent implements ActionListener{
378             public void actionPerformed(ActionEvent sBtnEvent){
379                     agent0.clearence = 2;
380                     gBtn.setVisible(true);
381                     sfBtn.setVisible(true);
382                     t1Btn.setVisible(true);
383                     t2Btn.setVisible(true);
384                     cBtn.setVisible(false);
385                     sBtn.setVisible(false);
386                     tBtn.setVisible(false);
387             }
388     }
389
390     public class tEvent implements ActionListener{
391             public void actionPerformed(ActionEvent tBtnEvent){
392                     agent0.clearence = 3;
393                     gBtn.setVisible(true);
394                     sfBtn.setVisible(true);
395                     t1Btn.setVisible(true);
396                     t2Btn.setVisible(true);
397                     cBtn.setVisible(false);
398                     sBtn.setVisible(false);
399                     tBtn.setVisible(false);
400             }
```

```
401         }
402
403         public class rEvent implements ActionListener{
404                 public void actionPerformed(ActionEvent rBtnEvent){
405                         Thread agent00 = new Thread(agent0);
406                                 agent00.start();
407                                 rBtn.setVisible(false);
408                 }
409         }
410
411         public void ElevatorPre(int agentnr, int startfloor) {
412                 addAgent(agentNum(agentnr), startfloor);
413                 agentsIn += 1;
414                 agentnumlabel.setText(agentsIn + "");
415                 sleep();
416         }
417
418         public void ElevatorEnter(int agentnr) {
419                 FloorLevel(elevator.start);
420                 agentEntered(agentNum(agentnr), elevator.start);
421                 enterExitlabel.setText("entered");
422                 sleep();
423                 ClearenceName();
424                 sleep();
425                 }
426
427         public void ElevatorRun(int agent) {
428                 FloorLevel(elevator.floor);
429                 agentEntered(agentNum(elevator.agentNr), elevator.floor);
430                 sleep();
431                 if (!elevator.FloorCheck(elevator.level, elevator.floor)){
432                          enterExitlabel.setText("no access");
433                         sleep();
434                         enterExitlabel.setText(" ");
435                 }else{
436                         enterExitlabel.setText("exit");
437                         sleep();
438                         agentExit(agentNum(elevator.agentNr), elevator.
                                floor);
439                         sleep();
440                         enterExitlabel.setText(" ");
441                         clearencelabel.setText(" ");
442                         agentNum(elevator.agentNr).setVisible(false);
443                         agentsIn -= 1;
444                         agentnumlabel.setText(agentsIn + "");
445                         if (agent == 1) {
446                                 agentcheck1 = 0;
447                         }else if (agent == 2) {
448                                 agentcheck2 = 0;
449                         }else if (agent == 3) {
450                                 agentcheck3 = 0;
451                         }
452                         elevator.inElevator = 0;
453                 }
```

```
454         }
455
456         public void AgentNewFloor() {
457                 check = 2;
458                 gBtn.setVisible(true);
459                 sfBtn.setVisible(true);
460                 t1Btn.setVisible(true);
461                 t2Btn.setVisible(true);
462
463         }
464
465         public void NewUserAgent() {
466                 check = 0;
467                 useragentlabel.setVisible(true);
468                 userAgentBtn.setVisible(true);
469                 gBtn.setVisible(false);
470                 sfBtn.setVisible(false);
471                 t1Btn.setVisible(false);
472                 t2Btn.setVisible(false);
473         }
474
475         public void FloorLevel(int floor) {
476                 Insets insets = Frame.getInsets();
477                 if (floor == 0){
478                         floorlabel.setText("Ground floor");
479                         elevatorPic.setBounds(226 + insets.left, 44 +
                                insets.top, 119, 203);
480                 }else if (floor == 1){
481                         floorlabel.setText("Secret floor");
482                         elevatorPic.setBounds(226 + insets.left, 281 +
                                insets.top, 119, 203);
483                 }else if (floor == 2){
484                         floorlabel.setText("Secret floor 2");
485                         elevatorPic.setBounds(226 + insets.left, 519 +
                                insets.top, 119, 203);
486                 }else{
487                         floorlabel.setText("TOP-Secret floor");
488                         elevatorPic.setBounds(226 + insets.left, 755 +
                                insets.top, 119, 203);
489                 }
490         }
491
492         public void ClearenceName()  {
493                 if (elevator.level == 1){
494                         clearencelabel.setText("Confidential");
495                 }else if (elevator.level == 2){
496                         clearencelabel.setText("Secret");
497                 }else{
498                         clearencelabel.setText("TOP-Secret");
499                 }
500         }
501
502         public JLabel agentNum(int agentnr) {
503                 if (agentnr == 0) {
```

```
504                     agent0Pic.setVisible(true);
505                     return agent0Pic;
506             }else if (agentnr == 1) {
507                     agent1Pic.setVisible(true);
508                     return agent1Pic;
509             }else if (agentnr == 2) {
510                     agent2Pic.setVisible(true);
511                     return agent2Pic;
512             }else if (agentnr == 3) {
513                     agent3Pic.setVisible(true);
514                     return agent3Pic;
515             }else{
516                     return null;
517             }
518     }
519
520     public void addAgent(JLabel agent, int floors) {
521             Insets insets = Frame.getInsets();
522             if (floors == 0) {
523                     agent.setBounds(350 + insets.left, 44 + insets.top,
                             119, 203);
524             }else if (floors == 1) {
525                     agent.setBounds(350 + insets.left, 281 + insets.top
                             , 119, 203);
526             }else if (floors == 2) {
527                     agent.setBounds(350 + insets.left, 519 + insets.top
                             , 119, 203);
528             }else if (floors == 3) {
529                     agent.setBounds(350 + insets.left, 755 + insets.top
                             , 119, 203);
530             }
531     }
532
533     public void agentEntered(JLabel agent, int floor) {
534             Insets insets = Frame.getInsets();
535             if (floor == 0){
536                     agent.setBounds(226 + insets.left, 44 + insets.top,
                             119, 203);
537             }else if (floor == 1){
538                     agent.setBounds(226 + insets.left, 281 + insets.top
                             , 119, 203);
539             }else if (floor == 2){
540                     agent.setBounds(226 + insets.left, 519 + insets.top
                             , 119, 203);
541             }else{
542                     agent.setBounds(226 + insets.left, 755 + insets.top
                             , 119, 203);
543             }
544     }
545
546     public void agentExit(JLabel agent, int floors) {
547             Insets insets = Frame.getInsets();
548             if (floors == 0) {
```

```
549                     agent.setBounds(450 + insets.left, 44 + insets.top,
                            119, 203);
550             }else if (floors == 1) {
551                     agent.setBounds(450 + insets.left, 281 + insets.top
                            , 119, 203);
552             }else if (floors == 2) {
553                     agent.setBounds(450 + insets.left, 519 + insets.top
                            , 119, 203);
554             }else if (floors == 3) {
555                     agent.setBounds(450 + insets.left, 755 + insets.top
                            , 119, 203);
556             }
557     }
558
559     public void sleep(){
560             try {
561                     Thread.sleep(1000);
562             } catch(InterruptedException ex) {
563                     Thread.currentThread().interrupt();
564             }
565     }
566 }
```

5.4 MainClass.java

```
1  public class MainClass{
2
3  public static void main(String[] args){
4
5                 Elevator elevator = new Elevator();
6                 Agent0 agent0 = new Agent0(elevator);
7                 Agent1 agent1 = new Agent1(elevator);
8                 Agent2 agent2 = new Agent2(elevator);
9                 Agent3 agent3 = new Agent3(elevator);
10                ElevatorGUI gui = new ElevatorGUI(elevator);
11                elevator.setgui(gui);
12                gui.setagent0(agent0);
13                gui.setagent1(agent1);
14                gui.setagent2(agent2);
15                gui.setagent3(agent3);
16                agent0.setgui(gui);
17        }
18 }
```

5.5 Agent0.java

```
1  public class Agent0 implements Runnable {
2         Elevator elevator;
3         ElevatorGUI gui;
4         public int startfloor;
```

```
5         public int clearence;
6         public int selectedfloor;
7
8         private int check = 0;
9         private int check1 = 1;
10        private int spawnCheck = 0;
11
12        public Agent0(Elevator elevator) {
13                this.elevator = elevator;
14        }
15        public void setgui(ElevatorGUI gui){
16                this.gui = gui;
17                }
18
19        public void run() {
20                while (check == 0){
21                        if (spawnCheck == 0){
22                                elevator.startFloor(startfloor, 0);
23                                spawnCheck = 1;
24                        }
25                        if (elevator.EnterElevator()){
26                                elevator.floorstart(startfloor);
27                                elevator.inElevator = 1;
28                                elevator.ChooseClearence(clearence, 0);
29                                elevator.SelectedFloor(selectedfloor, 0);
30                                System.out.println("1 running");
31                                while (check1 != 0) {
32                                        if (!elevator.FloorCheck(elevator.
                                              level, elevator.floor)){
33                                           gui.AgentNewFloor();
34                                           elevator.SelectedFloor(
                                                  selectedfloor, 1);
35                                        }else {
36                                                gui.NewUserAgent();
37                                                check = 1;
38                                                check1 = 0;
39                                                spawnCheck = 0;
40                                        }
41                                }
42                                check1= 1;
43                        }else{
44                                try {
45                                        Thread.sleep(1000);
46                                } catch(InterruptedException ex) {
47                                        Thread.currentThread().interrupt();
48                                }
49                        }
50                }
51                check = 0;
52    }
53 }
```

5.6 Agent1.java

```
1  public class Agent1 implements Runnable {
2         Elevator elevator;
3         private int check = 0;
4         private int spawnCheck = 0;
5
6         public Agent1(Elevator elevator) {
7                 this.elevator = elevator;
8         }
9
10        public void run() {
11                while (check == 0){
12                        if (spawnCheck == 0){
13                                elevator.startFloor(0, 1);
14                                spawnCheck = 1;
15                        }
16                        if (elevator.EnterElevator()){
17                                elevator.floorstart(0);
18                                elevator.inElevator = 1;
19                                elevator.ChooseClearence(1, 1);
20                                elevator.SelectedFloor(2, 1);
21                                System.out.println("1 running");
22                                if (!elevator.FloorCheck(elevator.level,
                                      elevator.floor)){
23                                        elevator.SelectedFloor(0, 1);
24                                }
25                                check = 1;
26                                spawnCheck = 0;
27                        }else{
28                                try {
29                                        Thread.sleep(1000);
30                                } catch(InterruptedException ex) {
31                                        Thread.currentThread().interrupt();
32                                }
33                        }
34                }
35                check = 0;
36    }
37 }
```

5.7 Agent2.java

```
1  public class Agent2 implements Runnable {
2         Elevator elevator;
3         private int check = 0;
4         private int spawnCheck = 0;
5
6         public Agent2(Elevator elevator) {
7                 this.elevator = elevator;
8         }
```

```
9
10          public void run() {
11                  while (check == 0){
12                          if (spawnCheck == 0){
13                                  elevator.startFloor(2, 2);
14                                  spawnCheck = 1;
15                          }
16                          if (elevator.EnterElevator()){
17                                  elevator.floorstart(2);
18                                  elevator.inElevator = 1;
19                                  elevator.ChooseClearence(3, 2);
20                                  elevator.SelectedFloor(3, 2);
21                                  System.out.println("2 running");
22                                  if (!elevator.FloorCheck(elevator.level,
                                        elevator.floor)){
23                                      elevator.SelectedFloor(0, 2);
24                                  }
25                                  check = 1;
26                                  spawnCheck = 0;
27                          }else{
28                                  try {
29                                          Thread.sleep(1000);
30                                  } catch(InterruptedException ex) {
31                                          Thread.currentThread().interrupt();
32                                  }
33                          }
34                  }
35                  check = 0;
36          }
37  }
```

5.8 Agent3.java

```
1   import java.util.*;
2   public class Agent3 implements Runnable {
3           Elevator elevator;
4           public boolean a3;
5           private int check = 0;
6           private int spawnCheck = 0;
7           private int startFloorLevel;
8
9           public Agent3(Elevator elevator) {
10                  this.elevator = elevator;
11          }
12
13          public void run() {
14                  int exitCheck = 0;
15                  while (check == 0){
16                          if (spawnCheck == 0){
17                                  startFloorLevel = randFloor();
18                                  elevator.startFloor(startFloorLevel, 3);
19                                  spawnCheck = 1;
```

```
20
21                      }
22                      if (elevator.EnterElevator()){
23                              elevator.floorstart(startFloorLevel);
24                              elevator.inElevator = 1;
25                              elevator.ChooseClearence(randClerance(), 3);
26                              elevator.SelectedFloor(randFloor(), 2);
27                              System.out.println("3 running");
28                              check = 1;
29                              while(exitCheck != 1) {
30                                      if (!elevator.FloorCheck(elevator.
                                            level, elevator.floor)){
31                                              elevator.SelectedFloor(
                                                    randFloor(), 3);
32                                      }else {
33                                              exitCheck = 1;
34                                      }
35                                      spawnCheck = 0;
36                              }
37                      }else{
38                              try {
39                                      Thread.sleep(1000);
40                              } catch(InterruptedException ex) {
41                                      Thread.currentThread().interrupt();
42                              }
43                      }
44              }
45              check = 0;
46      }
47
48      public int randClerance() {
49              Random rn = new Random();
50              int num = rn.nextInt(3) + 1;
51              return num;
52      }
53
54      public int randFloor() {
55              Random rn = new Random();
56              int num = rn.nextInt(4) + 0;
57              return num;
58      }
59
60  }
```