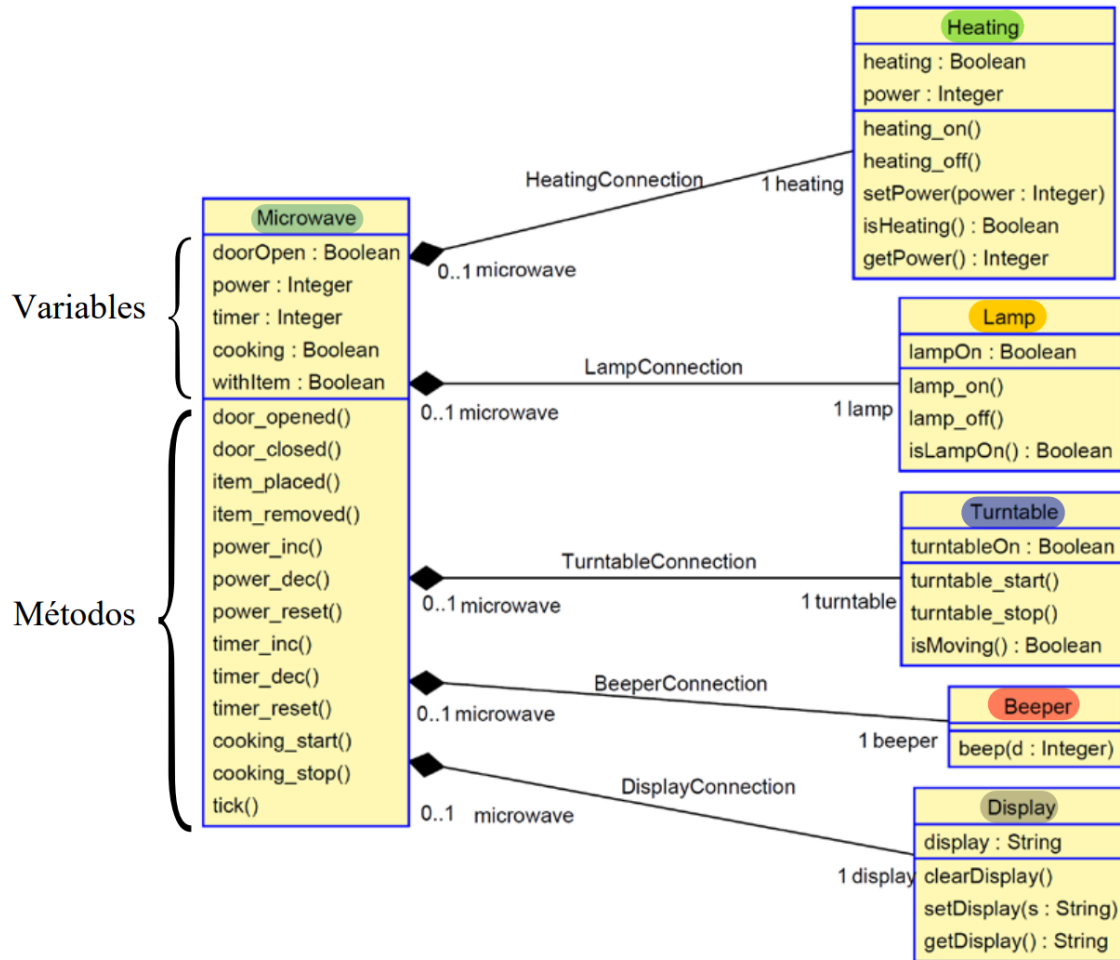


Ingeniería del Software Avanzada

Práctica Final: Horno Microondas



Autor: David Ramírez Arco

Fecha: 10 de junio de 2022

GitHub: https://github.com/Archerd6/Proyecto_Microondas

Índice




Apartados.....	2
Apartado A.....	2
Apartado B.....	20
Apartado C.....	26
StepDefinitions.....	29
Apartado D.....	31
Resultados.....	32
Conclusión.....	32

La interfaz de microondas que he creado si sigue el Diagrama:

```
1  package microwave;
2
3  public interface Microwave_Interface
4  {
5      /**Increment on 10w the power*/
6      public void power_inc(Microwave mw);
7      /**Decrement on 10w the power*/
8      public void power_dec(Microwave mw);
9      /**Set power to 0w*/
10     public void power_reset(Microwave mw);
11     /**Increase the timer*/
12     public void timer_inc(Microwave mw);
13     /**Decrease the timer*/
14     public void timer_dec(Microwave mw);
15     /**Set the timer to 0*/
16     public void timer_reset(Microwave mw);
17     /**Try to start the microwave*/
18
19     /**Try to open the door*/
20     public void door_opened(Microwave mw);
21     /**Try to close the door*/
22     public void door_closed(Microwave mw);
23     /**Try to place a item*/
24     public void item_placed(Microwave mw);
25     /**Try to remove a item*/
26     public void item_removed(Microwave mw);
27
28     public void cooking_start(Microwave mw);
29     /**Try to stop the microwave*/
30     public void cooking_stop(Microwave mw);
31     /**Once invoked this reduce the time remaining in timer one unit*/
32     public void tick(Microwave mw);
33 }
```

Beeper

15 lines (14 sloc) | 279 Bytes




Raw Blame   

```
1 package microwave;
2
3 /**Bell that warns when the timer has reached zero
4  * @author ap27r
5  */
6 public class Beeper
7 {
8     /**Makes the bell ring as many times as indicated by the parameter
9     * @param d - Times to beep
10    */
11    public void beep(int d)
12    {
13        BeeperCounter.listen(d);
14    }
15 }
```

BeeperCounter

He creado una clase pública llamada BeeperCounter para recibir el parámetro utilizado en la clase Beeper, lo guarda y comprueba para la hora de hacer las distintas pruebas en el proyecto.

31 lines (28 sloc) | 632 Bytes

Raw Blame   

```
1 package microwave;
2 /**Listener to the Beeper class, as Beeper don't have variables to save in the UML diagram
3  * @author ap27r
4  */
5 public class BeeperCounter
6 {
7     /**Variables to save the number of beeps*/
8     private static int num_beeps;
9
10    /**Main metod used by the Beeper class
11    * @param times_beeped
12    */
13    public static void listen(int times_beeped)
14    {
15        num_beeps = times_beeped;
16    }
17
18    /**Has the beep beeped?*/
19    public static boolean isBeeped(int t)
20    {
21        return (clear() == t);
22    }
23
24    /**Reset number of beep and return old value*/
25    private static int clear()
26    {
27        int bp = num_beeps;
28        num_beeps = 0;
29        return bp;
30    }
31 }
```

Display

26 lines (24 sloc) | 579 Bytes

Raw Blame



```
1 package microwave;
2
3 /**Display that allows the microwave to show different messages (for example, "The food is ready")*/
4 public class Display
5 {
6     //Variable
7     private String display;
8
9     /**The operation clearDisplay() clears the contents of the screen and turns it off*/
10    public void clearDisplay()
11    {
12        display = null;
13    }
14    /**The screen turns back on when calls the setDisplay() operation
15     * @param s - String that will show the GUI
16     */
17    public void setDisplay(String s)
18    {
19        display = s;
20    }
21    /**Getter of the display*/
22    public String getDisplay()
23    {
24        return display;
25    }
26 }
```

Heating

47 lines (42 sloc) | 989 Bytes

Raw Blame



```
1 package microwave;
2 /**Magnetron device that emits microwaves, in charge of heat the food to a certain power (<b>power</b>) <br>
3  * <br>
4  * The microwave turns it on and off using the <b>heating_on()</b> and <b>heating_off()</b> operations <br>
5  * You can also know whether it is on or not with the <b>isHeating()</b> query operation <br>
6  * The microwave component can also set the power and meet it with the <b>setPower()</b> and <b>getPower()</b> operations <br>
7  *
8  * @author ap27r
9  */
10 public class Heating
11 {
12     // Variables
13     private boolean heating = false;
14     private int power = 0;
15
16     /**Setter to true*/
17     public void heating_on()
18     {
19         heating = true;
20     }
21
22     /**Setter to false*/
23     public void heating_off()
24     {
25         heating = false;
26     }
27
28     /**Setter to value*/
29     public void setPower(int power)
30     {
31         if (power >= 0) // Can't give negative power
32         {
33             this.power = power;
34         }
35     }
36
37     // Getters
38     public boolean isHeating()
39     {
40         return heating;
41     }
42
43     public int getPower()
44     {
45         return power;
46     }
47 }
```

Lamp

31 lines (27 sloc) | 532 Bytes

Raw Blame



```
1 package microwave;
2
3 /** Lamp that turns on (<b>lamp_on()</b>) or off (<b>lamp_off()</b>) based on different events, such as: <br>
4  * <br>
5  * - Door open <br>
6  * - Microwave working a query operation
7  *
8  * @author ap27r
9  */
10 public class Lamp
11 {
12     private boolean lampOn = false;
13
14     /**Setter to true*/
15     public void lamp_on()
16     {
17         lampOn = true;
18     }
19
20     /**Setter to false*/
21     public void lamp_off()
22     {
23         lampOn = false;
24     }
25
26     /**Allows the microwave to know if the light is on or not*/
27     public boolean isLampOn()
28     {
29         return lampOn;
30     }
31 }
```

Turntable

30 lines (26 sloc) | 745 Bytes

Raw Blame



```
1 package microwave;
2
3 /**Turntable triggered by the <b>turntable_start()</b> operation when the microwave is running and
4  * stops (via the turntable_stop() operation) when the door is opened or cooking time is running out.<br>
5  * <br>
6  * It implements an <b>isMoving()</b> query operation that lets you know where all the time whether the platter is spinning or not.
7  * @author ap27r
8  * */
9 public class Turntable
10 {
11     private boolean turntableOn = false;
12
13     /**Setter to true*/
14     public void turntable_start()
15     {
16         turntableOn = true;
17     }
18
19     /**Setter to false*/
20     public void turntable_stop()
21     {
22         turntableOn = false;
23     }
24
25     /**Allows the microwave to know if the platterplatter is spinning*/
26     public boolean isMoving()
27     {
28         return turntableOn;
29     }
30 }
```

Microwave

205 lines (167 sloc) | 3.26 KB

Raw

Blame



```
1  package microwave;
2
3  /**This class represent a microwave, that might have a set of states, depending of de actions performed on it
4   * @author David RA
5   * @version 1.0
6   * */
7  public class Microwave
8  {
9      // Variables
10     private int power;
11     private int timer;
12     private boolean doorOpen;
13     private boolean cooking;
14     private boolean withItem;
15
16     // Variables componentes
17     private Heating heatingComponent = new Heating();
18     private Lamp lampComponent = new Lamp();
19     private Turntable turntableComponent = new Turntable();
20     private Beeper beeperComponent = new Beeper();
21     private Display displayComponent = new Display();
22     /**Class that implement the microwave_interface*/
23     private Microwave_Interface state;
24
25     /**Main constructor: creates a microwave closed with no item
26     *
27     * Must verify that the power and timer are 0
28     * the door, cook, item variables are false
29     * */
30     public Microwave()
31     {
32         state = new MW_ClosedWithNoItem(this);
33
34         power = 0;
35         timer = 0;
36         doorOpen = false;
37         cooking = false;
38         withItem = false;
39     }
40
41     public void power_inc()
42     {
43         state.power_inc(this);
44     }
45
46     public void power_dec()
47     {
48         state.power_dec(this);
49     }
50
51     public void power_reset()
52     {
53         state.power_reset(this);
54     }
55
56     public void timer_inc()
57     {
58         state.timer_inc(this);
59     }
60
61     public void timer_dec()
62     {
63         state.timer_dec(this);
64     }
65
66     public void timer_reset()
67     {
68         state.timer_reset(this);
69         displayComponent.setDisplay(Integer.toString(timer));
70     }
71 }
```

```
72
73     public void door_opened()
74     {
75         state.door_opened(this);
76     }
77
78     public void door_closed()
79     {
80         state.door_closed(this);
81     }
82
83     public void item_placed()
84     {
85         state.item_placed(this);
86     }
87
88     public void item_removed()
89     {
90         state.item_removed(this);
91     }
92
93     public void cooking_start()
94     {
95         state.cooking_start(this);
96     }
97
98     public void cooking_stop()
99     {
100         state.cooking_stop(this);
101     }
102
103     /**An external clock is in charge of invoking the tick() operation of the microwave every second, allowing you to know the passage of time*/
104     public void tick()
105     {
106         state.tick(this);
107     }
108
```



```
109
110     // Not represented in the main UML diagram but necessary
111
112     public int getPower()
113     {
114         return power;
115     }
116
117     public void setPower(int power)
118     {
119         this.power = power;
120     }
121
122     public int getTime()
123     {
124         return timer;
125     }
126
127     public void setTime(int timer)
128     {
129         if(timer > 0)
130         {
131             this.timer = timer;
132         }
133         else
134         {
135             this.timer = 0; // Not negative numbers in the timer
136         }
137     }
138
```



```
139
140     public boolean isOpen()
141     {
142         return doorOpen;
143     }
144
145     public void setOpen(boolean doorOpen)
146     {
147         this.doorOpen = doorOpen;
148     }
149
150     public boolean isItem()
151     {
152         return withItem;
153     }
154
155     public void setItem(boolean withItem)
156     {
157         this.withItem = withItem;
158     }
159
160     public boolean isCooking()
161     {
162         return cooking;
163     }
164
165     public void setCooking(boolean cooking)
166     {
167         this.cooking = cooking;
168     }
169
170     public Microwave_Interface getState()
171     {
172         return state;
173     }
174
175     public Heating getHeatComponent()
176     {
177         return heatingComponent;
178     }
179
180     public Lamp getLampComponent()
181     {
182         return lampComponent;
183     }
184
185     public Turntable getTurntComponent()
186     {
187         return turnableComponent;
188     }
189
190     public Beeper getBeepComponent()
191     {
192         return beeperComponent;
193     }
194
195     public Display getDisplayComponent()
196     {
197         return displayComponent;
198     }
199
200     /**Method to change the state automatically -Needed for some tests*/
201     public void setState(Microwave_Interface status)
202     {
203         this.state = status;
204     }
205 }
```

MW_ClosedWithNoItem

119 lines (103 sloc) | 3.14 KB




Raw Blame   

```
1 package microwave;
2
3 /**This class represent a microwave that is closed, with no item inside of it
4  * @author David RA
5  * @version 1.1
6  * */
7 public class MW_ClosedWithNoItem implements Microwave_Interface
8 {
9     /**The constructor must verify that the lamp, heating, turnable, Cooking and the door are off
10     * An must have a item inside
11     * */
12     public MW_ClosedWithNoItem(Microwave m)
13     {
14         m.getLampComponent().lamp_off(); // Check lamp
15         m.getHeatComponent().heating_off(); // Check heat
16         m.getTurntComponent().turntable_stop(); // Check rotation
17         m.setCooking(false); // Check cooking
18         m.setOpen(false); // Check door
19         m.setItem(false); // Check item
20         m.getDisplayComponent().clearDisplay(); // Display restarted, as this is the state created by default in the microwave
21     }
22
23     @Override
24     public void door_opened(Microwave m)
25     {
26         m.setState(new MW_OpenWithNoItem(m));
27     }
28
29     @Override
30     public void door_closed(Microwave m)
31     {
32         // Exception would be manage - Message written in English don't show to the users
33         throw new IllegalStateException("You cant close the door of a already closed microwave");
34     }
35
36     @Override
37     public void item_placed(Microwave m)
38     {
39         // Exception would be manage - Message written in English don't show to the users
40         throw new IllegalStateException("You cant place a item in a closed microwave");
41     }
42
43     @Override
44     public void item_removed(Microwave m)
45     {
46         // Exception would be manage - Message written in English don't show to the users
47         throw new IllegalStateException("You cant remove a item having the door closed");
48     }
49
50     @Override
51     public void power_inc(Microwave m)
52     {
53         m.setPower(m.getPower() + 10); // Increment on 10w the power as said in interface
54         m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
55     }
56
57     @Override
58     public void power_dec(Microwave m)
59     {
60         if (m.getPower() > 0)
61         {
62             m.setPower(m.getPower() - 10); // Decrement on 10w the power as said in interface
63             m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
64         }
65     }
66
67     @Override
68     public void power_reset(Microwave m)
69     {
70         m.setPower(0);
71         m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
72     }
73 }
```

```
74     @Override
75     public void timer_inc(Microwave mw)
76     {
77         mw.setTime(mw.getTime() + 1);
78         mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
79     }
80
81     @Override
82     public void timer_dec(Microwave m)
83     {
84         if (m.getTime() > 0)
85         {
86             m.setTime(m.getTime() - 1);
87             m.getDisplayComponent().setDisplay(Integer.toString(m.getTime()));
88         }
89     }
90
91     @Override
92     public void timer_reset(Microwave m)
93     {
94         m.setTime(0);
95     }
96
97     @Override
98     public void cooking_start(Microwave m)
99     {
100         // Exception would be manage - Message written in English don't show to the users
101         throw new IllegalStateException("You cant start cook with no item inside");
102     }
103
104     @Override
105     public void cooking_stop(Microwave m)
106     {
107         // Exception would be manage - Message written in English don't show to the users
108         throw new IllegalStateException("Microwave was already stopped");
109     }
110
111     @Override
112     public void tick(Microwave m)
113     {
114         // Exception would be manage - Message written in English don't show to the users
115         throw new IllegalStateException("Timer must not run when microwave not running");
116     }
117
118
119 }
```

MW_OpenWithNoItem

116 lines (101 sloc) | 2.95 KB

Raw Blame   

```
1 package microwave;
2
3 /**This class represent a microwave that is opened, with no item inside of it
4  * @author David RA
5  * @version 1.1
6  * */
7 public class MW_OpenWithNoItem implements Microwave_Interface
8 {
9     /**The constructor must verify that the heating, turnable, Cooking and the door are off
10     * Must have a item inside and lamp turned on
11     * */
12     public MW_OpenWithNoItem(Microwave mw)
13     {
14         mw.setCooking(false); // Check cooking
15         mw.setItem(false); // Check item
16         mw.setOpen(true); // Check door
17
18         mw.getLampComponent().lamp_on(); // Check lamp
19         mw.getHeatComponent().heating_off(); // Check heat
20         mw.getTurnComponent().turntable_stop(); // Check rotation
21     }
22
23     @Override
24     public void door_opened(Microwave mw)
25     {
26         // Exception would be manage - Message written in English don't show to the users
27         throw new IllegalStateException("You cant open a door in a opened microwave");
28     }
29
30     @Override
31     public void door_closed(Microwave mw)
32     {
33         mw.setState(new MW_ClosedWithNoItem(mw));
34     }
35
36     @Override
37     public void item_placed(Microwave mw)
38     {
39         mw.setState(new MW_OpenWithItem(mw));
40     }
41
42     @Override
43     public void item_removed(Microwave mw)
44     {
45         // Exception would be manage - Message written in English don't show to the users
46         throw new IllegalStateException("There is no item to be removed in the microwave");
47     }
48
49     @Override
50     public void power_inc(Microwave m)
51     {
52         m.setPower(m.getPower() + 10); // Increment on 10w the power as said in interface
53         m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
54     }
55
56     @Override
57     public void power_dec(Microwave mw)
58     {
59         if (mw.getPower() > 0)
60         {
61             mw.setPower(mw.getPower() - 10); // Decrement on 10w the power as said in interface
62             mw.getDisplayComponent().setDisplay(Integer.toString(mw.getPower()));
63         }
64     }
65
66     @Override
67     public void power_reset(Microwave m)
68     {
69         m.setPower(0);
70         m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
71     }
72 }
```

```
73     @Override
74     public void timer_inc(Microwave mw)
75     {
76         mw.setTime(mw.getTime() + 1);
77         mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
78     }
79
80     @Override
81     public void timer_dec(Microwave mw)
82     {
83         if (mw.getTime() > 0)
84         {
85             mw.setTime(mw.getTime() - 1);
86             mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
87         }
88     }
89
90     @Override
91     public void timer_reset(Microwave mw)
92     {
93         mw.setTime(0);
94     }
95
96     @Override
97     public void cooking_start(Microwave mw)
98     {
99         // Exception would be manage - Message written in English don't show to the users
100        throw new IllegalStateException("You must no be able to start the microwave with the door oppened");
101    }
102
103    @Override
104    public void cooking_stop(Microwave mw)
105    {
106        // Exception would be manage - Message written in English don't show to the users
107        throw new IllegalStateException("Microwave was stopped before");
108    }
109
110    @Override
111    public void tick(Microwave mw)
112    {
113        // Exception would be manage - Message written in English don't show to the users
114        throw new IllegalStateException("Timer must not run when microwave not running");
115    }
116 }
```

MW_OpenWithItem

115 lines (101 sloc) | 2.94 KB




Raw Blame

```
1 package microwave;
2
3 /**This class represent a microwave that is opened, with a item inside of it
4  * @author David RA
5  * @version 1.1
6  * */
7 public class MW_OpenWithItem implements Microwave_Interface
8 {
9     /**The constructor must verify that the heating, turnable, Cooking and the door are off
10     * Must have a item inside and lamp turned on
11     * */
12     public MW_OpenWithItem(Microwave mw)
13     {
14         mw.setCooking(false); // Check cooking
15         mw.setItem(true); // Check item
16         mw.setOpen(true); // Check door
17         mw.getLampComponent().lamp_on(); // Check lamp
18         mw.getHeatComponent().heating_off(); // Check heat
19         mw.getTurnComponent().turntable_stop(); // Check rotation
20     }
21
22     @Override
23     public void door_opened(Microwave mw)
24     {
25         // Exception would be manage - Message written in English don't show to the users
26         throw new IllegalStateException("You cant open a door in a opened microwave");
27     }
28
29     @Override
30     public void door_closed(Microwave mw)
31     {
32         mw.setState(new MW_ClosedWithItem(mw));
33     }
34
35     @Override
36     public void item_placed(Microwave mw)
37     {
38         // Exception would be manage - Message written in English don't show to the users
39         throw new IllegalStateException("You cant place a item in a microwave that have some item inside");
40     }
41
42     @Override
43     public void item_removed(Microwave mw)
44     {
45         mw.setState(new MW_OpenWithNoItem(mw));
46     }
47
48     @Override
49     public void power_inc(Microwave m)
50     {
51         m.setPower(m.getPower() + 10); // Increment on 10w the power as said in interface
52         m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
53     }
54
55     @Override
56     public void power_dec(Microwave m)
57     {
58         if (m.getPower() > 0)
59         {
60             m.setPower(m.getPower() - 10); // Decrement on 10w the power as said in interface
61             m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
62         }
63     }
64
65     @Override
66     public void power_reset(Microwave m)
67     {
68         m.setPower(0);
69         m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
70     }
71 }
```

```
72     @Override
73     public void timer_inc(Microwave mw)
74     {
75         mw.setTime(mw.getTime() + 1);
76         mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
77     }
78
79     @Override
80     public void timer_dec(Microwave m)
81     {
82         if (m.getTime() > 0)
83         {
84             m.setTime(m.getTime() - 1);
85             m.getDisplayComponent().setDisplay(Integer.toString(m.getTime()));
86         }
87     }
88
89     @Override
90     public void timer_reset(Microwave mw)
91     {
92         mw.setTime(0);
93     }
94
95     @Override
96     public void cooking_start(Microwave mw)
97     {
98         // Exception would be manage - Message written in English don't show to the users
99         throw new IllegalStateException("You must no be able to start the microwave with the door oppened");
100     }
101
102     @Override
103     public void cooking_stop(Microwave mw)
104     {
105         // Exception would be manage - Message written in English don't show to the users
106         throw new IllegalStateException("Microwave was stopped before");
107     }
108
109     @Override
110     public void tick(Microwave m)
111     {
112         // Exception would be manage - Message written in English don't show to the users
113         throw new IllegalStateException("Timer must not run when microwave not running");
114     }
115 }
```

MW_ClosedWithItem

139 lines (123 sloc) | 3.67 KB

Raw Blame   




```
1 package microwave;
2
3 /**This class represent a microwave that is closed, with a item inside of it
4  * @author David RA
5  * @version 1.1
6  * */
7 public class MW_ClosedWithItem implements Microwave_Interface
8 {
9     /**The constructor must verify that the lamp, heating, turnable, Cooking and the door are off
10     * An must have a item inside
11     * */
12     public MW_ClosedWithItem(Microwave mw)
13     {
14         mw.getLampComponent().lamp_off(); // Check lamp
15         mw.getHeatComponent().heating_off(); // Check heat
16         mw.getTurntComponent().turntable_stop(); // Check rotation
17         mw.setCooking(false); // Check cooking
18         mw.setOpen(false); // Check door
19         mw.setItem(true); // Check item
20     }
21
22     @Override
23     public void door_opened(Microwave mw)
24     {
25         // Exception would be manage - Message written in English don't show to the users
26         mw.setState(new MW_OpenWithItem(mw));
27     }
28
29     @Override
30     public void door_closed(Microwave mw)
31     {
32         // Exception would be manage - Message written in English don't show to the users
33         throw new IllegalStateException("You cant close the door of a already closed microwave");
34     }
35
36     @Override
37     public void item_placed(Microwave mw)
38     {
39         // Exception would be manage - Message written in English don't show to the users
40         throw new IllegalStateException("You cant place a item in a closed microwave");
41     }
42
43     @Override
44     public void item_removed(Microwave mw)
45     {
46         // Exception would be manage - Message written in English don't show to the users
47         throw new IllegalStateException("You cant remove a item having the door closed");
48     }
49
50     @Override
51     public void power_inc(Microwave mw)
52     {
53         mw.setPower(mw.getPower() + 10); // Increment on 10w the power as said in interface
54         mw.getDisplayComponent().setDisplay(Integer.toString(mw.getPower()));
55     }
56
57     @Override
58     public void power_dec(Microwave mw)
59     {
60         if (mw.getPower() > 0)
61         {
62             mw.setPower(mw.getPower() - 10); // Decrement on 10w the power as said in interface
63             mw.getDisplayComponent().setDisplay(Integer.toString(mw.getPower()));
64         }
65     }
66
67     @Override
68     public void power_reset(Microwave mw)
69     {
70         mw.setPower(0);
71         mw.getDisplayComponent().setDisplay(Integer.toString(mw.getPower()));
72     }
73 }
```



```
74     @Override
75     public void timer_inc(Microwave mw)
76     {
77         mw.setTime(mw.getTime() + 1);
78         mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
79     }
80
81     @Override
82     public void timer_dec(Microwave mw)
83     {
84         if (mw.getTime() > 0)
85         {
86             mw.setTime(mw.getTime() - 1);
87             mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
88         }
89     }
90
91     @Override
92     public void timer_reset(Microwave mw)
93     {
94         mw.setTime(0);
95     }
96
97     @Override
98     public void cooking_start(Microwave mw)
99     {
100         if(!(mw.getTime() > 0) && !(mw.getPower() > 0))
101         {
102             // Exception would be manage - Message written in English don't show to the users
103             throw new IllegalStateException("You cant start cook with no time neither power");
104         }
105
106         if(mw.getTime() > 0 && mw.getPower() > 0)
107         {
108             mw.setState(new Mw_Cooking(mw));
109         }
110         else
111         {
112             if(mw.getTime() > 0)
113             {
114                 // Exception would be manage - Message written in English don't show to the users
115                 throw new IllegalStateException("You cant start cook with no time");
116             }
117             else
118             {
119                 // Exception would be manage - Message written in English don't show to the users
120                 throw new IllegalStateException("You cant start cook with no power");
121             }
122         }
123     }
124
125     @Override
126     public void cooking_stop(Microwave mw)
127     {
128         // Exception would be manage - Message written in English don't show to the users
129         throw new IllegalStateException("Microwave was already stopped");
130     }
131
132     @Override
133     public void tick(Microwave mw)
134     {
135         // Exception would be manage - Message written in English don't show to the users
136         throw new IllegalStateException("Timer must not run when microwave not running");
137     }
138
139 }
```

MW_Cooking

148 lines (131 sloc) | 3.52 KB

Raw Blame   

```
1 package microwave;
2
3 /**This class represent a microwave that is cooking (so have a item, ligh on, rotating ...)
4  * @author David RA
5  * @version 1.2
6  * */
7 public class MW_Cooking implements Microwave_Interface
8 {
9     /**The constructor must verify that the lamp, heating, turnable, Cooking are on
10     * Must have a item inside and the door closed
11     * */
12     public MW_Cooking(Microwave mw)
13     {
14         mw.setCooking(true); // Check cooking
15         mw.setOpen(false); // Check door
16         mw.setItem(true); // Check item
17
18         mw.getHeatComponent().setPower(mw.getPower()); // Set power value saved in the microwave variable
19
20         mw.getLampComponent().lamp_on(); // Check lamp
21         mw.getHeatComponent().heating_on(); // Check heat
22         mw.getTurnComponent().turntable_start(); // Check rotation
23     }
24
25     @Override
26     public void door_opened(Microwave mw)
27     {
28         mw.setState(new MW_OpenWithItem(mw));
29     }
30
31     @Override
32     public void door_closed(Microwave mw)
33     {
34         // Exception would be manage - Message written in English don't show to the users
35         throw new IllegalStateException("You cant close the door of a already closed microwave");
36     }
37
38     @Override
39     public void item_placed(Microwave mw)
40     {
41         // Exception would be manage - Message written in English don't show to the users
42         throw new IllegalStateException("You cant place a item in a closed microwave");
43     }
44
45     @Override
46     public void item_removed(Microwave mw)
47     {
48         // Exception would be manage - Message written in English don't show to the users
49         throw new IllegalStateException("You cant remove a item having the door closed");
50     }
51
52     @Override
53     public void power_inc(Microwave m)
54     {
55         m.setPower(m.getPower() + 10); // Increment on 10w the power as said in interface
56         m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
57     }
58
59     @Override
60     public void power_dec(Microwave mw)
61     {
62         if (mw.getPower() > 0)
63         {
64             mw.setPower(mw.getPower() - 10); // Decrement on 10w the power as said in interface
65             mw.getDisplayComponent().setDisplay(Integer.toString(mw.getPower()));
66         }
67         if (mw.getPower() == 0)
68         {
69             cooking_stop(mw);
70         }
71     }
72 }
```

```
73     @Override
74     public void power_reset(Microwave mw)
75     {
76         mw.setState(new Mw_ClosedWithItem(mw));
77         mw.setPower(0);
78         mw.getDisplayComponent().setDisplay(Integer.toString(mw.getPower()));
79     }
80
81     @Override
82     public void timer_inc(Microwave mw)
83     {
84         mw.setTime(mw.getTime() + 1);
85         mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
86     }
87
88     @Override
89     public void timer_dec(Microwave mw)
90     {
91         if (mw.getTime() > 0)
92         {
93             mw.setTime(mw.getTime() - 1);
94             mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
95         }
96         if (mw.getTime() == 0)
97         {
98             mw.getBeepComponent().beep(3);
99             mw.getDisplayComponent().setDisplay("Item ready");
100             cooking_stop(mw);
101         }
102     }
103
104     @Override
105     public void timer_reset(Microwave mw)
106     {
107         mw.setState(new Mw_ClosedWithItem(mw));
108         mw.setTime(0);
109     }
110
111
112     @Override
113     public void cooking_start(Microwave mw)
114     {
115         // Exception would be manage - Message written in English don't show to the users
116         throw new IllegalStateException("You cant start cooking if the microwave was already cooking ...");
117     }
118
119     @Override
120     public void cooking_stop(Microwave mw)
121     {
122         mw.setState(new Mw_ClosedWithItem(mw));
123     }
124
125     @Override
126     public void tick(Microwave mw)
127     {
128         if (mw.getTime() > 1)
129         {
130             mw.timer_dec();
131             try
132             {
133                 Thread.sleep(1); // TODO Implement real tick in the GUI
134             }
135             catch (InterruptedException e)
136             {
137                 e.printStackTrace();
138             }
139         }
140         else
141         {
142             mw.timer_dec();
143             mw.getBeepComponent().beep(3);
144             mw.getDisplayComponent().setDisplay("Item ready");
145             cooking_stop(mw);
146         }
147     }
148 }
```

Apartado B

Definir pruebas unitarias con Junit para cada uno de los componentes que conforman el sistema;

Mi implementación de los tests con Junit está en la clase [MicrowaveTest](#)

```
360 lines (296 sloc) | 8.82 KB
Raw Blame

1 package microwave_Junit_Test;
2
3 import org.junit.jupiter.api.Test;
4 import static org.junit.jupiter.api.Assertions.*;
5
6 import microwave.*;
7
8 public class MicrowaveTest
9 {
10     private Microwave mw = new Microwave();
11
12     //COMPONENTS TESTS
13
14     // Heating component test
15     @Test
16     public void MagnetronTest()
17     {
18         Heating h = new Heating();
19
20         // Base
21         assertEquals(0, h.getPower());
22         assertEquals(false, h.isHeating());
23
24         // On - Off
25         h.heating_on();
26         assertEquals(true, h.isHeating());
27         h.heating_off();
28         assertEquals(false, h.isHeating());
29
30         // Increase Power
31         h.setPower(888);
32         assertEquals(h.getPower(), 888);
33         h.setPower(0);
34         assertEquals(h.getPower(), 0);
35
36     }
37
38     // Lamp component test
39     @Test
40     public void lumixTest()
41     {
42         Lamp l = new Lamp();
43
44         // Base
45         assertFalse(l.isLampOn());
46
47         // On - Off
48         assertFalse(l.isLampOn());
49         l.lamp_on();
50         assertTrue(l.isLampOn());
51         l.lamp_off();
52         assertFalse(l.isLampOn());
53     }
```

```
54
55 // Turntable component test
56 @Test
57 public void rotationTest()
58 {
59     Turntable t = new Turntable();
60
61     // Base
62     assertFalse(t.isMoving());
63
64     // Start - Stop
65     t.turntable_start();
66     assertTrue(t.isMoving());
67     t.turntable_stop();
68     assertFalse(t.isMoving());
69 }
70
71 // Beeper component test
72 @Test
73 public void beeperTest()
74 {
75     Beeper b = new Beeper();
76
77     // Beep
78     b.beep(5);
79     assertTrue(BeeperCounter.isBeeped(5));
80
81     // No Beep
82     assertTrue(BeeperCounter.isBeeped(0));
83 }
84
85 // Display component test
86 @Test
87 public void displayTest()
88 {
89     Display d = new Display();
90
91     // Base
92     assertNull(d.getDisplay());
93
94     // Set - Clear
95     d.setDisplay("Test");
96     assertEquals("Test", d.getDisplay());
97     d.clearDisplay();
98     assertNull(d.getDisplay());
99 }
100
101 /** Metod to simulate the increase of the microwave power
102  * @param time - Amount of time to increase (Works fine with mult of 10)
103  * */
104 private void increment_power(int p)
105 {
106     for (int i = 0; i < p; i = i + 10) // The increment is 10 by 10
107     {
108         mw.power_inc();
109     }
110 }
111
```

```
112     /** Metod to simulate the decrease of the microwave power
113      * @param time - Amount of time to decrease (Works fine with mult of 10)
114      * */
115     private void decrease_power(int p)
116     {
117         for (int i = p; i > 0; i = i - 10) // The decrement is 10 by 10
118         {
119             mw.power_dec();
120         }
121     }
122
123     /** Metod to simulate the action of increment the time remaining
124     * @param time - Amount of time to increase
125     * */
126     private void timer_inc(int time)
127     {
128         for (int i = 0; i < time; i++)
129         {
130             mw.timer_inc();
131         }
132     }
133
134     /** Metod to simulate the action of decrease the time remaining
135     * @param time - Amount of time to decrease
136     * */
137     private void timer_dec(int time)
138     {
139         for (int i = 0; i < time; i++)
140         {
141             mw.timer_dec();
142         }
143     }
144
145     /** Metod to simulate the action of time
146     * @param time - Amount of time passed
147     * */
148     private void timer_works(int time)
149     {
150         for (int i = 0; i < time; i++)
151         {
152             mw.tick();
153         }
154     }
155
```

```
156     // MICROWAVE TESTS
157
158     // Test for power
159     @Test
160     public void testPower()
161     {
162         // Start
163         mw.power_reset();
164         assertEquals(0, mw.getPower());
165
166         // Increase - Decrease
167         increment_power(80);
168         assertEquals(mw.getPower(), 80);
169         assertEquals(mw.getDisplayComponent().getDisplay(), "80");
170         decrease_power(100);
171         assertEquals(mw.getPower(), 0);
172
173         // End
174         mw.power_reset();
175         assertEquals(mw.getPower(), 0);
176         assertEquals(mw.getDisplayComponent().getDisplay(), "0");
177     }
178
```

```
179 // Test for timer
180 @Test
181 public void testTimer()
182 {
183     // Start
184     mw.timer_reset();
185     assertEquals(0, mw.getTime());
186
187     // Increase - Decrease
188     timer_inc(80);
189     assertEquals(80, mw.getTime());
190     assertEquals("80", mw.getDisplayComponent().getDisplay());
191     timer_dec(35);
192     assertEquals(45, mw.getTime());
193     assertEquals("45", mw.getDisplayComponent().getDisplay());
194
195     // End
196     mw.timer_reset();
197     assertEquals(mw.getTime(), 0);
198     assertEquals(mw.getDisplayComponent().getDisplay(), "0");
199 }
200
201 // State 1: ClosedWithNoItem
202 @Test
203 public void closedWithNoItemTest()
204 {
205     mw.setState(new MW_ClosedWithNoItem(mw));
206     // Exceptions check
207     assertThrows(IllegalStateException.class, () -> mw.item_placed());
208     assertThrows(IllegalStateException.class, () -> mw.item_removed());
209     assertThrows(IllegalStateException.class, () -> mw.door_closed());
210     assertThrows(IllegalStateException.class, () -> mw.cooking_start());
211     assertThrows(IllegalStateException.class, () -> mw.cooking_stop());
212
213     // Status check
214     assertEquals(false, mw.isCooking());
215     assertEquals(false, mw.isItem());
216     assertEquals(false, mw.isOpen());
217     assertEquals(false, mw.getHeatComponent().isHeating());
218     assertEquals(false, mw.getLampComponent().isLampOn());
219     assertEquals(false, mw.getTurntComponent().isMoving());
220     assertEquals(true, mw.getState() instanceof MW_ClosedWithNoItem);
221 }
222
```

```
223
224 // State 2: OpenWithNoItem
225 @Test
226 public void openWithNoItemTest()
227 {
228     mw.setState(new MW_OpenWithNoItem(mw));
229
230     // Exceptions check
231     assertThrows(IllegalStateException.class, () -> mw.item_removed());
232     assertThrows(IllegalStateException.class, () -> mw.cooking_start());
233     assertThrows(IllegalStateException.class, () -> mw.door_opened());
234     assertThrows(IllegalStateException.class, () -> mw.cooking_stop());
235
236     // Status check
237     assertEquals(false, mw.isCooking());
238     assertEquals(false, mw.isItem());
239     assertEquals(true, mw.isOpen());
240     assertEquals(false, mw.getHeatComponent().isHeating());
241     assertEquals(true, mw.getLampComponent().isLampOn());
242     assertEquals(false, mw.getTurntComponent().isMoving());
243     assertEquals(true, mw.getState() instanceof MW_OpenWithNoItem);
244 }
245
246
```

```
247 // Phase 3: Test for an OpenWithItem situation
248 @Test
249 public void openWithItemTest()
250 {
251     mw.setState(new MW_OpenWithItem(mw));
252
253     // Exceptions check
254     assertThrows(IllegalStateException.class, () -> mw.item_placed());
255     assertThrows(IllegalStateException.class, () -> mw.cooking_start());
256     assertThrows(IllegalStateException.class, () -> mw.door_opened());
257     assertThrows(IllegalStateException.class, () -> mw.cooking_stop());
258
259
260     // Status check
261     assertEquals(false, mw.isCooking());
262     assertEquals(true, mw.isItem());
263     assertEquals(true, mw.isOpen());
264     assertEquals(false, mw.getHeatComponent().isHeating());
265     assertEquals(true, mw.getLampComponent().isLampOn());
266     assertEquals(false, mw.getTurntComponent().isMoving());
267     assertEquals(true, mw.getState() instanceof MW_OpenWithItem);
268
269     // Removing item
270     mw.item_removed();
271     assertEquals(mw.getState().getClass(), MW_OpenWithNoItem.class);
272
273 }
274
275 // Phase 4: Test for a ClosedWithItem situation
276 @Test
277 public void closedWithItemTest()
278 {
279     mw.setState(new MW_ClosedWithItem(mw));
280
281     // Exceptions check
282     assertThrows(IllegalStateException.class, () -> mw.item_placed());
283     assertThrows(IllegalStateException.class, () -> mw.item_removed());
284     assertThrows(IllegalStateException.class, () -> mw.door_closed());
285     assertThrows(IllegalStateException.class, () -> mw.cooking_stop());
286
287
288     // Status check
289     assertEquals(false, mw.isCooking());
290     assertEquals(true, mw.isItem());
291     assertEquals(false, mw.isOpen());
292     assertEquals(false, mw.getHeatComponent().isHeating());
293     assertEquals(false, mw.getLampComponent().isLampOn());
294     assertEquals(false, mw.getTurntComponent().isMoving());
295     assertEquals(true, mw.getState() instanceof MW_ClosedWithItem);
296
297     // Opening door
298     mw.door_opened();
299     assertEquals(mw.getState().getClass(), MW_OpenWithItem.class);
300 }
301
```



```
302 // Phase 5: Test for a Cooking situation
303 @Test
304 public void cookingTest()
305 {
306     mw.setState(new MW_Cooking(mw));
307
308     // Cooking with wrong inputs
309
310     // time == 0 & power == 0
311     mw.timer_reset();
312     assertThrows(IllegalStateException.class, () -> mw.cooking_start());
313
314     // time == 60 & power == 0
315     timer_inc(60);
316     assertThrows(IllegalStateException.class, () -> mw.cooking_start());
317
318     // time == 0 & power == 800
319     mw.timer_reset();
320     increment_power(800);
321     assertThrows(IllegalStateException.class, () -> mw.cooking_start());
322
323     // Start cooking
324     timer_inc(25);
325     increment_power(100);
326     mw.cooking_start();
327
328     // Exceptions check
329     assertThrows(IllegalStateException.class, () -> mw.cooking_start());
330     assertThrows(IllegalStateException.class, () -> mw.door_closed());
331
332     assertThrows(IllegalStateException.class, () -> mw.item_placed());
333     assertThrows(IllegalStateException.class, () -> mw.item_removed());
334
335     // Status check
336     assertEquals(mw.isCooking(), true);
337     assertEquals(mw.isItem(), true);
338     assertEquals(mw.isOpen(), false);
339     assertEquals(mw.getHeatComponent().isHeating(), true);
340     assertEquals(mw.getLampComponent().isLampOn(), true);
341     assertEquals(mw.getTurntComponent().isMoving(), true);
342     assertEquals(mw.getState() instanceof MW_Cooking, true);
343
344     // Open door
345     mw.door_opened();
346     assertTrue(mw.getState() instanceof MW_OpenWithItem);
347     assertFalse(BeeperCounter.isBeeped(3));
348
349     mw.door_closed();
350     mw.cooking_start();
351
352     assertEquals(25, mw.getTime());
353     assertEquals(mw.getState().getClass(), MW_Cooking.class);
354
355     // Time ends
356     timer_works(25);
357     assertEquals("Item ready", mw.getDisplayComponent().getDisplay());
358     assertTrue(BeeperCounter.isBeeped(3));
359 }
360 }
```

Apartado C

Definir un conjunto de escenarios de prueba para el sistema completo con Gherkin, e implementarlas en Cucumber

Yo he decidido crear una feature por cada estado del microondas, y así poder aislar las distintas etapas para poder realizadas las pruebas de forma más ordenada. Están en la ruta resources de la parte de test del src ([Enlace](#) a la carpeta que los contienen en GitHub)

MW_ClosedWithNoItem.feature

```
23 lines (19 sloc) | 476 Bytes
Raw Blame

1 Feature: Testing first State
2
3 Scenario Outline: Setting power
4   Given Testing first State
5   When Set power to <T>
6   Then Screen value is "<D>"
7
8 Examples:
9   | T | D |
10  | 10 | 100 |
11  | 0 | 0 |
12  | -1 | 0 |
13
14 Scenario Outline: Setting timer
15   Given Testing first State
16   When We settle timer at <T> s
17   Then Screen value is "<D>"
18
19 Examples:
20   | T | D |
21   | -888 | 0 |
22   | 0 | 0 |
23   | 88 | 88 |
```

MW_OpenWithNoItem.feature

```
23 lines (18 sloc) | 459 Bytes
Raw Blame

1 Feature: Testing second State
2
3 Scenario Outline: Setting power
4   Given Testing second State
5   When Set power to <a>
6   Then Screen value is "<b>"
7
8 Examples:
9   | a | b |
10  | 0 | 0 |
11  | 8 | 80 |
12
13 Scenario Outline: Setting timer
14   Given Testing second State
15   When We settle timer at <a> s
16   Then Screen value is "<b>"
17
18 Examples:
19   | a | b |
20   | -1 | 0 |
21   | 0 | 0 |
22   | 60 | 60 |
23
```

MW_OpenWithItem.feature

```

22 lines (18 sloc) | 452 Bytes
Raw Blame

1 Feature: Testing third State
2
3 Scenario Outline: Setting power
4   Given Testing third State
5   When Set power to <a>
6   Then Screen value is "<b>"
7
8   Examples:
9     | a | b |
10    | 1 | 10 |
11    | 0 | 0 |
12
13 Scenario Outline: Setting timer
14   Given Testing third State
15   When We settle timer at <a> s
16   Then Screen value is "<b>"
17
18   Examples:
19     | a | b |
20     | -1 | 0 |
21     | 0 | 0 |
22     | 60 | 60 |

```

MW_ClosedWithItem.feature

```

30 lines (25 sloc) | 652 Bytes
Raw Blame

1 Feature: Testing fourth State
2
3 Scenario: Cook
4   Given Testing fourth State
5   When Set power to 10
6   And We settle timer at 90 s
7   And Press start cooking
8   Then We are in the fifth State
9
10 Scenario Outline: Setting power
11   Given Testing fourth State
12   When Set power to <T>
13   Then Screen value is "<D>"
14
15   Examples:
16     | T | D |
17     | 10 | 100 |
18     | 0 | 0 |
19     | -1 | 0 |
20
21 Scenario Outline: Setting timer
22   Given Testing fourth State
23   When We settle timer at <S> s
24   Then Screen value is "<D>"
25
26   Examples:
27     | S | D |
28     | 88 | 88 |
29     | 0 | 0 |
30     | -888 | 0 |

```

MW_Cooking.feature

```
38 lines (32 sloc) | 1.05 KB
Raw Blame

1 Feature: Testing fifth State
2
3 Scenario: Cook correctly
4   Given Testing fourth State
5   When Increase the power 10 times
6   And We settle timer at 60 s
7   And Press start cooking
8   Then We are in the fifth State
9   And Heating heats
10  And Lamp turns on
11  And Turntable turns
12
13 Scenario: Testing change while runing 1
14   Given A running microwave with 2000 power and 60 timer
15   When Increase the power 1 times
16   Then Screen value is "2010"
17
18 Scenario: Testing change while runing 2
19   Given A running microwave with 800 power and 1 timer
20   When We settle timer at 2 s
21   Then Screen value is "2"
22
23 Scenario: Testing change while runing 3
24   Given A running microwave with 800 power and 2 timer
25   When We settle timer at 1 s
26   Then Screen value is "1"
27
28 Scenario Outline: Testing Display
29   Given Testing fourth State
30   When We settle timer at <S> s
31   And Increase the power <T> times
32   Then Screen value is "<D>"
33
34 Examples:
35   | S | T | D |
36   | 15 | 63 | 630 |
37   | 30 | 75 | 750 |
38   | 100 | 89 | 890 |
```

StepDefinitions

Aquí he definido cada uno de los métodos que se invocarán con los escenarios
([Enlace](#) a GitHub del archivo)

120 lines (99 sloc) | 2.29 KB

Raw Blame

```
1 package microwave_Cucumber_Test;
2
3 import io.cucumber.java.en.*;
4
5 import static org.junit.jupiter.api.Assertions.*;
6
7 import microwave.*;
8
9 /**Steps definition stores the mapping between each step of the scenario defined in the feature file with a code of function to be executed*/
10 public class StepDefinitions
11 {
12     /**Our microwave instance that we are gona use for test*/
13     private Microwave mw = new Microwave();
14
15     // GIVEN
16
17     @Given("Testing first State")
18     public void first_State()
19     {
20         mw.setState(new Mmw_ClosedWithNoItem(mw));
21     }
22
23     @Given("Testing second State")
24     public void second_State()
25     {
26         mw.setState(new Mmw_OpenWithNoItem(mw));
27     }
28
29     @Given("Testing third State")
30     public void third_State()
31     {
32         mw.setState(new Mmw_OpenWithItem(mw));
33     }
34
35     @Given("Testing fourth State")
36     public void fourth_State()
37     {
38         mw.setState(new Mmw_ClosedWithItem(mw));
39     }
40
41     @Given("A running microwave with {int} power and {int} timer")
42     public void cooking(Integer power, Integer time)
43     {
44         mw.setState(new Mmw_ClosedWithItem(mw));
45         mw.setTime(time);
46         set_Power(power/10);
47         mw.cooking_start();
48     }
49 }
```

```

50      // WHEN
51
52      @When("Set power to {int}")
53      public void set_Power(Integer new_power)
54      {
55          mw.power_reset();
56          increase_Power(new_power);
57      }
58
59      @When("Increase the power {int} times")
60      public void increase_Power(Integer simulated_times)
61      {
62          for (int i = 0; i < simulated_times; i++)
63          {
64              mw.power_inc();
65          }
66      }
67
68      @When("We settle timer at {int} s")
69      public void setTimer(Integer times)
70      {
71          mw.setTime(times);
72          mw.timer_inc();
73          mw.timer_dec();
74      }
75
76      @When("Press start cooking")
77      public void try_to_cook()
78      {
79          try
80          {
81              mw.cooking_start();
82          }
83          catch (IllegalStateException e)
84          {
85              assertEquals(false, true);
86          }
87      }
88
89      // THEN
90
91      @Then("Heating heats")
92      public void magnetronWorking()
93      {
94          assertEquals(mw.getHeatComponent().isHeating(), true);
95      }
96
97      @Then("Lamp turns on")
98      public void lampIsOn()
99      {
100          assertEquals(mw.getLampComponent().isLampOn(), true);
101      }
102
103      @Then("Turntable turns")
104      public void turntableIsTurning()
105      {
106          assertEquals(mw.getTurntComponent().isMoving(), true);
107      }
108
109      @Then("Screen value is {string}")
110      public void screenShows(String i)
111      {
112          assertEquals(mw.getDisplayComponent().getDisplay(), i);
113      }
114
115      @Then("We are in the fifth State")
116      public void microwaveCooking()
117      {
118          assertEquals(mw.getState().getClass(), MW_Cooking.class);
119      }
120  }

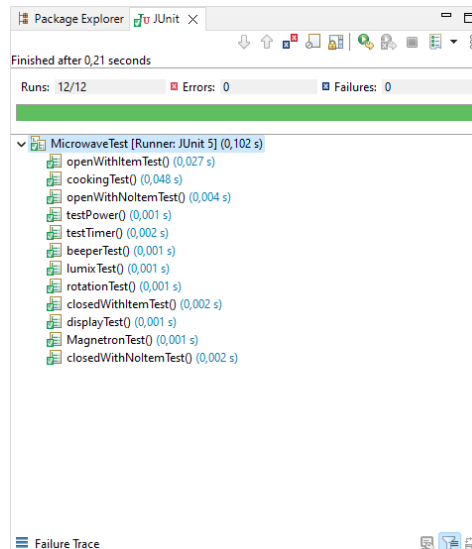
```

Apartado D

Definir e implementar tres interfaces de usuario que, a través de botones, permitan interactuar con el microondas de forma concurrente: uno con el panel de control, otro que simule la puerta y el hecho de meter y sacar un alimento del microondas, y un tercero que permita simular el tick de reloj.

Resultados

Test con Junit



Test con Cucumber

```
[INFO] Tests run: 30, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.763 s - in microwave_Cucumber_Test.RunCucumberTest
[INFO] Running microwave_Junit_Test.MicrowaveTest
[INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.153 s - in microwave_Junit_Test.MicrowaveTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 42, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.324 s
[INFO] Finished at: 2022-06-10T21:27:13+02:00
[INFO] -----
```

Conclusión

El patrón de diseño Estado se adapta muy bien para este proyecto (ya que el comportamiento del microondas cambia dependiendo de su estado) y gracias a las pruebas realizadas con Gherkin-Cucumber en este proyecto me han quedado más claro en qué situaciones se pueden usar estas herramientas.