



UNIVERSIDAD
DE MÁLAGA



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Práctica Final:

Microondas

Realizado por
Florín Babusca Voicu

UNIVERSIDAD DE MÁLAGA
MÁLAGA, MAYO DE 2022

Apartado a

Para realizar la implementación del microondas, se ha usado, como está pedido en la práctica, el patrón estado para implementar los diferentes estados del microondas (si está cocinando, cerrado...).

Se han implementado todos los métodos especificados en el diagrama UML del enunciado. No obstante, se han tenido que implementar getters y setters en la clase Microwave o bien porque es necesario que los estados del microondas dispongan de algún mecanismo que modifiquen atributos del microondas o bien porque es necesario acceder a estos en los tests para comprobar que el programa funciona correctamente.

Se ha intentado que todos los getters y setters tengan la menor visibilidad posible, pero debido a la implementación de tests con Gherkin, ha sido necesario establecer ciertos getters públicos para poder comprobar que el microondas realiza la tarea especificada en los escenarios correctamente.

Los principales retos a la hora de implementar este proyecto han sido:

Decisión de la interfaz de los estados del microondas

Para determinar los métodos los cuales serán implementados en los estados del microondas, se ha analizado minuciosamente el diagrama de estados, el cual ha sido clave para comprender qué métodos deben ser elegidos. Finalmente, se han escogido los siguientes:

```
1 package microwaveEngine;
2
3 public interface MicrowaveState {
4     public void door_opened(Microwave m);
5
6     public void door_closed(Microwave m);
7
8     public void item_placed(Microwave m);
9
10    public void item_removed(Microwave m);
11
12    public void cooking_start(Microwave m);
13
14    public void cooking_stop(Microwave m);
15
16    public void timer_reset(Microwave m);
17
18    public void power_reset(Microwave m);
19
20    public void tick(Microwave m);
21
22    public void power_desc(Microwave m);
23
24    public void timer_desc(Microwave m);
25 }
26
```

Se han elegido porque, si analizamos detenidamente cada uno de ellos, son los que están relacionados a algún estado en concreto del microondas. Por ejemplo, `door_opened()` alterará el estado de distintas formas, como cuando estamos cocinando y de repente se llama a dicho método, o el método `tick()`, el cual solo tendrá sentido ser implementado cuando el microondas se encuentre cocinando un alimento.

Podría sorprender la elección de los métodos `power_desc()`, `timer_desc()` ya que presentan, a priori, un carácter intrínseco al propio microondas y no tanto a sus estados. No obstante, estos métodos actúan diferentemente cuando el microondas está cocinando, ya que si la potencia o el tiempo tiene un valor de 1 y se desea reducir a estos métodos, el microondas cambiará de estado y será este mismo (estado cocinando) quien decida a qué estado debe cambiar y cómo deberá hacerlo: si la potencia se establece en 0, el microondas parará de cocinar y mostrará en pantalla la potencia nueva establecida 0, o bien, si el tiempo se establece a 0, el microondas parará de cocinar y actuará como una finalización del cocinado normal del microondas (mostrando por pantalla “Food is ready”).

La razón es similar para `power_reset()` y para `timer_reset()`.

Cabe destacar que estos métodos no realizan operaciones de interés en todos los estados sino que levantan una `IllegalStateException()` cuando no deban ser llamados, ya que no es posible, por ejemplo, insertar comida en un microondas con la puerta cerrada.

Representación del sonido de la bocina

Ha sido bastante interesante determinar cómo representar el sonido del beeper (bocina) del microondas cuando este ha terminado de cocinar. Se ha optado usar el **Patrón Observador** y por ende, se ha creado una clase pública llamada **BeeperListener** (**vista**) con un método publico estático llamado `notify(int i)`, el cual invocará el método `beep(int t)` de la clase **Beeper** (**modelo**) y le pasará como argumento el número de veces que este emite un sonido (el valor t). Este valor será almacenado en la clase BeeperListener y cuando esta reciba una llamada su método `hasTheBeeperSound(int times)`, devolverá un booleano comprobando si el valor especificado coincide con el número de veces que ha sonado el beeper. Una vez realizado esto, el contador interno de BeeperListener se pondrá a cero para garantizar que cuando se llame al método `hasTheBeeperSound`, este no compare con un valor residual de otra cocción realizada anteriormente.

```
1 package microwaveEngine;
2
3 public class BeeperListener {
4     private static int timesSound = 0;
5     static void notify(int timesBeeperSounds) {
6         timesSound = timesBeeperSounds;
7     }
8     /**
9      * Compares the amount of beeps the beeper has made with
10     * the expected amount of beeps and resets the counted times it has sound.
11     * @param times - Amount of beeps expected
12     * @return If the times expected matches or not.
13     */
14     public static boolean hasTheBeeperSound(int times) {
15         int copy = timesSound;
16         timesSound = 0;
17         return (copy == times);
18     }
19 }
20 }
21 }
```

La implementación del microondas es:

Nota: Se ha asumido que un microondas nuevo es un microondas cerrado sin alimentos y con temporizador y potencia a cero

```
1 package microwaveEngine;
2
3 public class Microwave {
4     private boolean doorOpen;
5     private int power;
6     private int timer;
7     private boolean cooking;
8     private boolean withItem;
9     private MicrowaveState state;
10    private Heating heatingConnection = new Heating();
11    private Lamp lampConnection = new Lamp();
12    private Turnable turnableConnection = new Turnable();
13    private Beeper beeperConnection = new Beeper();
14    private Display displayConnection = new Display();
15
16    public Microwave() {
17        cooking = false;
18        withItem = false;
19        doorOpen = false;
20        power = 0;
21        timer = 0;
22        state = new ClosedWithNoItem(this);
23    }
24
25    public void door_opened() {
26        state.door_opened(this);
27    }
28
29    public void door_closed() {
30        state.door_closed(this);
31    }
32
33    public void item_placed() {
34        state.item_placed(this);
35    }
36
37    public void item_removed() {
38        state.item_removed(this);
39    }
40
41    public void power_inc() {
42        power++;
43        displayConnection.setDisplay(Integer.toString(power));
44    }
45
46    public void power_desc() {
47        state.power_desc(this);
48    }
49
50    public void power_reset() {
51        state.power_reset(this);
52        displayConnection.setDisplay(Integer.toString(power));
53    }
54
55    public void timer_inc() {
56        timer++;
57        displayConnection.setDisplay(Integer.toString(timer));
58    }
```

```
59
60 public void timer_desc() {
61     state.timer_desc(this);
62 }
63
64 public void timer_reset() {
65     state.timer_reset(this);
66     displayConnection.setDisplay(Integer.toString(timer));
67 }
68
69 public void cooking_start() {
70     state.cooking_start(this);
71 }
72
73 public void cooking_stop() {
74     state.cooking_stop(this);
75 }
76
77 public void tick() {
78     state.tick(this);
79 }
80
81 // GETTERS Y SETTERS
82
83 public boolean isDoorOpen() {
84     return doorOpen;
85 }
86
87 void setDoorOpen(boolean doorOpen) {
88     this.doorOpen = doorOpen;
89 }
90
91 int getPower() {
92     return power;
93 }
94
95 void setPower(int power) {
96     this.power = power;
97 }
98
99 int getTimer() {
100     return timer;
101 }
102
103 void setTimer(int timer) {
104     this.timer = timer;
105 }
106
107 public boolean isCooking() {
108     return cooking;
109 }
110
111 void setCooking(boolean cooking) {
112     this.cooking = cooking;
113 }
114
115 public boolean isWithItem() {
116     return withItem;
117 }
118
```

```

119- void setWithItem(boolean withItem) {
120     this.withItem = withItem;
121 }
122
123- MicrowaveState getState() {
124     return state;
125 }
126
127- void setState(MicrowaveState state) {
128     this.state = state;
129 }
130
131- public Heating getHeatingConnection() {
132     return heatingConnection;
133 }
134
135- public Lamp getLampConnection() {
136     return lampConnection;
137 }
138
139- public Turnable getTurnableConnection() {
140     return turnableConnection;
141 }
142
143- Beeper getBeeperConnection() {
144     return beeperConnection;
145 }
146
147- public Display getDisplayConnection() {
148     return displayConnection;
149 }
150
151 }
152

```

La implementación de la clase Display es:

Nota: Todos los componentes del microondas (luz, plato, motor, bocina y la pantalla) se ha considerado razonable que cuando se creen estén todos apagados.

```

1 package microwaveEngine;
2
3 public class Display {
4     private String display;
5
6- void clearDisplay() {
7     display = null;
8 }
9
10- void setDisplay(String s) {
11     display = s;
12 }
13
14- public String getDisplay() {
15     return display;
16 }
17 }
18

```

La implementación de la clase Beeper es:

```

1 package microwaveEngine;
2
3 public class Beeper {
4- void beep(int d) {
5     BeeperListener.notify(d);
6 }
7 }
8

```

La implementación de la clase Heating es:

```
1 package microwaveEngine;
2
3 public class Heating {
4     private boolean heating = false;
5     private int power = 0;
6
7     void heatingOn() {
8         heating = true;
9     }
10
11    void heatingOff() {
12        heating = false;
13    }
14
15    void setPower(int power) {
16        if (power >= 0) {
17            this.power = power;
18        }
19    }
20
21    int getPower() {
22        return power;
23    }
24
25    public boolean isHeating() {
26        return heating;
27    }
28 }
29
```

La implementación de la clase Lamp es:

```
1 package microwaveEngine;
2
3 public class Lamp {
4     private boolean lampOn = false;
5
6     void lampOn() {
7         lampOn = true;
8     }
9
10    void lampOff() {
11        lampOn = false;
12    }
13
14    public boolean isLampOn() {
15        return lampOn;
16    }
17 }
18
```

La implementación de la clase Turnable es:

```
1 package microwaveEngine;
2
3 public class Turnable {
4     private boolean turnableOn = false;
5
6     void turnable_start() {
7         turnableOn = true;
8     }
9
10    void turnable_stop() {
11        turnableOn = false;
12    }
13
14    public boolean isMoving() {
15        return turnableOn;
16    }
17 }
18
```

La implementación de la clase que define el estado del microondas cerrado sin items dentro es:

```
1 package microwaveEngine;
2
3 public class ClosedWithNoItem implements MicrowaveState {
4     public ClosedWithNoItem(Microwave m) {
5         m.getHeatingConnection().heatingOff();
6         m.getLampConnection().lampOff();
7         m.getTurnableConnection().turnable_stop();
8         m.getDisplayConnection().clearDisplay();
9         m.setCooking(false);
10        m.setWithItem(false);
11        m.setDoorOpen(false);
12    }
13
14    @Override
15    public void door_opened(Microwave m) {
16        m.setState(new OpenWithNoItem(m));
17    }
18
19    @Override
20    public void door_closed(Microwave m) {
21        // Invalid action. It will do nothing.
22        throw new IllegalStateException("Error: Door already closed");
23    }
24
25    @Override
26    public void item_placed(Microwave m) {
27        // Invalid action. It will do nothing.
28        throw new IllegalStateException("Error: Door closed");
29    }
30
31    @Override
32    public void item_removed(Microwave m) {
33        // Invalid action. It will do nothing.
34        throw new IllegalStateException("Error: Door closed");
35    }
36
37    @Override
38    public void cooking_start(Microwave m) {
39        // Invalid action. It will do nothing.
40        throw new IllegalStateException("Error: Microwave does not have food");
41    }
42
43    @Override
44    public void cooking_stop(Microwave m) {
45        // Invalid action. It will do nothing.
46        throw new IllegalStateException("Error: Microwave was not cooking");
47    }
48
49    @Override
50    public void tick(Microwave m) {
51        // Invalid action. It will do nothing.
52        throw new IllegalStateException("Error: Microwave is not cooking");
53    }
54
55    @Override
56    public void timer_reset(Microwave m) {
57        m.setTimer(0);
58    }
59 }
60
```



```

61 @Override
62 public void power_reset(Microwave m) {
63     m.setPower(0);
64 }
65
66 @Override
67 public void power_desc(Microwave m) {
68     if (m.getPower() > 0) {
69         m.setPower(m.getPower() - 1);
70         m.getDisplayConnection().setDisplay(Integer.toString(m.getPower()));
71     }
72 }
73
74 @Override
75 public void timer_desc(Microwave m) {
76     if (m.getTimer() > 0) {
77         m.setTimer(m.getTimer() - 1);
78         m.getDisplayConnection().setDisplay(Integer.toString(m.getTimer()));
79     }
80 }
81 }
82 }
83

```

La implementación de la clase que define el estado del microondas abierto sin items dentro es:

```

1 package microwaveEngine;
2
3 public class OpenWithNoItem implements MicrowaveState {
4
5     public OpenWithNoItem(Microwave m) {
6         m.getLampConnection().lampOn();
7         m.getHeatingConnection().heatingOff();
8         m.getTurnableConnection().turnable_stop();
9         m.setCooking(false);
10        m.setWithItem(false);
11        m.setDoorOpen(true);
12    }
13
14    @Override
15    public void door_opened(Microwave m) {
16        // Invalid action. It will do nothing.
17        throw new IllegalStateException("Error: Door already opened");
18    }
19
20    @Override
21    public void door_closed(Microwave m) {
22        m.setState(new ClosedWithNoItem(m));
23    }
24
25    @Override
26    public void item_placed(Microwave m) {
27        m.setState(new OpenWithItem(m));
28    }
29
30    @Override
31    public void item_removed(Microwave m) {
32        // Invalid action. It will do nothing.
33        throw new IllegalStateException("Error: You cannot remove an item from an empty microwave");
34    }
35
36    @Override
37    public void cooking_start(Microwave m) {
38        // Invalid action. It will do nothing.
39        throw new IllegalStateException("Error: You cannot start cooking with the door opened");
40    }
41

```

```

41 |
42 | @Override
43 | public void cooking_stop(Microwave m) {
44 |     // Invalid action. It will do nothing.
45 |     throw new IllegalStateException("Error: Microwave is not cooking");
46 | }
47 |
48 | @Override
49 | public void tick(Microwave m) {
50 |     // Invalid action. It will do nothing.
51 |     throw new IllegalStateException("Error: Microwave is not cooking");
52 | }
53 |
54 | @Override
55 | public void timer_reset(Microwave m) {
56 |     m.setTimer(0);
57 | }
58 |
59 |
60 | @Override
61 | public void power_reset(Microwave m) {
62 |     m.setPower(0);
63 | }
64 |
65 |
66 | @Override
67 | public void power_desc(Microwave m) {
68 |     if (m.getPower() > 0) {
69 |         m.setPower(m.getPower() - 1);
70 |         m.getDisplayConnection().setDisplay(Integer.toString(m.getPower()));
71 |     }
72 | }
73 |
74 | @Override
75 | public void timer_desc(Microwave m) {
76 |     if (m.getTimer() > 0) {
77 |         m.setTimer(m.getTimer() - 1);
78 |         m.getDisplayConnection().setDisplay(Integer.toString(m.getTimer()));
79 |     }
80 | }
81 | }
82 |
83 | }
84 |

```

La implementación de la clase que define el estado del microondas abierto con comida dentro es:

```

1 | package microwaveEngine;
2 |
3 | public class OpenWithItem implements MicrowaveState {
4 |
5 |     public OpenWithItem(Microwave m) {
6 |         m.getLampConnection().lampOn();
7 |         m.getHeatingConnection().heatingOff();
8 |         m.getTurnableConnection().turnable_stop();
9 |         m.setCooking(false);
10 |        m.setWithItem(true);
11 |        m.setDoorOpen(true);
12 |    }
13 |
14 |    @Override
15 |    public void door_opened(Microwave m) {
16 |        // Invalid action. It will do nothing.
17 |        throw new IllegalStateException("Error: Door already opened");
18 |    }
19 |
20 |    @Override
21 |    public void door_closed(Microwave m) {
22 |        m.setState(new ClosedWithItem(m));
23 |    }
24 |

```

```

25 @Override
26 public void item_placed(Microwave m) {
27     // Invalid action. It will do nothing.
28     throw new IllegalStateException("Error: Microwave is full");
29 }
30
31 @Override
32 public void item_removed(Microwave m) {
33     m.setState(new OpenWithNoItem(m));
34 }
35
36 @Override
37 public void cooking_start(Microwave m) {
38     // Invalid action. It will do nothing.
39     throw new IllegalStateException("Error: You cannot start cooking with the door opened");
40 }
41
42 @Override
43 public void cooking_stop(Microwave m) {
44     // Invalid action. It will do nothing.
45     throw new IllegalStateException("Error: Microwave is not cooking");
46 }
47
48 @Override
49 public void timer_reset(Microwave m) {
50     m.setTimer(0);
51 }
52 }
53
54 @Override
55 public void power_reset(Microwave m) {
56     m.setPower(0);
57 }
58 }
59
60 @Override
61 public void tick(Microwave m) {
62     // Invalid action. It will do nothing.
63     throw new IllegalStateException("Error: Microwave is not cooking");
64 }
65
66 @Override
67 public void power_desc(Microwave m) {
68     if (m.getPower() > 0) {
69         m.setPower(m.getPower() - 1);
70         m.getDisplayConnection().setDisplay(Integer.toString(m.getPower()));
71     }
72 }
73
74 @Override
75 public void timer_desc(Microwave m) {
76     if (m.getTimer() > 0) {
77         m.setTimer(m.getTimer() - 1);
78         m.getDisplayConnection().setDisplay(Integer.toString(m.getTimer()));
79     }
80 }
81 }
82
83 }
84

```

La implementación de la clase que define el estado del microondas cerrado con comida dentro es:

```
1 package microwaveEngine;
2
3 public class ClosedWithItem implements MicrowaveState {
4
5     public ClosedWithItem(Microwave m) {
6         m.getLampConnection().lampOff();
7         m.getHeatingConnection().heatingOff();
8         m.getTurnableConnection().turnable_stop();
9         m.setCooking(false);
10        m.setDoorOpen(false);
11        m.setWithItem(true);
12    }
13
14    @Override
15    public void door_opened(Microwave m) {
16        m.setState(new OpenWithItem(m));
17    }
18
19    @Override
20    public void door_closed(Microwave m) {
21        // Invalid action. It will do nothing.
22        throw new IllegalStateException("Error: Door already closed");
23    }
24
25    @Override
26    public void item_placed(Microwave m) {
27        // Invalid action. It will do nothing.
28        throw new IllegalStateException("Error: Door closed");
29    }
30
31    @Override
32    public void item_removed(Microwave m) {
33        // Invalid action. It will do nothing.
34        throw new IllegalStateException("Error: Door closed");
35    }
36
37    @Override
38    public void cooking_start(Microwave m) {
39        if(m.getTimer() > 0 && m.getPower() > 0) {
40            m.setState(new Cooking(m));
41        } else if (m.getTimer() > 0) {
42            throw new IllegalStateException("Error: Timer is 0");
43        } else {
44            throw new IllegalStateException("Error: Power is 0");
45        }
46    }
47
48    @Override
49    public void cooking_stop(Microwave m) {
50        // Invalid action. It will do nothing.
51        throw new IllegalStateException("Error: Microwave is not cooking");
52    }
53
54    @Override
55    public void timer_reset(Microwave m) {
56        m.setTimer(0);
57    }
58
59 }
```

```

60 @Override
61 public void power_reset(Microwave m) {
62     m.setPower(0);
63 }
64
65
66 @Override
67 public void tick(Microwave m) {
68     // Invalid action. It will do nothing.
69     throw new IllegalStateException("Error: Microwave is not cooking");
70 }
71
72 @Override
73 public void power_desc(Microwave m) {
74     if (m.getPower() > 0) {
75         m.setPower(m.getPower() - 1);
76         m.getDisplayConnection().setDisplay(Integer.toString(m.getPower()));
77     }
78 }
79
80 @Override
81 public void timer_desc(Microwave m) {
82     if (m.getTimer() > 0) {
83         m.setTimer(m.getTimer() - 1);
84         m.getDisplayConnection().setDisplay(Integer.toString(m.getTimer()));
85     }
86 }
87 }
88
89 }
90

```

La implementación de la clase que define el estado del microondas cuando está cocinando es:

```

1 package microwaveEngine;
2
3 public class Cooking implements MicrowaveState{
4     public Cooking(Microwave m) {
5         m.getLampConnection().lampOn();
6         m.getHeatingConnection().setPower(m.getPower());
7         m.getHeatingConnection().heatingOn();
8         m.getTurnableConnection().turnable_start();
9         m.setCooking(true);
10        m.setDoorOpen(false);
11        m.setWithItem(true);
12    }
13    @Override
14    public void door_opened(Microwave m) {
15        m.setState(new OpenWithItem(m));
16    }
17
18    @Override
19    public void door_closed(Microwave m) {
20        // Invalid action. It will do nothing.
21        throw new IllegalStateException("Error: Door already closed");
22    }
23
24    @Override
25    public void item_placed(Microwave m) {
26        // Invalid action. It will do nothing.
27        throw new IllegalStateException("Error: Door closed");
28    }
29

```

```

30- @Override
31 public void item_removed(Microwave m) {
32     // Invalid action. It will do nothing.
33     throw new IllegalStateException("Error: Door closed");
34 }
35
36- @Override
37 public void cooking_start(Microwave m) {
38     // Invalid action. It will do nothing.
39     throw new IllegalStateException("Error: Microwave is already cooking");
40 }
41
42- @Override
43 public void cooking_stop(Microwave m) {
44     m.setState(new ClosedWithItem(m));
45 }
46
47- @Override
48 public void timer_reset(Microwave m) {
49     m.setState(new ClosedWithItem(m));
50     m.setTimer(0);
51 }
52
53- @Override
54 public void power_reset(Microwave m) {
55     m.setState(new ClosedWithItem(m));
56     m.setPower(0);
57 }
58
59- @Override
60 public void tick(Microwave m) {
61     if (m.getTimer() > 1) {
62         m.timer_desc();
63     } else {
64         m.timer_desc();
65         m.getBeeperConnection().beep(3);
66         m.getDisplayConnection().setDisplay("Food is ready");
67         cooking_stop(m);
68     }
69 }
70
71
72- @Override
73 public void power_desc(Microwave m) {
74     if (m.getPower() > 0) {
75         m.setPower(m.getPower() - 1);
76         m.getDisplayConnection().setDisplay(Integer.toString(m.getPower()));
77     } if (m.getPower() == 0) {
78         cooking_stop(m);
79     }
80 }
81

```

```

82 @Override
83 public void timer_desc(Microwave m) {
84     if (m.getTimer() > 0) {
85         m.setTimer(m.getTimer() - 1);
86         m.getDisplayConnection().setDisplay(Integer.toString(m.getTimer()));
87     } if (m.getTimer() == 0) {
88         m.getBeeperConnection().beep(3);
89         m.getDisplayConnection().setDisplay("Food is ready");
90         cooking_stop(m);
91     }
92 }
93
94 }
95

```

Apartado b

La implementación de los tests de JUnit es:

```

1 package microwaveEngine;
2
3 import org.junit.jupiter.api.*;
4
5 class MicrowaveEngineTest {
6
7     private Microwave m = new Microwave();
8
9     @Test
10    void heatingTest() {
11        Heating h = new Heating();
12
13        // Comprobamos que se inicializa correctamente
14        Assertions.assertEquals(0, h.getPower());
15        Assertions.assertFalse(h.isHeating());
16
17        // Comprobamos que se establece correctamente una potencia determinada
18        h.setPower(100);
19        Assertions.assertEquals(100, h.getPower());
20
21        // Comprobamos que se enciende y se apaga correctamente
22        h.heatingOn();
23        Assertions.assertTrue(h.isHeating());
24        h.heatingOn();
25        Assertions.assertTrue(h.isHeating());
26        h.heatingOff();
27        Assertions.assertFalse(h.isHeating());
28        h.heatingOff();
29        Assertions.assertFalse(h.isHeating());
30
31    }
32
33    @Test
34    void beeperTest() {
35        Beeper p = new Beeper();
36        p.beep(100);
37        Assertions.assertTrue(BeeperListener.hasTheBeeberSound(100));
38
39        // Comprobamos que el BeeperListener se ha reestablecido otra vez a 0
40        Assertions.assertTrue(BeeperListener.hasTheBeeberSound(0));
41    }
42

```



```

43 @Test
44 void lampTest() {
45     Lamp p = new Lamp();
46
47     // Comprobamos que la lampara esta apagada
48     Assertions.assertFalse(p.isLampOn());
49
50     // Cambiamos el estado de la lampara
51     p.lampOff();
52     Assertions.assertFalse(p.isLampOn());
53     p.lampOff();
54     p.lampOff();
55     Assertions.assertFalse(p.isLampOn());
56     p.lampOn();
57     Assertions.assertTrue(p.isLampOn());
58     p.lampOn();
59     p.lampOn();
60     Assertions.assertTrue(p.isLampOn());
61 }
62
63 @Test
64 void turnableTest() {
65     Turnable t = new Turnable();
66
67     // Comprobamos que el plato no esta girando
68     Assertions.assertFalse(t.isMoving());
69
70     // Cambiamos el estado del plato
71     t.turnable_stop();
72     Assertions.assertFalse(t.isMoving());
73     t.turnable_start();
74     Assertions.assertTrue(t.isMoving());
75     t.turnable_start();
76     t.turnable_start();
77     t.turnable_start();
78     Assertions.assertTrue(t.isMoving());
79 }
80
81 @Test
82 void testDisplay() {
83     Display d = new Display();
84
85     // Comprobamos que la pantalla este apagada (null)
86     Assertions.assertNull(d.getDisplay());
87
88     // Cambiamos el mensaje de la pantalla
89     d.setDisplay("Test");
90     Assertions.assertEquals("Test", d.getDisplay());
91
92     // Comprobamos que al limpiar la pantalla esta se apaga
93     d.clearDisplay();
94     Assertions.assertNull(d.getDisplay());
95 }
96

```



```

97 @Test
98 void testTimerYPower() {
99     // Probamos a modificar el timer y a resetearlo. Tambien probamos que se muestra
100     // la información correcta en el display
101     m.timer_reset();
102     m.power_reset();
103     m.timer_desc();
104     m.power_desc();
105     Assertions.assertEquals(0, m.getTimer());
106     Assertions.assertEquals(0, m.getPower());
107     aumentaTiempo(10);
108     Assertions.assertEquals(10, m.getTimer());
109     Assertions.assertEquals("10", m.getDisplayConnection().getDisplay());
110     disminuyeTiempo(5);
111     Assertions.assertEquals(5, m.getTimer());
112     Assertions.assertEquals("5", m.getDisplayConnection().getDisplay());
113     m.timer_reset();
114     Assertions.assertEquals(0, m.getTimer());
115     Assertions.assertEquals("0", m.getDisplayConnection().getDisplay());
116
117     // Probamos a modificar y a resetear la potencia. Tambien probamos que se
118     // muestra la información correcta en el display
119     aumentaPotencia(100);
120     Assertions.assertEquals(100, m.getPower());
121     Assertions.assertEquals("100", m.getDisplayConnection().getDisplay());
122     disminuyePotencia(25);
123     Assertions.assertEquals(75, m.getPower());
124     Assertions.assertEquals("75", m.getDisplayConnection().getDisplay());
125     m.power_reset();
126     Assertions.assertEquals(0, m.getPower());
127     Assertions.assertEquals("0", m.getDisplayConnection().getDisplay());
128 }
129
130 @Test
131 void testClosedWithNoItem() {
132     // Comprobamos que se lanzan las excepciones correctamente según su estado
133     // (cerrado sin comida)
134     Assertions.assertThrows(IllegalStateException.class, () -> m.door_closed());
135     Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
136     Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_stop());
137     Assertions.assertThrows(IllegalStateException.class, () -> m.item_placed());
138     Assertions.assertThrows(IllegalStateException.class, () -> m.item_removed());
139     Assertions.assertThrows(IllegalStateException.class, () -> m.tick());
140
141     // Comprobamos el estado interno del microondas
142
143     Assertions.assertFalse(m.isCooking());
144     Assertions.assertFalse(m.isWithItem());
145     Assertions.assertFalse(m.isDoorOpen());
146     Assertions.assertFalse(m.getHeatingConnection().isHeating());
147     Assertions.assertFalse(m.getLampConnection().isLampOn());
148     Assertions.assertFalse(m.getTurnableConnection().isMoving());
149     Assertions.assertTrue(m.getState() instanceof ClosedWithNoItem);
150 }
151
152

```

```

153 @Test
154 void testOpenWithNoItem() {
155     if (m.getState() instanceof ClosedWithNoItem) {
156         m.door_opened();
157     }
158
159     // Comprobamos que se lanzan las excepciones correctamente según su estado
160     // (abierto sin comida)
161
162     Assertions.assertThrows(IllegalStateException.class, () -> m.door_opened());
163     Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
164     Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_stop());
165     Assertions.assertThrows(IllegalStateException.class, () -> m.item_removed());
166     Assertions.assertThrows(IllegalStateException.class, () -> m.tick());
167
168     // Comprobamos el estado interno del microondas
169
170     Assertions.assertFalse(m.isCooking());
171     Assertions.assertFalse(m.isWithItem());
172     Assertions.assertTrue(m.isDoorOpen());
173     Assertions.assertFalse(m.getHeatingConnection().isHeating());
174     Assertions.assertTrue(m.getLampConnection().isLampOn());
175     Assertions.assertFalse(m.getTurnableConnection().isMoving());
176     Assertions.assertTrue(m.getState() instanceof OpenWithNoItem);
177
178     testTimerYPower(); // Comprobamos que se puede modificar el tiempo y la potencia del microondas
179
180     // Comprobamos que se cumplen los test al cerrar la puerta del microondas
181     m.door_closed();
182     testClosedWithNoItem();
183 }
184
185 @Test
186
187 void testOpenWithItem() {
188     if (m.getState() instanceof ClosedWithNoItem) {
189         m.door_opened();
190         m.item_placed();
191     }
192
193     // Comprobamos que se lanzan las excepciones correctamente según su estado
194     // (abierto con comida)
195
196     Assertions.assertThrows(IllegalStateException.class, () -> m.door_opened());
197     Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
198     Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_stop());
199     Assertions.assertThrows(IllegalStateException.class, () -> m.item_placed());
200     Assertions.assertThrows(IllegalStateException.class, () -> m.tick());
201
202     // Comprobamos el estado interno del microondas
203
204     Assertions.assertFalse(m.isCooking());
205     Assertions.assertTrue(m.isWithItem());
206     Assertions.assertTrue(m.isDoorOpen());
207     Assertions.assertFalse(m.getHeatingConnection().isHeating());
208     Assertions.assertTrue(m.getLampConnection().isLampOn());
209     Assertions.assertFalse(m.getTurnableConnection().isMoving());
210     Assertions.assertTrue(m.getState() instanceof OpenWithItem);
211

```

```

212 // Comprobamos que podemos modificar correctamente el tiempo
213
214 testTimerYPower();
215
216 // Comprobamos que el microondas cambia de estado correctamente cuando sacamos
217 // la comida
218 m.item_removed();
219 Assertions.assertTrue(m.getState() instanceof OpenWithNoItem);
220 testOpenWithNoItem();
221 }
222
223 @Test
224 void testClosedWithItem() {
225     if (m.getState() instanceof ClosedWithNoItem) {
226         m.door_opened();
227         m.item_placed();
228         m.door_closed();
229     }
230
231     // Comprobamos que se lanzan las excepciones correctamente según su estado
232     // (cerrado con comida)
233
234     Assertions.assertThrows(IllegalStateException.class, () -> m.door_closed());
235     Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_stop());
236     Assertions.assertThrows(IllegalStateException.class, () -> m.item_placed());
237     Assertions.assertThrows(IllegalStateException.class, () -> m.item_removed());
238     Assertions.assertThrows(IllegalStateException.class, () -> m.tick());
239
240     // Comprobamos el estado interno del microondas
241
242     Assertions.assertFalse(m.isCooking());
243     Assertions.assertTrue(m.isWithItem());
244     Assertions.assertFalse(m.isDoorOpen());
245     Assertions.assertFalse(m.getHeatingConnection().isHeating());
246     Assertions.assertFalse(m.getLampConnection().isLampOn());
247     Assertions.assertFalse(m.getTurnableConnection().isMoving());
248     Assertions.assertTrue(m.getState() instanceof ClosedWithItem);
249
250     // Comprobamos que podemos modificar correctamente el tiempo
251
252     testTimerYPower();
253
254     // Comprobamos el cambio de estado si abrimos la puerta
255
256     m.door_opened();
257     Assertions.assertTrue(m.getState() instanceof OpenWithItem);
258     testOpenWithItem();
259 }
260
261 @Test
262 void testCooking() {
263     if (m.getState() instanceof ClosedWithNoItem) {
264         m.door_opened();
265         m.item_placed();
266         m.door_closed();
267     }
268     // Comprobamos de que no podemos iniciar la coccion sin haber configurado
269     // correctamente el tiempo y la potencia
270     m.timer_reset();

```

```

271 Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
272 aumentaTiempo(15);
273 Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
274 m.timer_reset();
275 aumentaPotencia(50);
276 Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
277
278 // Iniciamos la coccion
279 aumentaTiempo(10);
280 m.cooking_start();
281
282 // Comprobamos que se lanzan las excepciones correctamente según su estado
283 // (cocinando)
284
285 Assertions.assertThrows(IllegalStateException.class, () -> m.door_closed());
286 Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
287 Assertions.assertThrows(IllegalStateException.class, () -> m.item_placed());
288 Assertions.assertThrows(IllegalStateException.class, () -> m.item_removed());
289
290 // Comprobamos el estado interno del microondas
291
292 Assertions.assertTrue(m.isCooking());
293 Assertions.assertTrue(m.isWithItem());
294 Assertions.assertFalse(m.isDoorOpen());
295 Assertions.assertTrue(m.getHeatingConnection().isHeating());
296 Assertions.assertTrue(m.getLampConnection().isLampOn());
297 Assertions.assertTrue(m.getTurnableConnection().isMoving());
298 Assertions.assertTrue(m.getState() instanceof Cooking);
299
300 // Comprobamos que el microondas para cuando se abre la puerta
301 m.door_opened();
302 Assertions.assertTrue(m.getState() instanceof OpenWithItem);
303 Assertions.assertFalse(BeeperListener.hasTheBeeberSound(3));
304 testOpenWithItem();
305
306 // Ponemos el microondas a cocinar otra vez
307 m.door_opened();
308 m.item_placed();
309 m.door_closed();
310 aumentaTiempo(10);
311 aumentaPotencia(100);
312 m.cooking_start();
313
314 // Comprobamos que al reiniciar el temporizador o la potencia el microondas deja
315 // de cocinar
316 m.timer_reset();
317 Assertions.assertTrue(m.getState() instanceof ClosedWithItem);
318 aumentaTiempo(10);
319 m.cooking_start();
320 m.power_reset();
321 Assertions.assertTrue(m.getState() instanceof ClosedWithItem);
322 aumentaPotencia(20);
323 Assertions.assertFalse(BeeperListener.hasTheBeeberSound(3));
324
325 // Probamos que el microondas se detiene correctamente tras quedarse sin tiempo
326 m.cooking_start();
327 Assertions.assertEquals(10, m.getTimer());
328 tiempoPasa(10);
329 Assertions.assertEquals(0, m.getTimer());

```

```


330     Assertions.assertTrue(m.getState() instanceof ClosedWithItem);
331     Assertions.assertEquals("Food is ready", m.getDisplayConnection().getDisplay());
332     Assertions.assertTrue(BeeperListener.hasTheBeeperSound(3));
333
334     // Comprobamos que cumple correctamente el resto de funcionalidades al estar
335     // terminar de cocinar
336     testClosedWithItem();
337 }
338
339 private void aumentaTiempo(int t) {
340     for (int i = 0; i < t; i++) {
341         m.timer_inc();
342     }
343 }
344
345 private void disminuyeTiempo(int t) {
346     for (int i = 0; i < t; i++) {
347         m.timer_desc();
348     }
349 }
350
351 private void aumentaPotencia(int p) {
352     for (int i = 0; i < p; i++) {
353         m.power_inc();
354     }
355 }
356
357 private void disminuyePotencia(int p) {
358     for (int i = 0; i < p; i++) {
359         m.power_desc();
360     }
361 }
362
363 private void tiempoPasa(int t) {
364     for (int i = 0; i < t; i++) {
365         m.tick();
366     }
367 }
368
369 }
370












```

Resultado del test:

Finished after 0,281 seconds

Runs: 11/11 ✖ Errors: 0 ✖ Failures: 0

▼  MicrowaveEngineTest [Runner: JUnit 5] (0,036 s)

-  heatingTest() (0,001 s)
-  testOpenWithItem() (0,000 s)
-  lampTest() (0,000 s)
-  testTimerYPower() (0,000 s)
-  beeperTest() (0,000 s)
-  testOpenWithNoItem() (0,000 s)
-  testClosedWithItem() (0,002 s)
-  testClosedWithNoItem() (0,001 s)
-  testCooking() (0,025 s)
-  testDisplay() (0,002 s)
-  turnableTest() (0,001 s)

Apartado c

Se han implementado cuatro características (features) diferentes.

ClosedEmptyMicrowave.feature

Feature: Using a closed microwave with no item on it

Scenario: Open a closed microwave

Given a closed empty microwave

When I open the door

Then the door opens

And the light turns on

And the plate is not turning

And the microwave is not heating

Scenario: Trying to reset time in a microwave

Given a closed empty microwave

When I press reset timer button

Then the timer must be set to zero

Scenario: Trying to reset power in a microwave

Given a closed empty microwave

When I press reset power button

Then the power must be set to zero

Scenario Outline: Trying to set time

Given a closed empty microwave

When I set time to **<a>** seconds

Then the microwave must display "****"

Examples:

a	b
-1	0
0	0
1	1
15	15
100	100
40	40

Scenario Outline: Trying to set power

Given a closed empty microwave

When I set the power to **<a>**

Then the microwave must display "****"

Examples:

a	b
-1	0
0	0
20	20
200	200
201	201
2	2

Scenario: Trying to cook in a closed microwave with no item

Given a closed empty microwave

When I set time to 20 seconds

And I set the power to 100

And I press start cooking button

Then the microwave must not start cooking

OpenedMicrowave.feature

Feature: Using an opened microwave

Scenario: Open a closed microwave

Given a closed empty microwave

When I open the door

Then the door opens

And the light turns on

And the plate is not turning

And the microwave is not heating

Scenario: Placing food in a microwave

Given an opened empty microwave

When I insert food

Then the food is placed

Scenario: Removing food from a microwave

Given an opened full microwave

When I retrieve the food

Then the microwave is empty

Scenario: Trying to reset time in a microwave

Given an opened empty microwave

When I press reset timer button

Then the timer must be set to zero

Scenario: Trying to reset power in a microwave

Given an opened empty microwave

When I press reset power button

Then the power must be set to zero

Scenario: Trying to cook in a opened microwave

Given an opened empty microwave

When I set time to 20 seconds

And I set the power to 100

And I press start cooking button

Then the microwave must not start cooking

Scenario Outline: Trying to set time

Given an opened empty microwave

When I set time to <a> seconds

Then the microwave must display ""

Examples:

a	b
-1	0
0	0
1	1
15	15
100	100
40	40

Scenario Outline: Trying to set power

Given an opened empty microwave

When I set the power to <a>

Then the microwave must display ""

Examples:

a	b
-1	0
0	0
20	20
200	200
201	201
2	2

ClosedFullMicrowave.feature

Feature: Using a closed Microwave with an item inside

Scenario: Close the door of a full door-opened microwave

Given an opened empty microwave

And I insert food

When I close the door

Then the door is closed

And the light are not on

And the plate is not turning

And the microwave is not heating

Scenario: Trying to reset time in a microwave

Given a closed empty microwave

When I press reset timer button

Then the timer must be set to zero

Scenario: Trying to reset power in a microwave

Given a closed empty microwave

When I press reset power button

Then the power must be set to zero

Scenario Outline: Trying to set time

Given a closed empty microwave

When I set time to <a> seconds

Then the microwave must display ""

Examples:

a	b
-1	0
0	0
1	1
15	15
100	100
40	40

Scenario Outline: Trying to set power

Given a closed empty microwave

When I set the power to <a>

Then the microwave must display ""

Examples:

a	b
-1	0
0	0
20	20
200	200
201	201
2	2

CookingWithMicrowave.feature

Feature: Cooking with a closed full microwave

Scenario: Start cooking with a microwave

Given a full closed microwave

When I set time to 20 seconds

And I set the power to 100

And I press start cooking button

Then the microwave must start cooking

And the light turns on

And the plate is turning

And the microwave is heating

Scenario Outline: Cooking time finishes correctly

Given a full closed microwave

When I set time to <a> seconds

And I set the power to 100

And I press start cooking button

And it passes seconds

Then the microwave is not cooking

And the light are not on

And the plate is not turning

And the microwave is not heating

And the beeper sounds 3 times

And the microwave must display "Food is ready"

Examples:

a	b
5	5
7	7
10	10
15	15
40	40
120	120

Scenario: Increase time during cooking process

Given a full closed microwave cooking with a timing of 10 seconds and a power of 100

When I press the increase timer button

Then the microwave must display "11"

And the cooking time must be 11

Scenario: Decrease time during cooking process

Given a full closed microwave cooking with a timing of 10 seconds and a power of 100

When I press the decrease timer button

Then the microwave must display "9"

And the cooking time must be 9

Scenario: Increase power during cooking process

Given a full closed microwave cooking with a timing of 10 seconds and a power of 100

When I press the increase power button

Then the microwave must display "101"

Scenario: Decrease power during cooking process

Given a full closed microwave cooking with a timing of 10 seconds and a power of 100

When I press the decrease power button

Then the microwave must display "99"

Scenario: Abort cooking process by opening the door

Given a full closed microwave cooking with a timing of 5 seconds and a power of 50

When I open the door

Then the door opens

And the microwave is not cooking

And the light turns on

And the plate is not turning

And the microwave is not heating

Scenario: Start again cooking after opening the microwave while it was cooking

Given a full closed microwave cooking with a timing of 3 seconds and a power of 50

When I open the door

And I close the door

And I press the increase timer button

Then the microwave must display "4"

Scenario: Abort cooking process by resetting the timer

Given a full closed microwave cooking with a timing of 1 seconds and a power of 2

When I press reset timer button

Then the microwave is not cooking

And the light are not on

And the plate is not turning

And the microwave is not heating

And the microwave must display "0"

Scenario: Abort cooking process by resetting the power

Given a full closed microwave cooking with a timing of 11 seconds and a power of 600

When I press reset power button

Then the microwave is not cooking

And the light are not on

And the plate is not turning

And the microwave is not heating

And the microwave must display "0"

Scenario: Abort cooking process by pressing decrease time button when the time is 1

Given a full closed microwave cooking with a timing of 1 seconds and a power of 10

When I press the decrease timer button

Then the microwave is not cooking

And the light are not on

And the plate is not turning

And the microwave is not heating

And the microwave must display "Food is ready"

And the beeper sounds 3 times

Scenario: Abort cooking process by pressing decrease power button when the power is 1

Given a full closed microwave cooking with a timing of 10 seconds and a power of 1

When I press the decrease power button

Then the microwave is not cooking

And the light are not on

And the plate is not turning

And the microwave is not heating

And the microwave must display "0"

And the beeper sounds 0 times

StepDefinitions

En la clase StepDefinitions se ha definido cada uno de los métodos que se invocarán con los escenarios.

```
1 package microwave;
2
3 import io.cucumber.java.en.Given;
4
5
6
7
8
9
10
11
12 public class StepDefinitions {
13     private Microwave m;
14
15     @Given("a closed empty microwave")
16     public void a_closed_empty_microwave() {
17         m = new Microwave();
18     }
19
20     @Given("an opened empty microwave")
21     public void an_opened_empty_microwave() {
22         a_closed_empty_microwave();
23         m.door_opened();
24     }
25
26     @Given("an opened full microwave")
27     public void an_opened_full_microwave() {
28         an_opened_empty_microwave();
29         m.item_placed();
30     }
31
32
33     @Given("a full closed microwave")
34     public void a_full_closed_microwave() {
35         an_opened_full_microwave();
36         m.door_closed();
37     }
38
39     @Given("a full closed microwave cooking with a timing of {int} seconds and a power of {int}")
40     public void a_full_closed_microwave_cooking_with_a_timing_of_seconds_and_a_power_of(Integer time, Integer power) {
41         a_full_closed_microwave();
42         i_set_time_to_seconds(time);
43         i_set_the_power_to(power);
44         m.cooking_start();
45     }
46
47     @When("I open the door")
48     public void i_open_the_door() {
49         m.door_opened();
50     }
51
52     @When("I set time to {int} seconds")
53     public void i_set_time_to_seconds(Integer times) {
54         m.timer_reset();
55         for (int i = 0; i < times; i++) {
56             m.timer_inc();
57         }
58     }
59
60
61     @When("I set the power to {int}")
62     public void i_set_the_power_to(Integer power) {
63         m.power_reset();
64         for (int i = 0; i < power; i++) {
65             m.power_inc();
66         }
67     }
68 }
```

```

68
69 @When("I press start cooking button")
70 public void i_press_start_cooking_button() {
71     try {
72         m.cooking_start();
73     } catch (IllegalStateException e) {
74         the_microwave_must_not_start_cooking();
75     }
76 }
77
78 @When("I insert food")
79 public void i_insert_food() {
80     m.item_placed();
81 }
82
83 @When("I retrieve the food")
84 public void i_retrieve_the_food() {
85     m.item_removed();
86 }
87
88 @When("I press reset timer button")
89 public void i_press_reset_timer_button() {
90     m.timer_reset();
91 }
92
93 @When("I press reset power button")
94 public void i_press_reset_power_button() {
95     m.power_reset();
96 }
97
98 @When("I close the door")
99 public void i_close_the_door() {
100     m.door_closed();
101 }
102
103 @When("it passes {int} seconds")
104 public void it_passes_seconds(Integer t) {
105     pass_time(t);
106 }
107
108 @When("I press the increase power button")
109 public void i_press_the_increase_power_button() {
110     m.power_inc();
111     ;
112 }
113
114 @When("I press the decrease timer button")
115 public void i_press_the_decrease_timer_button() {
116     m.timer_desc();
117 }
118
119 @When("I press the decrease power button")
120 public void i_press_the_decrease_power_button() {
121     m.power_desc();
122 }
123
124 @When("I press the increase timer button")
125 public void i_press_the_increase_timer_button() {
126     m.timer_inc();
127 }

```

```
128
129⊖ @Then("the door opens")
130 public void the_door_opens_and_the_light_turns_on() {
131     Assertions.assertTrue(m.isDoorOpen());
132 }
133
134⊖ @Then("the light turns on")
135 public void the_light_turns_on() {
136     Assertions.assertTrue(m.getLampConnection().isLampOn());
137 }
138
139⊖ @Then("the microwave must not start cooking")
140 public void the_microwave_must_not_start_cooking() {
141     Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
142 }
143
144⊖ @Then("the microwave must display {string}")
145 public void the_microwave_must_display(String i) {
146     Assertions.assertEquals(i, m.getDisplayConnection().getDisplay());
147 }
148
149⊖ @Then("the plate is not turning")
150 public void the_plate_is_not_turning() {
151     Assertions.assertFalse(m.getTurnableConnection().isMoving());
152 }
153
154⊖ @Then("the microwave is not heating")
155 public void the_microwave_is_not_heating() {
156     Assertions.assertFalse(m.getHeatingConnection().isHeating());
157 }
158
159⊖ @Then("the timer must be set to zero")
160 public void the_timer_must_be_set_to_zero() {
161     Assertions.assertEquals("0", m.getDisplayConnection().getDisplay());
162 }
163
164⊖ @Then("the power must be set to zero")
165 public void the_power_must_be_set_to_zero() {
166     Assertions.assertEquals("0", m.getDisplayConnection().getDisplay());
167 }
168
169⊖ @Then("the microwave is empty")
170 public void the_microwave_is_empty() {
171     Assertions.assertFalse(m.isWithItem());
172 }
173
174⊖ @Then("the food is placed")
175 public void the_food_is_placed() {
176     Assertions.assertTrue(m.isWithItem());
177 }
178
179⊖ @Then("the door is closed")
180 public void the_door_is_closed() {
181     Assertions.assertFalse(m.isDoorOpen());
182 }
183
184⊖ @Then("the light are not on")
185 public void the_light_are_not_on() {
186     Assertions.assertFalse(m.getLampConnection().isLampOn());
187 }
```

```

188
189 @Then("the microwave must start cooking")
190 public void the_microwave_must_start_cooking() {
191     Assertions.assertTrue(m.isCooking());
192 }
193
194 @Then("the plate is turning")
195 public void the_plate_is_turning() {
196     Assertions.assertTrue(m.getTurnableConnection().isMoving());
197     ;
198 }
199
200 @Then("the microwave is heating")
201 public void the_microwave_is_heating() {
202     Assertions.assertTrue(m.getHeatingConnection().isHeating());
203 }
204
205 @Then("the cooking time must be {int}")
206 public void the_cooking_time_must_be(Integer time) {
207     pass_time(time);
208     Assertions.assertEquals("Food is ready", m.getDisplayConnection().getDisplay());
209 }
210
211 @Then("the microwave is not cooking")
212 public void the_microwave_is_not_cooking() {
213     Assertions.assertFalse(m.isCooking());
214 }
215
216 @Then("the beeper sounds {int} times")
217 public void the_beeper_sound_times(Integer t) {
218     Assertions.assertTrue(BeeperListener.hasTheBeeperSound(t));
219 }
220
221 private void pass_time(int t) {
222     for (int i = 0; i < t; i++) {
223         m.tick();
224     }
225 }
226 }
227

```

Resultados de ejecutar dichos tests

```

[INFO] Tests run: 66, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.727 s - in microwave.RunCucumberTest
[INFO] Running microwaveEngine.MicrowaveEngineTest
[INFO] Tests run: 11, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.07 s - in microwaveEngine.MicrowaveEngineTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 77, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.146 s
[INFO] Finished at: 2022-05-20T16:47:03+02:00
[INFO]

```

Enlace a GitHub

<https://github.com/FlorinUMA/Microwave>