

Proyecto final IS-Avanzada: Horno-Microondas

Hugo Ávalos de Rorthais

Enlace: https://github.com/hugoavalos01/ISA_Microwave

1. Implementación en Java, usando el patrón de diseño Estado.

Interfaz Estado: Distintos métodos para manejar los diferentes componentes del microondas como la puerta, la luz, el temporizador, la potencia y meter o sacar la comida. Estos métodos harán que se cambie de un estado a otro.

```
1 package microwave;
2
3 public interface Estado {
4
5     public void door_opened(Microwave m);
6
7     public void door_closed(Microwave m);
8
9     public void item_placed(Microwave m);
10
11     public void item_removed(Microwave m);
12
13     public void cooking_start(Microwave m);
14
15     public void cooking_stop(Microwave m);
16
17     public void timer_reset(Microwave m);
18
19     public void power_reset(Microwave m);
20
21     public void tick(Microwave m);
22
23     public void power_desc(Microwave m);
24
25     public void timer_desc(Microwave m);
26 }
27
```

Clase Microwave: El constructor inicial crea un microondas cerrado y vacío, con tiempo y potencia 0. Tenemos los mismos métodos que la interfaz y además los getters y los setters.

```
1 package microwave;
2 public class Microwave {
3
4     private boolean doorOpen;
5     private Integer power;
6     private Integer timer;
7     private boolean cooking;
8     private boolean withItem;
9     private Estado estado;
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    //Constructor
17
18    public Microwave() {
19
20        doorOpen = false;
21        cooking = false;
22        withItem = false;
23        power = 0;
24        timer = 0;
25        estado = new ClosedWithNoItem(this);
26    }
27
28    //Metodos
29
30    public void door_opened() {
31        estado.door_opened(this);
32    }
33
34    public void door_closed() {
35        estado.door_closed(this);
36    }
37
38    public void item_placed() {
39        estado.item_placed(this);
40    }
41
42    public void item_removed() {
43        estado.item_removed(this);
44    }
45
46    public void power_inc() {
47        power++;
48        displayConnection.setDisplay(Integer.toString(power));
49    }
50
51    public void power_desc() {
52        estado.power_desc(this);
53    }
54
55    public void power_reset() {
56        estado.power_reset(this);
57        displayConnection.setDisplay(Integer.toString(power));
58    }
59
60    public void timer_inc() {
61        timer++;
62        displayConnection.setDisplay(Integer.toString(timer));
63    }
64
65    public void timer_desc() {
66        estado.timer_desc(this);
67    }
68
69    public void timer_reset() {
70        estado.timer_reset(this);
71        displayConnection.setDisplay(Integer.toString(timer));
72    }
73
74 }
```

```

    public void cooking_start() {
        estado.cooking_start(this);
    }

    public void cooking_stop() {
        estado.cooking_stop(this);
    }

    public void tick() {
        estado.tick(this);
    }

    // Getters y Setters

    public boolean isDoorOpen() {
        return doorOpen;
    }

    public void setDoorOpen(boolean doorOpen) {
        this.doorOpen = doorOpen;
    }

    public int getPower() {
        return power;
    }

    public void setPower(int p) {
        power = p;
    }

    public int getTimer() {
        return timer;
    }

    public void setTimer(int t) {
        timer = t;
    }

```

```

    public boolean isCooking() {
        return cooking;
    }

    public void setCooking(boolean c) {
        cooking = c;
    }

    public boolean isWithItem() {
        return withItem;
    }

    public void setWithItem(boolean withItem) {
        this.withItem = withItem;
    }

    public Estado getEstado() {
        return estado;
    }

    public void setEstado(Estado e) {
        estado = e;
    }

    public Heating getHeatingConnection() {
        return heatingConnection;
    }

    public Lamp getLampConnection() {
        return lampConnection;
    }

    public Turnable getTurnableConnection() {
        return turnableConnection;
    }

```

```

        public Beeper getBeeperConnection() {
            return beeperConnection;
        }

        public Display getDisplayConnection() {
            return displayConnection;
        }
    }

```

Clase Beeper y beepListener: Para simular los sonidos del beeper se ha implementado el patrón Observador y se ha creado **beepListener**. Este recibe el número de beeps según el método de la clase Beeper y lo almacena en la variable “beeps”. Mediante el método “times” nos aseguramos que han sonado los beeps esperados y reiniciamos la variable a 0.

```
package microwave;

public class Beeper {

    public void beep(int d) {
        beepListener.listen(d);
    }
}
```

```
package microwave;

public class beepListener {

    private static int beeps = 0;

    public static void listen (int b) {
        beeps = b;
    }

    public static boolean times(int t) {
        int b = beeps;
        beeps = 0;
        return b == t;
    }
}
```

Clase Lamp:

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

Clase Heating:

```
1 package microwave;
2
3 public class Heating {
4
5     private boolean heating = false;
6     private int power = 0;
7
8     void heatingOn() {
9         heating = true;
10    }
11
12    void heatingOff() {
13        heating = false;
14    }
15
16    void setPower(int p) {
17        if (p >= 0) {
18            power = p;
19        }
20    }
21
22    int getPower() {
23        return power;
24    }
25
26    public boolean isHeating() {
27        return heating;
28    }
29 }
30
```

Clase Display:

```
1 package microwave;
2
3 public class Display {
4
5     private String display;
6
7     void clearDisplay() {
8         display = null;
9     }
10
11    void setDisplay(String s) {
12        display = s;
13    }
14
15    public String getDisplay() {
16        return display;
17    }
18 }
19
```

Clase Turnable:

```
1 package microwave;
2
3 public class Turnable {
4
5     private boolean turnableOn = false;
6
7     void turnable_start() {
8         turnableOn = true;
9     }
10
11     void turnable_stop() {
12         turnableOn = false;
13     }
14
15     public boolean isMoving() {
16         return turnableOn;
17     }
18 }
19
```

Microondas OpenWithNoItem:

```
1 package microwave;
2
3 public class OpenWithNoItem implements Estado {
4
5     public OpenWithNoItem(Microwave m) {
6         // TODO Auto-generated constructor stub
7
8         m.setDoorOpen(true);
9         m.setWithItem(false);
10        m.setCooking(false);
11
12        m.getHeatingConnection().heatingOff();
13        m.getTurnableConnection().turnable_stop();
14        m.getLampConnection().lampOn();
15    }
16
17    @Override
18    public void door_opened(Microwave m) {
19        // TODO Auto-generated method stub
20        throw new IllegalStateException("Error: La puerta ya esta abierta");
21    }
22
23    @Override
24    public void door_closed(Microwave m) {
25        // TODO Auto-generated method stub
26        m.setEstado(new ClosedWithNoItem(m));
27    }
28
29
30    @Override
31    public void item_placed(Microwave m) {
32        // TODO Auto-generated method stub
33        m.setEstado(new OpenWithItem(m));
34    }
35
36    @Override
37    public void item_removed(Microwave m) {
38        // TODO Auto-generated method stub
39        throw new IllegalStateException("Error: No hay ning n objeto para sacar");
40    }
41
42 }
```

```

43@ @Override
44 public void cooking_start(Microwave m) {
45     // TODO Auto-generated method stub
46     throw new IllegalStateException("Error: La puerta tiene que estar cerrada");
47 }
48
49
50@ @Override
51 public void cooking_stop(Microwave m) {
52     // TODO Auto-generated method stub
53     throw new IllegalStateException("Error: El microondas no esta cocinando");
54 }
55
56
57@ @Override
58 public void timer_reset(Microwave m) {
59     // TODO Auto-generated method stub
60     m.setTimer(0);
61 }
62
63@ @Override
64 public void power_reset(Microwave m) {
65     // TODO Auto-generated method stub
66     m.setPower(0);
67 }
68
69@ @Override
70 public void tick(Microwave m) {
71     // TODO Auto-generated method stub
72     throw new IllegalStateException("Error: El microondas no esta cocinando");
73 }
74
75
76@ @Override
77 public void power_desc(Microwave m) {
78     // TODO Auto-generated method stub
79     if (m.getPower() > 0) {
80         m.setPower(m.getPower() - 1);
81         m.getDisplayConnection().setDisplay(Integer.toString(m.getPower()));
82     }
83 }

```



```

@Override
public void timer_reset(Microwave m) {
    // TODO Auto-generated method stub
    m.setTimer(0);
}

@Override
public void power_reset(Microwave m) {
    // TODO Auto-generated method stub
    m.setPower(0);
}

@Override
public void tick(Microwave m) {
    // TODO Auto-generated method stub
    throw new IllegalStateException("Error: El microondas no esta cocinando");
}

@Override
public void power_desc(Microwave m) {
    // TODO Auto-generated method stub
    if (m.getPower() > 0) {
        m.setPower(m.getPower() - 1);
        m.getDisplayConnection().setDisplay(Integer.toString(m.getPower()));
    }
}

@Override
public void timer_desc(Microwave m) {
    // TODO Auto-generated method stub
    if (m.getTimer() > 0) {
        m.setTimer(m.getTimer() - 1);
        m.getDisplayConnection().setDisplay(Integer.toString(m.getTimer()));
    }
}
}

```

Microondas OpenWithItem:

```
1 package microwave;
2
3 public class OpenWithItem implements Estado {
4
5     public OpenWithItem (Microwave m) {
6
7         m.setDoorOpen(true);
8         m.setWithItem(true);
9         m.setCooking(false);
10
11         m.getHeatingConnection().heatingOff();
12         m.getTurnableConnection().turnable_stop();
13         m.getLampConnection().lampOn();
14
15     }
16
17     @Override
18     public void door_opened(Microwave m) {
19         // TODO Auto-generated method stub
20         throw new IllegalStateException("Error: La puerta ya esta abierta");
21     }
22
23     @Override
24     public void door_closed(Microwave m) {
25         // TODO Auto-generated method stub
26         m.setEstado(new ClosedWithItem(m));
27     }
28
29     @Override
30     public void item_placed(Microwave m) {
31         // TODO Auto-generated method stub
32         throw new IllegalStateException("Error: La puerta ya esta abierta");
33     }
34
35
36
37     public void item_removed(Microwave m) {
38         // TODO Auto-generated method stub
39         m.setEstado(new OpenWithNoItem(m));
40     }
41
42     @Override
43     public void cooking_start(Microwave m) {
44         // TODO Auto-generated method stub
45         throw new IllegalStateException("Error: La puerta tiene que estar cerrada");
46
47     }
48
49     @Override
50     public void cooking_stop(Microwave m) {
51         // TODO Auto-generated method stub
52         throw new IllegalStateException("Error: El microondas no esta cocinando");
53
54     }
55
56     @Override
57     public void timer_reset(Microwave m) {
58         // TODO Auto-generated method stub
59         m.setTimer(0);
60     }
61
62     @Override
63     public void power_reset(Microwave m) {
64         // TODO Auto-generated method stub
65         m.setPower(0);
66     }
67
68     @Override
69     public void tick(Microwave m) {
70         // TODO Auto-generated method stub
71         throw new IllegalStateException("Error: El microondas no esta cocinando");
72
73     }
```



```

@Override
public void power_desc(Microwave m) {
    // TODO Auto-generated method stub
    if (m.getPower() > 0) {
        m.setPower(m.getPower() - 1);
        m.getDisplayConnection().setDisplay(Integer.toString(m.getPower()));
    }
}

@Override
public void timer_desc(Microwave m) {
    // TODO Auto-generated method stub
    if (m.getTimer() > 0) {
        m.setTimer(m.getTimer() - 1);
        m.getDisplayConnection().setDisplay(Integer.toString(m.getTimer()));
    }
}
}

```

Microondas ClosedWithNoItem:

```

1 package microwave;
2
3 public class ClosedWithNoItem implements Estado {
4
5     public ClosedWithNoItem(Microwave m) {
6         // TODO Auto-generated constructor stub
7
8         m.setDoorOpen(false);
9         m.setWithItem(false);
10        m.setCooking(false);
11
12        m.getHeatingConnection().heatingOff();
13        m.getTurnableConnection().turnable_stop();
14        m.getLampConnection().lampOff();
15    }
16
17    @Override
18    public void door_opened(Microwave m) {
19        // TODO Auto-generated method stub
20        m.setEstado(new OpenWithNoItem(m));
21    }
22
23    @Override
24    public void door_closed(Microwave m) {
25        // TODO Auto-generated method stub
26        throw new IllegalStateException("Error: La puerta ya esta cerrada");
27    }
28
29
30    @Override
31    public void item_placed(Microwave m) {
32        // TODO Auto-generated method stub
33        throw new IllegalStateException("Error: Hay que abrir la puerta primero");
34    }
35
36 }

```

```

37 @Override
38 public void item_removed(Microwave m) {
39     // TODO Auto-generated method stub
40     throw new IllegalStateException("Error: Hay que abrir la puerta primero");
41 }
42
43
44 @Override
45 public void cooking_start(Microwave m) {
46     // TODO Auto-generated method stub
47     throw new IllegalStateException("Error: No hay nada dentro");
48 }
49
50
51 @Override
52 public void cooking_stop(Microwave m) {
53     // TODO Auto-generated method stub
54     throw new IllegalStateException("Error: El microondas no esta cocinando");
55 }
56
57
58 @Override
59 public void timer_reset(Microwave m) {
60     // TODO Auto-generated method stub
61     m.setTimer(0);
62 }
63
64 @Override
65 public void power_reset(Microwave m) {
66     // TODO Auto-generated method stub
67     m.setPower(0);
68 }
69
70 @Override
71 public void tick(Microwave m) {
72     // TODO Auto-generated method stub
73     throw new IllegalStateException("Error: El microondas no esta cocinando");

```

```

@Override
public void power_desc(Microwave m) {
    // TODO Auto-generated method stub
    if (m.getPower() > 0) {
        m.setPower(m.getPower() - 1);
        m.getDisplayConnection().setDisplay(Integer.toString(m.getPower()));
    }
}

@Override
public void timer_desc(Microwave m) {
    // TODO Auto-generated method stub
    if (m.getTimer() > 0) {
        m.setTimer(m.getTimer() - 1);
        m.getDisplayConnection().setDisplay(Integer.toString(m.getTimer()));
    }
}

}

```

Microondas ClosedWithItem:

```
1 package microwave;
2
3 public class ClosedWithItem implements Estado {
4
5     public ClosedWithItem(Microwave m) {
6         // TODO Auto-generated constructor stub
7
8         m.setDoorOpen(false);
9         m.setWithItem(true);
10        m.setCooking(false);
11
12        m.getHeatingConnection().heatingOff();
13        m.getTurnableConnection().turnable_stop();
14        m.getLampConnection().lampOff();
15    }
16
17    @Override
18    public void door_opened(Microwave m) {
19        // TODO Auto-generated method stub
20        m.setEstado(new OpenWithItem(m));
21    }
22
23    @Override
24    public void door_closed(Microwave m) {
25        // TODO Auto-generated method stub
26        throw new IllegalStateException("Error: La puerta ya esta cerrada");
27    }
28
29
30    @Override
31    public void item_placed(Microwave m) {
32        // TODO Auto-generated method stub
33        throw new IllegalStateException("Error: Hay que abrir la puerta primero");
34    }
35 }
```

```
37 @Override
38 public void item_removed(Microwave m) {
39     // TODO Auto-generated method stub
40     throw new IllegalStateException("Error: Hay que abrir la puerta primero");
41 }
42
43
44 @Override
45 public void cooking_start(Microwave m) {
46     // TODO Auto-generated method stub
47
48     if (m.getTimer() == 0 || m.getPower() == 0) {
49
50         throw new IllegalStateException("Error: Power y Timer deben ser mayor a 0");
51     }
52
53     else {
54
55         m.setEstado(new Cooking(m));
56     }
57 }
58
59
60 @Override
61 public void cooking_stop(Microwave m) {
62     // TODO Auto-generated method stub
63     throw new IllegalStateException("Error: El microondas no esta cocinando");
64 }
65
66
67 @Override
68 public void timer_reset(Microwave m) {
69     // TODO Auto-generated method stub
70     m.setTimer(0);
71 }
72
```

```

@Override
public void power_reset(Microwave m) {
    // TODO Auto-generated method stub
    m.setPower(0);
}

@Override
public void tick(Microwave m) {
    // TODO Auto-generated method stub
    throw new IllegalStateException("Error: El microondas no esta cocinando");
}

@Override
public void power_desc(Microwave m) {
    // TODO Auto-generated method stub
    if (m.getPower() > 0) {
        m.setPower(m.getPower() - 1);
        m.getDisplayConnection().setDisplay(Integer.toString(m.getPower()));
    }
}

@Override
public void timer_desc(Microwave m) {
    // TODO Auto-generated method stub
    if (m.getTimer() > 0) {
        m.setTimer(m.getTimer() - 1);
        m.getDisplayConnection().setDisplay(Integer.toString(m.getTimer()));
    }
}
}

```

Microondas Cooking:

```
1 package microwave;
2
3 public class Cooking implements Estado {
4
5     public Cooking(Microwave m) {
6         // TODO Auto-generated constructor stub
7
8         m.setDoorOpen(false);
9         m.setWithItem(true);
10        m.setCooking(true);
11
12        m.getHeatingConnection().heatingOn();
13        m.getTurnableConnection().turnable_start();
14        m.getLampConnection().lampOn();
15        m.getHeatingConnection().setPower(m.getPower());
16    }
17
18    @Override
19    public void door_opened(Microwave m) {
20        // TODO Auto-generated method stub
21        m.setEstado(new OpenWithItem(m));
22    }
23
24    @Override
25    public void door_closed(Microwave m) {
26        // TODO Auto-generated method stub
27        throw new IllegalStateException("Error: La puerta ya esta cerrada");
28    }
29
30
31    @Override
32    public void item_placed(Microwave m) {
33        // TODO Auto-generated method stub
34        throw new IllegalStateException("Error: Hay que abrir la puerta primero");
35    }
36
37
38    @Override
39    public void item_removed(Microwave m) {
40        // TODO Auto-generated method stub
41        throw new IllegalStateException("Error: Hay que abrir la puerta primero");
42    }
43
44
45    @Override
46    public void cooking_start(Microwave m) {
47        // TODO Auto-generated method stub
48        throw new IllegalStateException("Error: Ya est  cocinando");
49    }
50
51
52    @Override
53    public void cooking_stop(Microwave m) {
54        // TODO Auto-generated method stub
55        m.setEstado(new ClosedWithItem(m));
56    }
57
58
59    @Override
60    public void timer_reset(Microwave m) {
61        // TODO Auto-generated method stub
62        m.setEstado(new ClosedWithItem(m));
63        m.setTimer(0);
64    }
65
66
67    @Override
68    public void power_reset(Microwave m) {
69        // TODO Auto-generated method stub
70        m.setEstado(new ClosedWithItem(m));
71        m.setPower(0);
72    }
73
74 }
```

```

73     public void tick(Microwave m) {
74         // TODO Auto-generated method stub
75
76         if (m.getTimer() > 1) {
77             m.timer_desc();
78         } else {
79             m.timer_desc();
80             m.getBeeperConnection().beep(3);
81             m.getDisplayConnection().setDisplay("Listo");
82             cooking_stop(m);
83         }
84     }
85
86     @Override
87     public void power_desc(Microwave m) {
88         // TODO Auto-generated method stub
89
90         if (m.getPower() == 0) {
91             cooking_stop(m);
92         } else {
93
94             m.setPower(m.getPower() - 1);
95             m.getDisplayConnection().setDisplay(Integer.toString(m.getPower()));
96         }
97     }
98
99 }
100

```

```

    @Override
    public void timer_desc(Microwave m) {
        // TODO Auto-generated method stub

        if (m.getTimer() == 1) {
            m.getBeeperConnection().beep(3);
            m.getDisplayConnection().setDisplay("Listo");
            cooking_stop(m);

        } else {

            m.setTimer(m.getTimer() - 1);
            m.getDisplayConnection().setDisplay(Integer.toString(m.getTimer()));

        }

    }

}

```

2. Definir pruebas unitarias con Junit para cada uno de los componentes que conforman el sistema:

```
1 package microwave;
2 import org.junit.jupiter.api.*;
3
4 public class Test_Microwave {
5
6     private Microwave m;
7
8     @BeforeEach
9     void init() {
10         m = new Microwave();
11     }
12
13
14     /*
15      * Comprobamos los componentes del microondas
16      */
17
18     @Test
19     void testBeeper() {
20
21         Beeper b = new Beeper();
22
23         // Comprobamos que escucha los beeps
24         b.beep(3);
25         Assertions.assertTrue(beeperListener.times(3));
26
27         // Comprobamos que se reinicia a 0
28         Assertions.assertTrue(beeperListener.times(0));
29     }
30
31     @Test
32     void testDisplay() {
33
34         Display d = new Display();
35
36         // Comprobamos que se inicia sin ningun mensaje
37         Assertions.assertEquals(null, d.getDisplay());
38
39         // Comprobamos que cambia el mensaje
40         d.setDisplay("Mensaje");
41         Assertions.assertEquals("Mensaje", d.getDisplay());
42
43         // Comprobamos que el mensaje se borra
44         d.clearDisplay();
45         Assertions.assertEquals(null, d.getDisplay());
46
47     }
48 }
```



```

@Test
void testTimer() {

    // Comprobamos que se inicia en 0
    Assertions.assertEquals(0, m.getTimer());

    // Comprobamos que incrementa el tiempo
    m.timer_inc();
    Assertions.assertEquals(1, m.getTimer());

    // Comprobamos que decrementa el tiempo
    m.timer_inc();
    m.timer_desc();
    Assertions.assertEquals(1, m.getTimer());

    // Comprobamos que el display muestra los cambios de tiempo
    Assertions.assertEquals("1", m.getDisplayConnection().getDisplay());

    // Comprobamos que se reinicia el tiempo
    m.timer_reset();
    Assertions.assertEquals(0, m.getTimer());

    // Comprobamos que no baja de 0
    m.timer_desc();
    Assertions.assertEquals(0, m.getTimer());

}

```

```

77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122

@Test
void testPower() {

    // Comprobamos que se inicia en 0
    Assertions.assertEquals(0, m.getPower());

    // Comprobamos que incrementa la potencia
    m.power_inc();
    Assertions.assertEquals(1, m.getPower());

    // Comprobamos que decrementa la potencia
    m.power_inc();
    m.power_desc();
    Assertions.assertEquals(1, m.getPower());

    // Comprobamos que el display muestra los cambios de potencia
    Assertions.assertEquals("1", m.getDisplayConnection().getDisplay());

    // Comprobamos que se reinicia la potencia
    m.power_reset();
    Assertions.assertEquals(0, m.getPower());

    // Comprobamos que no baja de 0
    m.power_desc();
    Assertions.assertEquals(0, m.getPower());

}

@Test
void turnableTest() {

    Turnable t = new Turnable();

    // Comprobamos que se inicia parado
    Assertions.assertFalse(t.isMoving());

    // Comprobamos que gira y se para correctamente
    t.turnable_start();
    Assertions.assertTrue(t.isMoving());

    t.turnable_stop();
    Assertions.assertFalse(t.isMoving());

}

```

```

123 @Test
124 void heatingTest() {
125
126     Heating h = new Heating();
127
128     // Comprobamos que se inicia con potencia 0 y sin calentar
129     Assertions.assertFalse(h.isHeating());
130     Assertions.assertEquals(0, h.getPower());
131
132     // Comprobamos que se enciende y se apaga
133     h.heatingOn();
134     Assertions.assertTrue(h.isHeating());
135
136     h.heatingOff();
137     Assertions.assertFalse(h.isHeating());
138
139     // Comprobamos que podemos ajustar la potencia
140     h.setPower(5);
141     Assertions.assertEquals(5, h.getPower());
142
143 }
144
145 @Test
146 void lampTest() {
147
148     Lamp l = new Lamp();
149
150     // Comprobamos que se inicia apagada
151     Assertions.assertFalse(l.isLampOn());
152
153     // Comprobamos que se enciende y apaga
154     l.lampOn();
155     Assertions.assertTrue(l.isLampOn());
156
157     l.lampOff();
158     Assertions.assertFalse(l.isLampOn());
159
160
161 }

```

```

/*
 * Comprobamos los estados del microondas
 */

@Test
void testOpenWithNoItem() {

    m.setEstado(new OpenWithNoItem(m));
    Assertions.assertTrue(m.getEstado() instanceof OpenWithNoItem);

    // Comprobamos que se inicia correctamente
    Assertions.assertFalse(m.isWithItem());
    Assertions.assertTrue(m.isDoorOpen());
    Assertions.assertFalse(m.isCooking());

    Assertions.assertFalse(m.getHeatingConnection().isHeating());
    Assertions.assertTrue(m.getLampConnection().isLampOn());
    Assertions.assertFalse(m.getTurnableConnection().isMoving());

    // Comprobamos que se puede modificar el tiempo y la potencia
    testPower();
    testTimer();

    // Comprobamos las excepciones
    Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_stop());
    Assertions.assertThrows(IllegalStateException.class, () -> m.item_removed());
    Assertions.assertThrows(IllegalStateException.class, () -> m.door_opened());
    Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
    Assertions.assertThrows(IllegalStateException.class, () -> m.tick());

    // Comprobamos que podemos cerrar la puerta y cambia de estado
    m.door_closed();
    Assertions.assertTrue(m.getEstado() instanceof ClosedWithNoItem);

    // Comprobamos que si metemos comida, cambia de estado
    m.setEstado(new OpenWithNoItem(m));
    m.item_placed();
    Assertions.assertTrue(m.getEstado() instanceof OpenWithItem);
}

```

```

@Test
void testClosedWithNoItem() {

    m.setEstado(new ClosedWithNoItem(m));
    Assertions.assertTrue(m.getEstado() instanceof ClosedWithNoItem);

    // Comprobamos que se inicia correctamente
    Assertions.assertFalse(m.isWithItem());
    Assertions.assertFalse(m.isDoorOpen());
    Assertions.assertFalse(m.isCooking());

    Assertions.assertFalse(m.getHeatingConnection().isHeating());
    Assertions.assertFalse(m.getLampConnection().isLampOn());
    Assertions.assertFalse(m.getTurnableConnection().isMoving());

    // Comprobamos que se puede modificar el tiempo y la potencia
    testPower();
    testTimer();

    // Comprobamos las excepciones
    Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_stop());
    Assertions.assertThrows(IllegalStateException.class, () -> m.item_removed());
    Assertions.assertThrows(IllegalStateException.class, () -> m.item_placed());
    Assertions.assertThrows(IllegalStateException.class, () -> m.door_closed());
    Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
    Assertions.assertThrows(IllegalStateException.class, () -> m.tick());

    // Comprobamos que podemos abrir la puerta y cambia de estado
    m.door_opened();
    Assertions.assertTrue(m.getEstado() instanceof OpenWithNoItem);
}

@Test
void testOpenWithItem() {

    m.setEstado(new OpenWithItem(m));
    Assertions.assertTrue(m.getEstado() instanceof OpenWithItem);

    // Comprobamos que se inicia correctamente
    Assertions.assertTrue(m.isWithItem());
    Assertions.assertTrue(m.isDoorOpen());
    Assertions.assertFalse(m.isCooking());

    Assertions.assertFalse(m.getHeatingConnection().isHeating());
    Assertions.assertTrue(m.getLampConnection().isLampOn());
    Assertions.assertFalse(m.getTurnableConnection().isMoving());

    // Comprobamos que se puede modificar el tiempo y la potencia
    testPower();
    testTimer();
}

```

```

    // Comprobamos las excepciones
    Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_stop());
    Assertions.assertThrows(IllegalStateException.class, () -> m.item_placed());
    Assertions.assertThrows(IllegalStateException.class, () -> m.door_opened());
    Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
    Assertions.assertThrows(IllegalStateException.class, () -> m.tick());

    // Comprobamos que podemos cerrar la puerta y cambia de estado
    m.door_closed();
    Assertions.assertTrue(m.getEstado() instanceof ClosedWithItem);

    // Comprobamos que si sacamos comida, cambia de estado
    m.setEstado(new OpenWithItem(m));
    m.item_removed();
    Assertions.assertTrue(m.getEstado() instanceof OpenWithNoItem);
}

@Test
void testClosedWithItem() {

    m.setEstado(new ClosedWithItem(m));
    Assertions.assertTrue(m.getEstado() instanceof ClosedWithItem);

    // Comprobamos que se inicia correctamente
    Assertions.assertTrue(m.isWithItem());
    Assertions.assertFalse(m.isDoorOpen());
    Assertions.assertFalse(m.isCooking());

    Assertions.assertFalse(m.getHeatingConnection().isHeating());
    Assertions.assertFalse(m.getLampConnection().isLampOn());
    Assertions.assertFalse(m.getTurnableConnection().isMoving());

    // Comprobamos que se puede modificar el tiempo y la potencia
    testPower();
    testTimer();

    // Comprobamos las excepciones
    Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_stop());
    Assertions.assertThrows(IllegalStateException.class, () -> m.item_removed());
    Assertions.assertThrows(IllegalStateException.class, () -> m.item_placed());
    Assertions.assertThrows(IllegalStateException.class, () -> m.door_closed());
    Assertions.assertThrows(IllegalStateException.class, () -> m.tick());

    // Comprobamos que podemos abrir la puerta y cambia de estado
    m.door_opened();
    Assertions.assertTrue(m.getEstado() instanceof OpenWithItem);
}

```

```

302 // Comprobamos que lo podemos poner a cocinar si tiene la potencia
303 // y tiempo definido, sino salta una excepcion
304 m.setEstado(new ClosedWithItem(m));
305 Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
306
307 m.setPower(1);
308 m.setTimer(1);
309 m.cooking_start();
310 Assertions.assertTrue(m.getEstado() instanceof Cooking);
311 }
312
313 @Test
314 void testCooking() {
315
316     m.setEstado(new ClosedWithItem(m));
317     m.setPower(10);
318     m.setTimer(10);
319     m.cooking_start();
320     Assertions.assertTrue(m.getEstado() instanceof Cooking);
321
322     // Comprobamos que se inicia correctamente
323     Assertions.assertTrue(m.isWithItem());
324     Assertions.assertFalse(m.isDoorOpen());
325     Assertions.assertTrue(m.isCooking());
326
327     Assertions.assertTrue(m.getHeatingConnection().isHeating());
328     Assertions.assertTrue(m.getLampConnection().isLampOn());
329     Assertions.assertTrue(m.getTurnableConnection().isMoving());
330
331     // Comprobamos que si el tiempo se reinicia para y cambia de estado
332     m.timer_reset();
333     Assertions.assertFalse(m.isCooking());
334     Assertions.assertTrue(m.getEstado() instanceof ClosedWithItem);
335
336     // Comprobamos que si la potencia se reinicia 0 para y cambia de estado
337     m.setTimer(10);
338     m.cooking_start();
339     m.power_reset();
340     Assertions.assertFalse(m.isCooking());
341     Assertions.assertTrue(m.getEstado() instanceof ClosedWithItem);
342
343     // Comprobamos que podemos cambiar la potencia y el tiempo mientras cocina
344     m.setTimer(1);
345     m.setPower(10);
346     m.cooking_start();
347
348     m.power_desc();
349     Assertions.assertEquals(9, m.getPower());
350
351     m.power_inc();
352     Assertions.assertEquals(10, m.getPower());
353
354     m.timer_inc();
355     Assertions.assertEquals(2, m.getTimer());
356
357     m.timer_desc();
358     Assertions.assertEquals(1, m.getTimer());
359
360     // Comprobamos que si el tiempo llega a 0 se para y cambia de estado
361     m.timer_desc();
362     Assertions.assertFalse(m.isCooking());
363     Assertions.assertTrue(m.beepListener.times(3));
364
365     // Comprobamos que al abrir la puerta se para y cambia de estado
366     m.setEstado(new Cooking(m));
367     m.door_opened();
368     Assertions.assertFalse(m.isCooking());
369     Assertions.assertTrue(m.getEstado() instanceof OpenWithItem);
370
371 }
372 }
373

```

Resultados:

Runs: 12/12

Errors: 0

Failures: 0

Test_Microwave [Runner: JUnit 5] (0,020 s)

heatingTest() (0,000 s)

testOpenWithItem() (0,000 s)

lampTest() (0,000 s)

testPower() (0,000 s)

testTimer() (0,000 s)

testOpenWithNoItem() (0,002 s)

testClosedWithItem() (0,004 s)

testClosedWithNoItem() (0,003 s)

testCooking() (0,002 s)

testBeeper() (0,001 s)

testDisplay() (0,002 s)

turnableTest() (0,001 s)

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

Resultados:

```
[INFO] Tests run: 53, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.528 s - in hellocucumber.RunCucumberTest
[INFO] Running microwave.Test_Microwave
[INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.101 s - in microwave.Test_Microwave
[INFO] Results:
[INFO] Tests run: 65, Failures: 0, Errors: 0, Skipped: 0
[INFO] BUILD SUCCESS
[INFO] Total time: 3.362 s
[INFO] Finished at: 2022-06-09T21:34:08+02:00
```


StepDefinitions:

```
1 package hellocucumber;
2
3 import io.cucumber.java.en.Given;
11
12 public class StepDefinitions {
13
14     private Microwave m;
15
16     /*
17      * Given
18      */
19
20     @Given("closed empty microwave")
21     public void closedEmpty() {
22         m = new Microwave();
23     }
24
25     @Given("closed full microwave")
26     public void closedFull() {
27         openedFull();
28         m.door_closed();
29     }
30
31     @Given("opened empty microwave")
32     public void openedEmpty() {
33         closedEmpty();
34         m.door_opened();
35     }
36
37     @Given("opened full microwave")
38     public void openedFull() {
39         openedEmpty();
40         m.item_placed();
41     }
42
43     @Given("cooking microwave with {int} power and {int} time")
44     public void cooking(Integer power, Integer time) {
45         closedFull();
46         setTimer(time);
47         setPower(power);
48         m.cooking_start();
49     }
50
51     /*
52      * When
53      */
54
55     @When("open the door")
56     public void openDoor() {
57         m.door_opened();
58     }
```



```
60 @When("close the door")
61 public void closeDoor() {
62     m.door_closed();
63 }
64
65 @When("place food")
66 public void placeFood() {
67     m.item_placed();
68 }
69
70 @When("remove food")
71 public void removeFood() {
72     m.item_removed();
73 }
74
75 @When("set the timer to {int}")
76 public void setTimer(Integer t) {
77     m.timer_reset();
78     for (int i = 0; i < t; i++) {
79         m.timer_inc();
80     }
81 }
82
83 @When("set the power to {int}")
84 public void setPower(Integer p) {
85     m.power_reset();
86     for (int i = 0; i < p; i++) {
87         m.power_inc();
88     }
89 }
90
91 @When("increase timer")
92 public void incTimer() {
93     m.timer_inc();
94 }
95
96 @When("increase power")
97 public void incPower() {
98     m.power_inc();
99 }
100
101 @When("decrease timer")
102 public void decTimer() {
103     m.timer_desc();
104 }
105
106 @When("decrease power")
107 public void decPower() {
108     m.power_desc();
109 }
110
```

```

111 @When("reset timer")
112 public void resetTimer() {
113     m.timer_reset();
114 }
115
116 @When("reset power")
117 public void resetPower() {
118     m.power_reset();
119 }
120
121 @When("start cooking")
122 public void startCooking() {
123     try {
124         m.cooking_start();
125     } catch (IllegalStateException ex) {
126         notStartCooking();
127     }
128 }
129
130 @When("timer goes down {int} seconds")
131 public void timePasses(Integer t) {
132     for (int i = 0; i < t; i++) {
133         m.tick();
134     }
135 }
136
137
138 /*
139  * Then
140  */
141
142 @Then("microwave doesnt start cooking")
143 public void notStartCooking() {
144     Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
145 }
146
147
148 @Then("microwave starts cooking")
149 public void StartCooking() {
150     Assertions.assertTrue(m.isCooking());
151 }
152
153 @Then("microwave isnt cooking")
154 public void notCooking() {
155     Assertions.assertFalse(m.isCooking());
156 }
157
158 @Then("door opens")
159 public void doorOpens() {
160     Assertions.assertTrue(m.isDoorOpen());
161 }
162

```

```

163 @Then("door closes")
164 public void doorCloses() {
165     Assertions.assertFalse(m.isDoorOpen());
166 }
167
168 @Then("light turns on")
169 public void lightOn() {
170     Assertions.assertTrue(m.getLampConnection().isLampOn());
171 }
172
173 @Then("light turns off")
174 public void lightOff() {
175     Assertions.assertFalse(m.getLampConnection().isLampOn());
176 }
177
178 @Then("microwave is full")
179 public void isFull() {
180     Assertions.assertTrue(m.isWithItem());
181 }
182
183 @Then("microwave is empty")
184 public void isEmpty() {
185     Assertions.assertFalse(m.isWithItem());
186 }
187
188 @Then("turnable turns")
189 public void turnableT() {
190     Assertions.assertTrue(m.getTurnableConnection().isMoving());
191 }
192
193 @Then("turnable not turns")
194 public void turnableNT() {
195     Assertions.assertFalse(m.getTurnableConnection().isMoving());
196 }
197
198 @Then("power is zero")
199 public void powerZero() {
200     Assertions.assertEquals("0", m.getDisplayConnection().getDisplay());
201 }
202
203 @Then("timer is zero")
204 public void timerZero() {
205     Assertions.assertEquals("0", m.getDisplayConnection().getDisplay());
206 }
207
208 @Then("heating on")
209 public void heatingOn() {
210     Assertions.assertTrue(m.getHeatingConnection().isHeating());
211 }
212
213 @Then("heating off")
214 public void heatingOff() {
215     Assertions.assertFalse(m.getHeatingConnection().isHeating());
216 }
217
218 @Then("display shows {string}")
219 public void displayShows(String s) {
220     Assertions.assertEquals(s, m.getDisplayConnection().getDisplay());
221 }
222
223 @Then("beeper sounds {int} times")
224 public void soundTimes(Integer t) {
225     Assertions.assertTrue(beepListener.times(t));
226 }
227 }
228

```

Features:

ClosedWithItem:

ClosedWithItem.feature ×

1 Feature: Closed full microwave

2

3

4 Scenario: Open a closed microwave

5 Given closed full microwave

6 When open the door

7 Then door opens

8 And light turns on

9 And heating off

10 And turnable not turns

11

12 Scenario: Reset power

13 Given closed full microwave

14 When reset power

15 Then power is zero

16

17 Scenario: Reset timer

18 Given closed full microwave

19 When reset timer

20 Then timer is zero

21

22 Scenario: Try to cook

23 Given closed full microwave

24 When set the power to 5

25 And set the timer to 5

26 And start cooking

27 Then microwave starts cooking

28

29 Scenario Outline: Set the timer

30 Given closed full microwave

31 When set the timer to <a>

32 Then display shows ""

33

34 Examples:

35 | a | b |

36 | -5 | 0 |

37 | 0 | 0 |

38 | 5 | 5 |

39

40 Scenario Outline: Set the power

41 Given closed full microwave

42 When set the power to <a>

43 Then display shows ""

44

45 Examples:

46 | a | b |

47 | -5 | 0 |

48 | 0 | 0 |

49 | 5 | 5 |

50

ClosedWithNoItem:

```
1 Feature: Closed empty microwave
2
3
4 Scenario: Open a closed microwave
5   Given closed empty microwave
6   When open the door
7   Then door opens
8   And light turns on
9   And heating off
10  And turnable not turns
11
12 Scenario: Reset power
13   Given closed empty microwave
14   When reset power
15   Then power is zero
16
17 Scenario: Reset timer
18   Given closed empty microwave
19   When reset timer
20   Then timer is zero
21
22 Scenario: Try to cook
23   Given closed empty microwave
24   When set the power to 5
25   And set the timer to 5
26   And start cooking
27   Then microwave doesnt start cooking
28
29 Scenario Outline: Set the timer
30   Given closed empty microwave
31   When set the timer to <a>
32   Then display shows "<b>"
33
34 Examples:
35   | a | b |
36   | -5 | 0 |
37   | 0 | 0 |
38   | 5 | 5 |
39
40 Scenario Outline: Set the power
41   Given closed empty microwave
42   When set the power to <a>
43   Then display shows "<b>"
44
45 Examples:
46   | a | b |
47   | -5 | 0 |
48   | 0 | 0 |
49   | 5 | 5 |
50
```

OpenWithNoItem:

```
1 Feature: Opened empty microwave
2
3
4 Scenario: Place food in the microwave
5   Given opened empty microwave
6   When place food
7   Then microwave is full
8
9 Scenario: Close an opened microwave
10  Given opened empty microwave
11  When close the door
12  Then door closes
13  And light turns off
14  And heating off
15  And turnable not turns
16
17 Scenario: Reset power
18   Given opened empty microwave
19   When reset power
20   Then power is zero
21
22 Scenario: Reset timer
23   Given opened empty microwave
24   When reset timer
25   Then timer is zero
26
27 Scenario: Try to cook
28   Given opened empty microwave
29   When set the power to 5
30   And set the timer to 5
31   And start cooking
32   Then microwave doesnt start cooking
33
34 Scenario Outline: Set timer
35   Given opened empty microwave
36   When set the timer to <a>
37   Then display shows "<b>"
38
39 Examples:
40   | a | b |
41   | -5 | 0 |
42   | 0 | 0 |
43   | 5 | 5 |
44
45 Scenario Outline: Set power
46   Given opened empty microwave
47   When set the power to <a>
48   Then display shows "<b>"
49
50 Examples:
51   | a | b |
52   | -5 | 0 |
53   | 0 | 0 |
54   | 5 | 5 |
55
```

OpenWithItem:

Feature: Opened full microwave

Scenario: Remove food from the microwave

Given opened full microwave

When remove food

Then microwave is empty

Scenario: Close an opened microwave

Given opened full microwave

When close the door

Then door closes

And light turns off

And heating off

And turnable not turns

Scenario: Reset power

Given opened full microwave

When reset power

Then power is zero

Scenario: Reset timer

Given opened full microwave

When reset timer

Then timer is zero

Scenario: Try to cook

Given opened full microwave

When set the power to 5

And set the timer to 5

And start cooking

Then microwave doesnt start cooking

Scenario Outline: Set timer

Given opened full microwave

When set the timer to <a>

Then display shows ""

Examples:

a	b
-5	0
0	0
5	5

Scenario Outline: Set power

Given opened full microwave

When set the power to <a>

Then display shows ""

Examples:

a	b
-5	0
0	0
5	5

Cooking:

Feature: microwave cooking

Scenario: start cooking

- Given closed full microwave
- When set the power to 5
- And set the timer to 5
- And start cooking
- Then microwave starts cooking
- And light turns on
- And heating on
- And turnable turns

Scenario: increase power

- Given cooking microwave with 5 power and 5 time
- When increase power
- Then display shows "6"

Scenario: decrease power

- Given cooking microwave with 5 power and 5 time
- When decrease power
- Then display shows "4"

Scenario: increase timer

- Given cooking microwave with 5 power and 5 time
- When increase timer
- Then display shows "6"

Scenario: decrease timer

- Given cooking microwave with 5 power and 5 time
- When decrease timer
- Then display shows "4"

Scenario: stop cooking after resetting the power

- Given cooking microwave with 5 power and 5 time
- When reset power
- Then microwave isnt cooking
- And heating off
- And light turns off
- And turnable not turns
- And display shows "0"

Scenario: stop cooking after resetting the timer

- Given cooking microwave with 5 power and 5 time
- When reset timer
- Then microwave isnt cooking
- And heating off
- And light turns off
- And turnable not turns
- And display shows "0"

Scenario: stop cooking if the door is opened
Given cooking microwave with 5 power and 5 time
When open the door
Then door opens
And microwave isnt cooking
And heating off
And light turns on
And turnable not turns

Scenario Outline: cooking stop if timer finishes
Given closed full microwave
When set the power to <a>
And set the timer to
And start cooking
And timer goes down <c> seconds
Then microwave isnt cooking
And light turns off
And turnable not turns
And heating off
And beeper sounds 3 times
And display shows "Listo"

Examples:

a	b	c
10	5	5
10	100	100
10	15	15