

PROYECTO **MICROONDAS**

PABLO MOLINA SÁNCHEZ

<https://github.com/pablomolinasanchez/Proyecto-Microondas>

IMPLEMENTACIÓN DEL SISTEMA

El sistema está formado por las siguientes clases:

- Componentes
 - **Beeper**
 - **Lamp**
 - **Turntable**
 - **Heating**
 - **Display**
- **Microwave**: clase principal que implementa las funcionalidades del sistema
- **State**: clase abstracta que sirve de “plantilla” a los estados que son:
 - **ClosedWithNoltem**
 - **OpenWithNoltem**
 - **OpenWithItem**
 - **ClosedWithItem**
 - **Cooking**

COMPONENTES

Beeper

El beeper avisa cuando el temporizador haya llegado a cero. La funcionalidad representada en la siguiente imagen hace que la campana suene tantas veces como se indique.

```
public class Beeper {  
    private Integer beep;  
    public void beep(Integer d) {  
        beep=d;  
    }  
}
```

Lamp

La lámpara se enciende o apaga en función de distintos eventos, como pueden ser que la puerta esté abierta o que el microondas esté cocinando. También implementamos una funcionalidad que nos permita saber si la lámpara está encendida

```
public class Lamp {  
    private boolean lampOn=false;  
  
    public void lamp_on() {  
        lampOn=true;  
    }  
  
    public void lamp_off() {  
        lampOn=false;  
    }  
  
    public boolean isLampOn() {  
        return lampOn;  
    }  
}
```

Turntable

El plato giratorio gira, cuando se está cocinando, o se para, cuando se abre la puerta o se acaba el tiempo. Implementamos una funcionalidad que nos permita saber si el plato está girando, y la activación o el apagado de este giro.

```
public class Turntable {  
    private boolean turntableOn=false;  
  
    public void turntable_start() {  
        turntableOn=true;  
    }  
  
    public void turntable_stop() {  
        turntableOn=false;  
    }  
  
    public boolean isMoving() {  
        return turntableOn;  
    }  
}
```

Heating

Es el encargado de calentar la comida a una determinada potencia. Puede calentar (encenderse) y no (apagarse). Implementamos una funcionalidad para saber si está encendido o no. También implementamos que podamos establecer la potencia y conocerla.

```
public class Heating {  
    private boolean heating=false;  
    private Integer power=0;  
  
    public void heating_on() {  
        heating=true;  
    }  
  
    public void heating_off() {  
        heating=false;  
    }  
  
    public void setPower(Integer power) {  
        this.power=power;  
    }  
  
    public boolean isHeating() {  
        return heating;  
    }  
  
    public Integer getPower() {  
        return power;  
    }  
}
```

Display

Su función principal es mostrar el tiempo restante del temporizador, así como los ajustes de la potencia y el temporizador. Implementamos una funcionalidad que nos permita limpiarlo y también otra que nos permita escribir lo que deseemos.

```
public class Display {  
    private String display;  
  
    public void clearDisplay() {  
        display="";  
    }  
  
    public void setDisplay(String s) {  
        display=s;  
    }  
  
    public String getDisplay() {  
        return display;  
    }  
}
```

MICROWAVE

Esta será la clase principal y a través de ésta llamaremos a los métodos de los distintos estados. Inicializamos las variables de cada componente, así como una variable estado que nos servirá para, como anteriormente he dicho, llamar a los métodos de estos estados.

```
public class Microwave {  
    private boolean doorOpen=false;  
    private Integer power;  
    private Integer timer;  
    private boolean cooking=false;  
    private boolean withItem=false;  
    private Heating heating;  
    private Lamp lamp;  
    private Display display;  
    private Turntable turntable;  
    private Beeper beeper;  
    private State state;
```

Por defecto, se creará un microondas con la puerta cerrada, potencia y temporizador a 0, sin cocinar, sin comida, con calentador, lámpara, display, beeper y plato giratorio, y por defecto en el estado ClosedWithNoItem, ya que es, como nos marca el ejercicio, el estado normal de un microondas sin usar.

```
public Microwave() {  
    doorOpen=false;  
    power=0;  
    timer=0;  
    cooking=false;  
    withItem=false;  
    heating=new Heating();  
    lamp=new Lamp();  
    display=new Display();  
    turntable=new Turntable();  
    beeper=new Beeper();  
    state=new ClosedWithNoItem(this);  
  
}
```

Como he dicho antes, esta clase servirá para llamar a los métodos de los diferentes estados, como podemos ver a continuación con los procedimientos de abrir/cerrar puerta y quitar/colocar comida.

```
public void door_opened() {
    state.door_opened(this);
}

public void door_closed() {
    state.door_closed(this);
}

public void item_placed() {
    state.item_placed(this);
}

public void item_removed() {
    state.item_removed(this);
}
```

El aumento/reducción de potencia/tiempo se realizará en esta clase sumando/restando 50 kwh o 5 seg. Como es de esperar, no se aceptarán valores negativos y se elevará excepción en tal caso, al igual que si superamos el tiempo/potencia máximos de 60 minutos/300 kwh. El reseteo de potencia/temporizador se llevará a cabo llamando al método según el estado.

```
public void power_inc() {
    if(power==300) {
        throw new RuntimeException("No puede incrementar la potencia porque "
            + "excederías la potencia del microondas");
    }else {
        power=power+50;
    }
}

public void power_dec() {
    if(power>=50) {
        power=power-50;
    }else {
        throw new RuntimeException("No existe una potencia negativa");
    }
}

public void power_reset() {
    state.power_reset(this);
}

public void timer_inc() {
    if(timer==3600) {
        throw new RuntimeException("No puede incrementar el tiempo porque "
            + "excederías el tiempo del microondas");
    }else {
        timer=timer+5;
    }
}

public void timer_dec() {
    if(timer>=5) {
        timer=timer-5;
    }else {
        throw new RuntimeException("No existe un temporizador negativo");
    }
}

public void timer_reset() {
    state.timer_reset(this);
}
```

La función del temporizador al cocinar y del empezar/parar de cocinar también llamarán al método del estado.

```
public void tick() {
    state.tick(this);
}

public void cooking_start() {
    cooking=true;
    state.cooking_start(this);
}

public void cooking_stop() {
    cooking=false;
    state.cooking_stop(this);
}
```

Por último, la clase Microwave también tiene getters y setters de cada una de sus variables. Estas funcionalidades serán de gran ayuda para los métodos en los distintos estados.

STATE

La clase State es una clase abstracta que sirve como plantilla a los diferentes estados que tendrá el microondas. Esta clase State tiene la definición de métodos como abrir/cerrar la puerta, colocar/quitar comida, parar/empezar a cocinar, reseteo/aumento/incremento de la potencia y tiempo, la función tick() que será un temporizador cuando empiezemos a cocinar.

```
public abstract class State {
    public abstract void door_opened(Microwave m);
    public abstract void door_closed(Microwave m);
    public abstract void item_placed(Microwave m);
    public abstract void item_removed(Microwave m);
    public abstract void cooking_start(Microwave m);
    public abstract void cooking_stop(Microwave m);
    public abstract void timer_reset(Microwave m);
    public abstract void power_reset(Microwave m);
    public abstract void tick(Microwave m);
    public abstract void power_desc(Microwave m);
    public abstract void timer_desc(Microwave m);
    public abstract void power_inc(Microwave m);
    public abstract void timer_inc(Microwave m);
}
```

ClosedWithNoItem

Esta clase implementa el estado del microondas cuando está cerrado sin comida. Por ello, en el constructor definimos sus características: puerta cerrada, sin cocinar, sin comida, el calentador apagado, la lámpara apagada, y el plato giratorio parado.

```
public ClosedWithNoItem(Microwave m) {
    m.setCooking(false);
    m.setItem(false);
    m.setDoorOpen(false);
    m.getHeating().heating_off();
    m.getLamp().lamp_off();
    m.getTurntable().turntable_stop();
}
```

Implementamos ahora los métodos definidos en la clase abstracta State:

- **abrir/cerrar puerta:** si abrimos la puerta del microondas nos cambiará de estado a OpenWithNoItem, mientras que no podemos cerrar una puerta ya cerrada, por lo que elevará una excepción del tipo RuntimeException con el mensaje (“La puerta ya está cerrada”)

```
@Override
public void door_opened(Microwave m) {
    m.setState(new OpenWithNoItem(m));
}

@Override
public void door_closed(Microwave m) {
    throw new RuntimeException("La puerta ya está cerrada");
}
```

- **colocar/quitar comida:** no podemos colocar ni quitar comida ya que para ello la puerta tendría que estar abierta, por lo que se eleva excepción debido a que en este estado la puerta se encuentra cerrada.

```
@Override
public void item_placed(Microwave m) {
    throw new RuntimeException("Para colocar comida debe abrir la puerta");
}

@Override
public void item_removed(Microwave m) {
    throw new RuntimeException("Para quitar comida debe abrir la puerta");
}
```

- **empezar/parar de cocinar:** no podemos cocinar sin comida dentro, al igual que no podemos parar de cocinar si anteriormente no hemos empezado a hacerlo.

```

@Override
public void cooking_start(Microwave m) {
    throw new RuntimeException("No puede cocinar sin comida");
}

@Override
public void cooking_stop(Microwave m) {
    throw new RuntimeException("Actualmente no estás cocinando");
}

```

- **reseteo/reducción/incremento de la potencia/tiempo:** esto podremos hacerlo en cualquier estado. El reseteo consiste en colocar la /potencia/tiempo a 0, el incremento/reducción consiste en aumentar/reducir la potencia/tiempo en 50 kwh/5 seg. El valor de la potencia/tiempo aparecerá en el display con cualquier acción.

```

@Override
public void timer_reset(Microwave m) {
    m.setTimer(0);
    m.getDisplay().setDisplay(Integer.toString(m.getTimer()));
}

@Override
public void power_reset(Microwave m) {
    m.setPower(0);
    m.getDisplay().setDisplay(Integer.toString(m.getPower()));
}

@Override
public void power_desc(Microwave m) {
    m.power_dec();
    m.getDisplay().setDisplay(Integer.toString(m.getPower()));
}

@Override
public void timer_desc(Microwave m) {
    m.timer_dec();
    m.getDisplay().setDisplay(Integer.toString(m.getTimer()));
}

@Override
public void power_inc(Microwave m) {
    m.power_inc();
    m.getDisplay().setDisplay(Integer.toString(m.getPower()));
}

@Override
public void timer_inc(Microwave m) {
    m.timer_inc();
    m.getDisplay().setDisplay(Integer.toString(m.getTimer()));
}

```

- **temporizador:** no podemos usar el temporizador si no se está cocinando

```

@Override
public void tick(Microwave m) {
    throw new RuntimeException("Actualmente no estás cocinando, por lo que el contador no puede actuar");
}

```

OpenWithNoItem

Esta clase implementa el estado del microondas cuando está abierto sin comida. Por ello, en el constructor definimos sus características: puerta abierta, sin cocinar, sin comida, el calentador apagado, la lámpara encendida, y el plato giratorio parado.

```
public OpenWithNoItem(Microwave m) {
    m.setCooking(false);
    m.setItem(false);
    m.setDoorOpen(true);
    m.getHeating().heating_off();
    m.getLamp().lamp_on();
    m.getTurntable().turntable_stop();

}
```

Implementamos ahora los métodos definidos en la clase abstracta State:

- **abrir/cerrar puerta:** si cerramos la puerta del microondas nos cambiará de estado a ClosedWithNoItem, mientras que no podemos abrir una puerta ya abierta, por lo que elevará una excepción.

```
@Override
public void door_opened(Microwave m) {
    throw new RuntimeException("La puerta ya está abierta");
}

@Override
public void door_closed(Microwave m) {
    m.setState(new ClosedWithNoItem(m));
}
```

- **colocar/quitar comida:** si colocamos comida nos cambia de estado a OpenWithItem, pero no podemos quitar comida ya que para ello la puerta tendría que haber comida, por lo que se eleva excepción.

```
@Override
public void item_placed(Microwave m) {
    m.setState(new OpenWithItem(m));
}

@Override
public void item_removed(Microwave m) {
    throw new RuntimeException("Todavía no hay comida");
}
```

- **empezar/parar de cocinar:** no podemos cocinar sin comida dentro, al igual que no podemos parar de cocinar si anteriormente no hemos empezado a hacerlo.

```

@Override
public void cooking_start(Microwave m) {
    throw new RuntimeException("No puede cocinar sin comida");
}

@Override
public void cooking_stop(Microwave m) {
    throw new RuntimeException("Actualmente no estás cocinando");
}

```

- **reseteo/reducción/incremento de la potencia/tiempo:** esto podremos hacerlo en cualquier estado. El reseteo consiste en colocar la /potencia/tiempo a 0, el incremento/reducción consiste en aumentar/reducir la potencia/tiempo en 50 kwh/5 seg. El valor de la potencia/tiempo aparecerá en el display con cualquier acción.

```

@Override
public void timer_reset(Microwave m) {
    m.setTimer(0);
    m.getDisplay().setDisplay(Integer.toString(m.getTimer()));
}

@Override
public void power_reset(Microwave m) {
    m.setPower(0);
    m.getDisplay().setDisplay(Integer.toString(m.getPower()));
}

@Override
public void power_desc(Microwave m) {
    m.power_dec();
    m.getDisplay().setDisplay(Integer.toString(m.getPower()));
}

@Override
public void timer_desc(Microwave m) {
    m.timer_dec();
    m.getDisplay().setDisplay(Integer.toString(m.getTimer()));
}

@Override
public void power_inc(Microwave m) {
    m.power_inc();
    m.getDisplay().setDisplay(Integer.toString(m.getPower()));
}

@Override
public void timer_inc(Microwave m) {
    m.timer_inc();
    m.getDisplay().setDisplay(Integer.toString(m.getTimer()));
}

```

- **temporizador:** no podemos usar el temporizador si no se está cocinando

```

@Override
public void tick(Microwave m) {
    throw new RuntimeException("Actualmente no estás cocinando, por lo que el contador no puede actuar");
}

```

OpenWithItem

Esta clase implementa el estado del microondas cuando está abierto con comida. Por ello, en el constructor definimos sus características: puerta abierta, con comida, el calentador apagado, la lámpara encendida, y el plato giratorio parado.

```
public OpenWithItem(Microwave m) {
    m.setCooking(false);
    m.setItem(true);
    m.setDoorOpen(true);
    m.getHeating().heating_off();
    m.getLamp().lamp_on();
    m.getTurntable().turntable_stop();

}
```

Implementamos ahora los métodos definidos en la clase abstracta State:

- **abrir/cerrar puerta:** si cerramos la puerta del microondas nos cambiará de estado a ClosedWithItem, mientras que no podemos abrir una puerta ya abierta, por lo que elevará una excepción.

```
@Override
public void door_opened(Microwave m) {
    throw new RuntimeException("La puerta ya está abierta");
}

@Override
public void door_closed(Microwave m) {
    m.setState(new ClosedWithItem(m));
}

}
```

- **colocar/quitar comida:** si quitamos comida nos cambia de estado a OpenWithNoItem, pero no podemos colocar comida ya que ya tenemos comida colocada, por lo que se eleva excepción.

```
@Override
public void item_placed(Microwave m) {
    throw new RuntimeException("Ya tienes comida colocada");
}

@Override
public void item_removed(Microwave m) {
    m.setState(new OpenWithNoItem(m));
}

}
```

- **empezar/parar de cocinar:** no podemos cocinar con la puerta abierta, al igual que no podemos parar de cocinar si anteriormente no hemos empezado a hacerlo.

```

@Override
public void cooking_start(Microwave m) {
    throw new RuntimeException("Para cocinar debes tener la puerta cerrada");
}

@Override
public void cooking_stop(Microwave m) {
    throw new RuntimeException("Aún no estás cocinando");
}

```

- **reseteo/reducción/incremento de la potencia/tiempo:** esto podremos hacerlo en cualquier estado. El reseteo consiste en colocar la /potencia/tiempo a 0, el incremento/reducción consiste en aumentar/reducir la potencia/tiempo en 50 kwh/5 seg. El valor de la potencia/tiempo aparecerá en el display con cualquier acción.

```

@Override
public void timer_reset(Microwave m) {
    m.setTimer(0);
    m.getDisplay().setDisplay(Integer.toString(m.getTimer()));
}

@Override
public void power_reset(Microwave m) {
    m.setPower(0);
    m.getDisplay().setDisplay(Integer.toString(m.getPower()));
}

@Override
public void power_desc(Microwave m) {
    m.power_dec();
    m.getDisplay().setDisplay(Integer.toString(m.getPower()));
}

@Override
public void timer_desc(Microwave m) {
    m.timer_dec();
    m.getDisplay().setDisplay(Integer.toString(m.getTimer()));
}

@Override
public void power_inc(Microwave m) {
    m.power_inc();
    m.getDisplay().setDisplay(Integer.toString(m.getPower()));
}

@Override
public void timer_inc(Microwave m) {
    m.timer_inc();
    m.getDisplay().setDisplay(Integer.toString(m.getTimer()));
}

```

- **temporizador:** no podemos usar el temporizador si no se está cocinando

```

@Override
public void tick(Microwave m) {
    throw new RuntimeException("Actualmente no estás cocinando, por lo que el contador no puede actuar");
}

```

ClosedWithItem

Esta clase implementa el estado del microondas cuando está cerrado con comida. Por ello, en el constructor definimos sus características: puerta cerrada, sin cocinar, con comida, el calentador apagado, la lámpara apagada, y el plato giratorio parado.

```
public ClosedWithItem(Microwave m) {
    m.setCooking(false);
    m.setItem(true);
    m.setDoorOpen(false);
    m.getHeating().heating_off();
    m.getLamp().lamp_off();
    m.getTurntable().turntable_stop();
    m.getDisplay().clearDisplay();
}
```

Implementamos ahora los métodos definidos en la clase abstracta State:

- **abrir/cerrar puerta:** si abrimos la puerta del microondas nos cambiará de estado a OpenWithItem, mientras que no podemos cerrar una puerta ya cerrada, por lo que elevará una excepción.

```
@Override
public void door_opened(Microwave m) {
    m.setState(new OpenWithItem(m));
}

@Override
public void door_closed(Microwave m) {
    throw new IllegalStateException("Puerta cerrada");
}
```

- **colocar/quitar comida:** no podemos colocar ni quitar comida ya que para ello la puerta tendría que estar abierta, por lo que se eleva excepción debido a que en este estado la puerta se encuentra cerrada.

```
@Override
public void item_placed(Microwave m) {
    throw new RuntimeException("Puerta cerrada");
}

@Override
public void item_removed(Microwave m) {
    throw new RuntimeException("Puerta cerrada");
}
```

- **empezar/parar de cocinar:** si queremos cocinar primero tendremos que ver si la potencia y el temporizador están valorados por encima de 0. Si lo están cambiaremos de estado a Cooking. Si la potencia y el temporizador no están valorizados por encima de 0 elevará excepción. Tampoco podemos parar de cocinar si anteriormente no hemos empezado a hacerlo.

```

@Override
public void cooking_start(Microwave m) {
    if(m.getTimer() > 0 && m.getPower() > 0) {
        m.setState(new Cooking(m));
    } else {
        throw new RuntimeException("Debe seleccionar tiempo y potencia adecuados");
    }
}

@Override
public void cooking_stop(Microwave m) {
    throw new RuntimeException("No estamos cocinando");
}

```

- **reseteo/reducción/incremento de la potencia/tiempo:** esto podremos hacerlo en cualquier estado. El reseteo consiste en colocar la /potencia/tiempo a 0, el incremento/reducción consiste en aumentar/reducir la potencia/tiempo en 50 kwh/5 seg. El valor de la potencia/tiempo aparecerá en el display con cualquier acción.

```

@Override
public void timer_reset(Microwave m) {
    m.setTimer(0);
    m.getDisplay().setDisplay(Integer.toString(m.getTimer()));
}

@Override
public void power_reset(Microwave m) {
    m.setPower(0);
    m.getDisplay().setDisplay(Integer.toString(m.getPower()));
}

@Override
public void power_desc(Microwave m) {
    m.power_dec();
    m.getDisplay().setDisplay(Integer.toString(m.getPower()));
}

@Override
public void timer_desc(Microwave m) {
    m.timer_dec();
    m.getDisplay().setDisplay(Integer.toString(m.getTimer()));
}

@Override
public void power_inc(Microwave m) {
    m.power_inc();
    m.getDisplay().setDisplay(Integer.toString(m.getPower()));
}

@Override
public void timer_inc(Microwave m) {
    m.timer_inc();
    m.getDisplay().setDisplay(Integer.toString(m.getTimer()));
}

```

- **temporizador:** no podemos usar el temporizador si no se está cocinando

```

@Override
public void tick(Microwave m) {
    throw new RuntimeException("Actualmente no estás cocinando, por lo que el contador no puede actuar");
}

```

ClosedWithItem

Esta clase implementa el estado del microondas cuando está cocinando. Por ello, en el constructor definimos sus características: puerta cerrada, cocinando, con comida, el calentador encendido, la lámpara encendida, y el plato giratorio girando.

```
public Cooking(Microwave m) {
    m.setCooking(true);
    m.setWithItem(true);
    m.setDoorOpen(false);
    m.getHeating().heating_on();
    m.getLamp().lamp_on();
    m.getTurntable().turntable_start();

}
```

Implementamos ahora los métodos definidos en la clase abstracta State:

- **abrir/cerrar puerta:** si abrimos la puerta del microondas nos cambiará de estado a OpenWithItem, mientras que no podemos cerrar una puerta ya cerrada, por lo que elevará una excepción.

```
@Override
public void door_opened(Microwave m) {
    m.setState(new OpenWithItem(m));
}

@Override
public void door_closed(Microwave m) {
    throw new RuntimeException("La puerta ya está cerrada");
}
```

- **colocar/quitar comida:** no podemos colocar ni quitar comida ya que para ello la puerta tendría que estar abierta, por lo que se eleva excepción.

```
@Override
public void item_placed(Microwave m) {
    throw new RuntimeException("Puerta cerrada");
}

@Override
public void item_removed(Microwave m) {
    throw new RuntimeException("Puerta cerrada");
}
```

- **empezar/parar de cocinar:** no podemos empezar a cocinar porque ya lo estamos haciendo, por lo que elevamos excepción. Si paramos de cocinar cambiamos de estado a ClosedWithItem.

```

@Override
public void cooking_start(Microwave m) {
    throw new RuntimeException("Ya estás cocinando");
}

@Override
public void cooking_stop(Microwave m) {
    m.setState(new ClosedWithItem(m));
}

```

- **reseteo/reducción/incremento de la potencia/tiempo:** esto podremos hacerlo en cualquier estado. El reseteo consiste en colocar la potencia/tiempo a 0. La reducción consiste en reducir la potencia/tiempo en 50 kwh/5 seg. Si la potencia/tiempo alcanza valor de 0, se deja de cocinar y se cambia de estado a ClosedWithItem. El aumento funciona igual que en el resto de estados. El valor de la potencia/tiempo aparecerá en el display con cualquier acción.

```

@Override
public void timer_reset(Microwave m) {
    m.setTimer(0);
    m.setState(new ClosedWithItem(m));
}

@Override
public void power_reset(Microwave m) {
    m.setPower(0);
    m.setState(new ClosedWithItem(m));
}

@Override
public void power_desc(Microwave m) {
    m.power_dec();
    m.getDisplay().setDisplay(Integer.toString(m.getPower()));
    if(m.getPower()==0) {
        m.setState(new ClosedWithItem(m));
    }
}

@Override
public void timer_desc(Microwave m) {
    m.timer_dec();
    Beeper beep=new Beeper();
    beep.beep(3);
    if(m.getBeeper()==beep) {
        m.setState(new ClosedWithItem(m));
        m.getDisplay().setDisplay("Comida lista");
    } else {
        m.getDisplay().setDisplay(Integer.toString(m.getTimer()));
    }
}

@Override
public void power_inc(Microwave m) {
    m.power_inc();
    m.getDisplay().setDisplay(Integer.toString(m.getPower()));
}

@Override
public void timer_inc(Microwave m) {
    m.timer_inc();
    m.getDisplay().setDisplay(Integer.toString(m.getTimer()));
}

```

- **temporizador**: utilizamos un objeto de tipo Timer que con scheduleAtFixedRate() realizaremos la misma acción hasta que cancelemos el timer. Así, el tiempo que hayamos elegido irá bajando cada de uno en uno cada 1000ms como marcamos al final de la función. Cuando se haya repetido el proceso (variable i) el mismo numero de veces que tiempo hayamos elegido, se cancelará el timer, crearemos un beeper que avise tres veces del acabado del cocinado y mostraremos por el display que la comida está lista.

```
@Override  
public void tick(Microwave m) {  
  
    Timer timer=new Timer();  
    timer.scheduleAtFixedRate(new TimerTask() {  
        int i=1;  
        public void run() {  
            m.getDisplay().setDisplay(Integer.toString(m.getTimer()-1));  
            i++;  
  
            if (m.getTimer()-i == 0) {  
                timer.cancel();  
                Beeper beeper=new Beeper();  
                beeper.beep(3);  
                m.setBeeper(beeper);  
                m.getDisplay().setDisplay("Comida lista");  
            }  
        }  
    }, 0, 1000);  
}
```

PRUEBAS UNITARIAS

Realizaremos pruebas unitarias de cada uno de los componentes del microondas, así como de cada uno de sus estados. Para ello creamos un objeto de tipo Microwave(). Comenzamos con el **Beeper**, el cual veremos si se inicializa correctamente:

```
@Test  
void testBeeper() {  
    // Vemos si se crea correctamente el beeper  
    Beeper beeper= new Beeper();  
    beeper.beep(3);  
    Assertions.assertNotNull(peeper);  
  
}
```

Seguimos con el **Display** en el que vemos si escribe mensajes correctamente y si al limpiarlo se nos vacía:

```
@Test  
void testDisplay() {  
    Display display = new Display();  
  
    // Vemos si imprime bien mensajes  
    display.setDisplay("Prueba en el display");  
    Assertions.assertEquals("Prueba en el display", display.getDisplay());  
  
    // Vemos si al limpiar la pantalla esta se queda vacía  
    display.clearDisplay();  
    Assertions.assertEquals("", display.getDisplay());  
}
```

Continuamos con el calentador (**Heating**) y vemos si cambia correctamente la potencia y su apagado o encendido ayudándonos de la funcionalidad isHeating():

```
@Test  
void testHeating() {  
    Heating heating = new Heating();  
  
    // Vemos si cambia correctamente la potencia  
    heating.setPower(50);  
    Assertions.assertEquals(50, heating.getPower());  
  
    // Vemos su encendido  
    heating.heating_on();  
    Assertions.assertTrue(heating.isHeating());  
  
    //Vemos su apagado  
    heating.heating_off();  
    Assertions.assertFalse(heating.isHeating());  
}
```

Ahora probamos la lámpara (**Lamp**), probando su apagado o encendido.

```
@Test
void testLamp() {
    Lamp lamp = new Lamp();

    // Vemos el encendido
    lamp.lamp_on();
    Assertions.assertTrue(lamp.isLampOn());

    // Vemos el apagado
    lamp.lamp_off();
    Assertions.assertFalse(lamp.isLampOn());

}
```

Probamos el plato giratorio (**Turtable**) y vemos si funcionan correctamente su giro y parado.

```
@Test
void testTurtable() {
    Turtable turtable = new Turtable();

    // Vemos el inicio
    turtable.turntable_start();
    Assertions.assertTrue(turtable.isMoving());

    // Vemos el parado
    turtable.turntable_stop();
    Assertions.assertFalse(turtable.isMoving());

}
```

Probamos el **Timer** para comprobar si funciona el incremento de tiempo y el reseteo. Además, tenemos que probar la reducción de tiempo teniéndonos que elevar una excepción, ya que como implementamos en la clase Microwave, no podemos tener un timer negativo. Además, probamos que no pueda superar el tiempo máximo de 60 minutos.

```
@Test
void testTimer() {
    // Vemos si funciona el aumentar el tiempo
    m.timer_inc();
    Assertions.assertEquals(5, m.getTimer());

    // Vemos si funciona el resetear el tiempo
    m.timer_reset();
    Assertions.assertEquals(0, m.getTimer());

    // Vemos si lanza excepción si el timer toma valor negativo
    Assertions.assertThrows(RuntimeException.class, () -> m.timer_dec());

    // Vemos si lanza excepción al definir mayor tiempo que el posible
    int i=0;
    while(i<=719) {
        m.timer_inc();
        i++;
    }
    Assertions.assertThrows(RuntimeException.class, () -> m.timer_inc());
}
```

Probamos la potencia (**power**) de igual forma que el timer:

```
@Test
void testPower() {
    // Vemos si funciona el aumentar la potencia
    m.power_inc();
    Assertions.assertEquals(50, m.getPower());

    // Vemos si funciona el resetear la potencia
    m.power_reset();
    Assertions.assertEquals(0, m.getPower());

    // Vemos si lanza excepción si el timer toma valor negativo
    Assertions.assertThrows(RuntimeException.class, () -> m.power_dec());

    // Vemos si lanza excepción al definir mayor potencia que la posible
    int i=0;
    while(i<=5) {
        m.power_inc();
        i++;
    }
    Assertions.assertThrows(RuntimeException.class, () -> m.power_inc());
}
```

Comenzamos con las pruebas de los estados. Empezamos por el estado por defecto del microondas cuando no está en uso: **ClosedWithNoItem**. Vemos si se inicializa correctamente. Coloco en comentarios la inicialización del estado en la clase propia y vemos si esto se lleva a cabo correctamente con asserts. Posteriormente, vemos si lanza las excepciones pertinentes en los métodos que no pueden llevarse a cabo como ya expliqué en la descripción de la clase. Vemos también si al abrir la puerta nos cambia de estado a OpenWithNoItem. Realizo esto en todos y cada uno de los estados.

```
@Test
void testClosedWithNoItem() {
    // Vemos si se inicializa correctamente recordando que el estado
    //tiene las siguientes características
    //m.setCooking(false);
    //m.setItem(false);
    //m.setDoorOpen(false);
    //m.getHeating().heating_off();
    //m.getLamp().lamp_off();
    //m.getTurntable().turntable_stop();

    Assertions.assertFalse(m.isCooking());
    Assertions.assertFalse(m.isWithItem());
    Assertions.assertFalse(m.isDoorOpen());
    Assertions.assertFalse(m.getHeating().isHeating());
    Assertions.assertFalse(m.getLamp().isLampOn());
    Assertions.assertFalse(m.getTurntable().isMoving());
    Assertions.assertTrue(m.getState() instanceof ClosedWithNoItem);

    // Vemos si lanza excepciones
    Assertions.assertThrows(RuntimeException.class, () -> m.door_closed());
    Assertions.assertThrows(RuntimeException.class, () -> m.cooking_start());
    Assertions.assertThrows(RuntimeException.class, () -> m.cooking_stop());
    Assertions.assertThrows(RuntimeException.class, () -> m.item_placed());
    Assertions.assertThrows(RuntimeException.class, () -> m.item_removed());
    Assertions.assertThrows(RuntimeException.class, () -> m.tick());
    //Vemos si cambia de estado cuando abre la puerta
    m.door_opened();
    Assertions.assertTrue(m.getState() instanceof OpenWithNoItem);
}
```

Seguimos con la prueba del estado **OpenWithNoItem**, realizando los mismos pasos. Para obtener este estado, primeramente el objeto Microwave debe abrir su puerta. Vemos también si al cerrar la puerta, cambia de estado a ClosedWithNoItem:

```
@Test
void testOpenWithNoItem() {
    //Abrimos la puerta para obtener el estado
    m.door_opened();

    // Vemos si se inicializa correctamente recordando que el estado
    //tiene las siguientes características
    //m.setCooking(false);
    //m.setItem(false);
    //m.setDoorOpen(true);
    //m.getHeating().heating_off();
    //m.getLamp().lamp_on();
    //m.getTurntable().turntable_stop();
    Assertions.assertFalse(m.isCooking());
    Assertions.assertFalse(m.isWithItem());
    Assertions.assertTrue(m.isDoorOpen());
    Assertions.assertFalse(m.getHeating().isHeating());
    Assertions.assertTrue(m.getLamp().isLampOn());
    Assertions.assertFalse(m.getTurntable().isMoving());
    Assertions.assertTrue(m.getState() instanceof OpenWithNoItem);

    // Vemos si lanza excepciones
    Assertions.assertThrows(RuntimeException.class, () -> m.door_opened());
    Assertions.assertThrows(RuntimeException.class, () -> m.cooking_start());
    Assertions.assertThrows(RuntimeException.class, () -> m.cooking_stop());
    Assertions.assertThrows(RuntimeException.class, () -> m.item_removed());
    Assertions.assertThrows(RuntimeException.class, () -> m.tick());

    m.door_closed();
    Assertions.assertTrue(m.getState() instanceof ClosedWithNoItem);
```

Seguimos con la prueba del estado **OpenWithItem**. Para obtener este estado debemos abrir la puerta y colocar comida. Vemos si cerramos la puerta cambia de estado a ClosedWithItem y si quita la comida a OpenWithNoItem:

```
@Test
void testOpenWithItem() {
    //Abrimos la puerta y metemos comida para obtener el estado
    m.door_opened();
    m.item_placed();
    // Vemos si se inicializa correctamente recordando que el estado
    //tiene las siguientes características
    //m.setCooking(false);
    //m.setItem(true);
    //m.setDoorOpen(true);
    //m.getHeating().heating_off();
    //m.getLamp().lamp_on();
    //m.getTurntable().turntable_stop()
    Assertions.assertFalse(m.isCooking());
    Assertions.assertTrue(m.isWithItem());
    Assertions.assertTrue(m.isDoorOpen());
    Assertions.assertFalse(m.getHeating().isHeating());
    Assertions.assertTrue(m.getLamp().isLampOn());
    Assertions.assertFalse(m.getTurntable().isMoving());
    Assertions.assertTrue(m.getState() instanceof OpenWithItem);

    // Vemos si lanza excepciones
    Assertions.assertThrows(RuntimeException.class, () -> m.door_opened());
    Assertions.assertThrows(RuntimeException.class, () -> m.cooking_stop());
    Assertions.assertThrows(RuntimeException.class, () -> m.item_placed());
    Assertions.assertThrows(RuntimeException.class, () -> m.tick());
    // Vemos si cambia de estado al cerrar la puerta
    m.door_closed();
    Assertions.assertTrue(m.getState() instanceof ClosedWithItem);
    // Vemos si cambia de estado al eliminar la comida
    m.door_opened();
    m.item_removed();
    Assertions.assertTrue(m.getState() instanceof OpenWithNoItem);
```

Ahora, vemos la prueba del **ClosedWithItem**. Para lograr este estado abrimos la puerta, colocamos comida y cerramos de nuevo la puerta. Vemos si al abrir la puerta desde este estado nos cambia a OpenWithItem:

```
@Test
void testClosedWithItem() {
    //Abrimos la puerta, metemos comida y volvemos a cerrar la puerta para obtener
    m.door_opened();
    m.item_placed();
    m.door_closed();
    // Vemos si se inicializa correctamente recordando que el estado
    //tiene las siguientes características
    //m.setCooking(false);
    //m.setItem(true);
    //m.setDoorOpen(false);
    //m.getHeating().heating_off();
    //m.getLamp().lamp_off();
    //m.getTurntable().turntable_stop();
    Assertions.assertThat(m.isCooking()).isFalse();
    Assertions.assertThat(m.isWithItem()).isTrue();
    Assertions.assertThat(m.isDoorOpen()).isFalse();
    Assertions.assertThat(m.getHeating().isHeating()).isFalse();
    Assertions.assertThat(m.getLamp().isLampOn()).isFalse();
    Assertions.assertThat(m.getTurntable().isMoving()).isFalse();
    Assertions.assertThat(m.getState() instanceof ClosedWithItem).isTrue();
    // Vemos si lanza excepciones
    Assertions.assertThatThrownBy(() -> m.door_closed())
        .isInstanceOf(RuntimeException.class);
    Assertions.assertThatThrownBy(() -> m.cooking_start())
        .isInstanceOf(RuntimeException.class);
    Assertions.assertThatThrownBy(() -> m.cooking_stop())
        .isInstanceOf(RuntimeException.class);
    Assertions.assertThatThrownBy(() -> m.item_placed())
        .isInstanceOf(RuntimeException.class);
    Assertions.assertThatThrownBy(() -> m.item_removed())
        .isInstanceOf(RuntimeException.class);
    Assertions.assertThatThrownBy(() -> m.tick())
        .isInstanceOf(RuntimeException.class);

    // Vemos si cambia el estado al abrir la puerta
    m.door_opened();
    Assertions.assertThat(m.getState() instanceof OpenWithItem).isTrue();
}
```

Por último, vemos el estado **Cooking**. Para llegar a ese estado hemos tenido que abrir la puerta, colocar comida, cerrar la puerta y definir potencia y tiempo antes de comenzar a cocinar ya que sino no podemos hacerlo.

```
@Test
void testCooking() {
    //Abrimos la puerta, metemos comida, volvemos a cerrar la puerta
    //y metemos potencia y tiempo para obtener el estado
    m.door_opened();
    m.item_placed();
    m.door_closed();
    m.power_inc();
    m.timer_inc();
    m.cooking_start();

    // Vemos si se inicializa correctamente recordando que el estado
    //tiene las siguientes características
    //m.setCooking(true);
    //m.setItem(true);
    //m.setDoorOpen(false);
    //m.getHeating().heating_on();
    //m.getLamp().lamp_on();
    //m.getTurntable().turntable_start();
    Assertions.assertThat(m.isCooking()).isTrue();
    Assertions.assertThat(m.isWithItem()).isTrue();
    Assertions.assertThat(m.isDoorOpen()).isFalse();
    Assertions.assertThat(m.getHeating().isHeating()).isTrue();
    Assertions.assertThat(m.getLamp().isLampOn()).isTrue();
    Assertions.assertThat(m.getTurntable().isMoving()).isFalse();
    Assertions.assertThat(m.getState() instanceof Cooking).isTrue();

    // Vemos si lanza excepciones
    Assertions.assertThatThrownBy(() -> m.door_closed())
        .isInstanceOf(RuntimeException.class);
    Assertions.assertThatThrownBy(() -> m.cooking_start())
        .isInstanceOf(RuntimeException.class);
    Assertions.assertThatThrownBy(() -> m.item_placed())
        .isInstanceOf(RuntimeException.class);
    Assertions.assertThatThrownBy(() -> m.item_removed())
        .isInstanceOf(RuntimeException.class);
```

Vemos ahora los posibles cambios de estados que se podrían dar en **Cooking**:

```
//Vemos si cambia de estado al abrir la puerta  
m.door_opened();  
Assertions.assertThat(m.getState()).isInstanceOf(OpenWithItem);  
  
//Vemos si cambia de estado al parar de cocinar  
m.door_closed();  
m.cooking_start();  
m.cooking_stop();  
Assertions.assertThat(m.getState()).isInstanceOf(ClosedWithItem);  
  
//Vemos si cambia de estado al resetear el tiempo  
m.timer_reset();  
Assertions.assertThat(m.getState()).isInstanceOf(ClosedWithItem);  
  
//Vemos si cambia de estado al resetear la potencia  
m.timer_inc();  
m.cooking_start();  
m.power_reset();  
Assertions.assertThat(m.getState()).isInstanceOf(ClosedWithItem);  
}
```

PRUEBAS CUCUMBER

Realizamos pruebas para cada uno de los estados. Comenzamos con **ClosedWithNoItem**, en el que probamos si se inicializa correctamente en el primer escenario. En el segundo vemos si al abrir el microondas nos cambia de estado correctamente. En los siguientes dos escenarios vemos el funcionamiento del sistema cuando queremos incrementar tanto la potencia como el tiempo.

```
Scenario: Tenemos un microondas ClosedWithNoItem  
Given ClosedWithNoItem  
When Tengo un microondas ClosedWithNoItem  
Then la luz está apagada  
And el plato no gira  
And no tiene comida  
And no calienta  
And la puerta cerrada  
And no cocina  
  
Scenario: Abrir un microondas ClosedWithNoItem  
Given ClosedWithNoItem  
When Abro la puerta  
Then Obtenemos un OpenWithNoItem  
  
Scenario: Incrementamos la potencia  
Given ClosedWithNoItem  
When Incrementamos la potencia a los <a> de potencia  
Then Teníamos <a> y ahora <b> de potencia  
Examples:  
| a | b |  
| 0 | 50 |  
| 50 | 100 |  
| 200 | 250 |  
  
Scenario: Incrementamos el tiempo  
Given ClosedWithNoItem  
When Incrementamos el tiempo a los <a> de tiempo  
Then Teníamos <a> y ahora <b> de tiempo  
Examples:  
| a | b |  
| 0 | 5 |  
| 5 | 10 |  
| 20 | 25 |
```

Continuamos con **OpenWithNoItem** en el que vemos si se inicializa correctamente, si cambia de estado al cerrar la puerta o colocar comida y los testeos del sistema del incremento de potencia y tiempo.

```
Scenario: Tenemos un microondas OpenWithNoItem
  Given OpenWithNoItem
  When Tengo un microondas OpenWithNoItem
    Then la luz está encendida|
      And el plato no gira
      And no tiene comida
      And no calienta
      And la puerta abierta
      And no cocina

Scenario: Cerrar un microondas OpenWithNoItem
  Given OpenWithNoItem
  When Cierro la puerta
  Then Obtenemos un ClosedWithNoItem

Scenario: Colocar comida en un microondas OpenWithNoItem
  Given OpenWithNoItem
  When Coloco la comida
  Then Obtenemos un OpenWithItem

Scenario: Incrementamos la potencia
  Given OpenWithNoItem
  When Incrementamos la potencia a los <a> de potencia
  Then Teniamos <a> y ahora <b> de potencia
  Examples:
  | a | b |
  | 0 | 50 |
  | 50 | 100|
  | 200| 250| 

Scenario: Incrementamos el tiempo
  Given OpenWithNoItem
  When Incrementamos el tiempo a los <a> de tiempo
  Then Teniamos <a> y ahora <b> de tiempo
  Examples:
  | a | b |
  | 0 | 5 |
  | 5 | 10| 
```

Seguimos con **OpenWithItem** en el que vemos si se inicializa correctamente, si cambia de estado al cerrar la puerta o quitar comida y los testeos del sistema del incremento de potencia y tiempo.

```
Scenario: Tenemos un microondas OpenWithItem
  Given OpenWithItem
  When Tengo un microondas OpenWithItem
  Then la luz está encendida
  And el plato no gira
  And tiene comida
  And no calienta
  And la puerta abierta
  And no cocina

Scenario: Cerrar un microondas OpenWithItem
  Given OpenWithItem
  When Cierro la puerta
  Then Obtenemos un ClosedWithItem

Scenario: Quitar comida de un microondas OpenWithItem
  Given OpenWithItem
  When Quito comida
  Then Obtenemos un OpenWithNoItem

Scenario: Incrementamos la potencia
  Given OpenWithItem
  When Incrementamos la potencia a los <a> de potencia
  Then Teniamos <a> y ahora <b> de potencia
  Examples:
  | a | b |
  | 0 | 50 |
  | 50 | 100|
  | 200| 250| 

Scenario: Incrementamos el tiempo
  Given OpenWithItem
  When Incrementamos el tiempo a los <a> de tiempo
  Then Teniamos <a> y ahora <b> de tiempo
  Examples:
  | a | b |
  | 0 | 5 |
  | 5 | 10| 
```

Seguimos con **ClosedWithItem** en el que vemos si se inicializa correctamente, si cambia de estado al abrir la puerta y los testeos del sistema del incremento de potencia y tiempo. Además, probamos que si queremos cocinar primero la potencia y el tiempo deben ser mayores a 0.

```

Scenario: Tenemos un microondas ClosedWithItem
Given ClosedWithItem
When Tengo un microondas ClosedWithItem
Then la luz está apagada
And el plato no gira
And tiene comida
And no calienta
And la puerta cerrada
And no cocina

Scenario: Abrir un microondas ClosedWithItem
Given ClosedWithItem
When Abro la puerta
Then Obtenemos un OpenWithItem

Scenario: Incrementamos la potencia
Given ClosedWithItem
When Incrementamos la potencia a los <a> de potencia
Then Teniamos <a> y ahora <b> de potencia
Examples:
| a | b |
| 0 | 50 |
| 50 | 100|
| 200| 250 |

Scenario: Incrementamos el tiempo
Given ClosedWithItem
When Incrementamos el tiempo a los <a> de tiempo
Then Teniamos <a> y ahora <b> de tiempo
Examples:
| a | b |
| 0 | 5 |
| 5 | 10|
| 20| 25|

```

Scenario: Queremos cocinar

Given ClosedWithItem
When La potencia no es cero
And El tiempo no es cero
Then Cocinamos

Por último, vamos con **Cooking** en el que vemos si se inicializa correctamente y si cambia de estado al abrir la puerta, parar de cocinar o resetear potencia o tiempo.

```
Scenario: Tenemos un microondas Cooking
Given Cooking
When Tengo un microondas Cooking
Then la luz está encendida
And el plato gira
And tiene comida
And calienta
And la puerta cerrada
And la potencia mayor a 0
And el tiempo mayor a 0
And cocina

Scenario: Abrir un microondas Cooking
Given Cooking
When Abro la puerta
Then Obtenemos un OpenWithItem

Scenario: Parar de cocinar un microondas Cooking
Given Cooking
When Paro de cocinar
Then Obtenemos un ClosedWithItem

Scenario: Resetear el tiempo de un microondas Cooking
Given Cooking
When Reseteo el tiempo
Then Obtenemos un ClosedWithItem

Scenario: Resetear la potencia de un microondas Cooking
Given Cooking
When Reseteo la potencia
Then Obtenemos un ClosedWithItem
```

Adjunto, por último, la clase con los **StepDefinitions**:

```

@Given("Cooking")
public void Cooking() {
    ClosedWithItem();
    m.door_opened();
    m.item_placed();
    m.door_closed();
    m.power_inc();
    m.timer_inc();
    m.cooking_start();
}

@When("Abro la puerta")
public void abro_puerta() {
    m.door_opened();
}

@When("Cierro la puerta")
public void cierro_puerta() {
    m.door_opened();
}

@When("Pongo de cocinar")
public void paro_cocinar() {
    m.cooking_stop();
}

@When("Reseteo la potencia")
public void reseteo_potencia() {
    m.power_reset();
}

@When("El tiempo no es cero")
public void tiempo_no_cero() {
    Assertions.assertTrue(m.getTimer()>0);
}

@Then("Obtenemos un ClosedWithNoItem")
public void obtenemos_ClosedWithNoItem() {
    Assertions.assertTrue(m.getState() instanceof ClosedWithNoItem);
}

@Then("Obtenemos un ClosedWithItem")
public void obtenemos_ClosedWithItem() {
    Assertions.assertTrue(m.getState() instanceof ClosedWithItem);
}

@Then("Obtenemos un OpenWithNoItem")
public void obtenemos_OpenWithNoItem() {
    Assertions.assertTrue(m.getState() instanceof OpenWithNoItem);
}

@Then("Obtenemos un OpenWithItem")
public void obtenemos_OpenWithItem() {
    Assertions.assertTrue(m.getState() instanceof OpenWithItem);
}

@Then("Obtenemos un Cooking")
public void obtenemos_Cooking() {
    Assertions.assertTrue(m.getState() instanceof Cooking);
}

@Then("Teniamos <a> y ahora <b> de potencia")
public void teniamos_tenemos_potencia() {
    b=m.getPower();
}
}

public class StepDefinitions {
    Microwave m;
    int a;
    int b;

    @Given("ClosedWithNoItem")
    public void ClosedWithNoItem() {
        ClosedWithNoItem();
        m.door_opened();
        m.item_placed();
        m.door_closed();
    }

    @Given("OpenWithNoItem")
    public void OpenWithNoItem() {
        ClosedWithNoItem();
        m.door_opened();
    }

    @Given("ClosedWithItem")
    public void ClosedWithItem() {
        ClosedWithItem();
        m.door_opened();
        m.item_placed();
    }

    @Given("OpenWithItem")
    public void OpenWithItem() {
        ClosedWithNoItem();
        m.door_opened();
        m.item_placed();
    }
}

@When("Reseteo el tiempo")
public void reseteo_tiempo() {
    m.timer_reset();
}

@When("Quito comida")
public void quito_comida() {
    m.item_removed();
}

@When("Coloco la comida")
public void coloco_comida() {
    m.item_placed();
}

@When("Incrementamos la potencia a los <a> de potencia")
public void incrementamos_potencia() {
    a=m.getPower();
    m.power_inc();
}

@When("Incrementamos el tiempo a los <a> de tiempo")
public void incrementamos_tiempo() {
    a=m.getTimer();
    m.timer_inc();
}

@When("La potencia no es cero")
public void potencia_no_cero() {
    Assertions.assertTrue(m.getPower()>0);
}

@Then("Teniamos <a> y ahora <b> de tiempo")
public void teniamos_tenemos_tiempo() {
    b=m.getTimer();
}

@Then("el plato no gira")
public void plato_no_gira() {
    Assertions.assertFalse(m.getTurntable().isMoving());
}

@Then("el plato gira")
public void plato_gira() {
    Assertions.assertTrue(m.getTurntable().isMoving());
}

@Then("la luz está apagada")
public void luz_apagada() {
    Assertions.assertFalse(m.getLamp().isLampOn());
}

@Then("la luz está encendida")
public void luz_encendida() {
    Assertions.assertTrue(m.getLamp().isLampOn());
}

@Then("no tiene comida")
public void no_comida() {
    Assertions.assertFalse(m.isWithItem());
}

@Then("tiene comida")
public void si_comida() {
    Assertions.assertTrue(m.isWithItem());
}

@Then("no calienta")
public void no_calienta() {
    Assertions.assertFalse(m.getHeating().isHeating());
}
}

```

```
public void no_calienta() {
    Assertions.assertThat(m.getHeating().isHeating()).isFalse();
}

@Then("calienta")
public void caliente() {
    Assertions.assertThat(m.getHeating().isHeating()).isTrue();
}

@Then("la puerta abierta")
public void puerta_abierta() {
    Assertions.assertThat(m.isDoorOpen()).isTrue();
}

@Then("la puerta cerrada")
public void puerta_cerrada() {
    Assertions.assertThat(m.isDoorOpen()).isFalse();
}

@Then("no cocina")
public void no_cocina() {
    Assertions.assertThat(m.isCooking()).isFalse();
}

@Then("cocina")
public void cocina() {
    Assertions.assertThat(m.isCooking()).isTrue();
}

@Then("Cocinamos")
public void cocinamos() {
    m.cooking_start();
    Assertions.assertThat(m.getState() instanceof Cooking).isTrue();
}
```