ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Práctica Final: Microondas

Realizado por

Florín Babusca Voicu

UNIVERSIDAD DE MÁLAGA
MÁLAGA, MAYO DE 2022

Apartado a

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

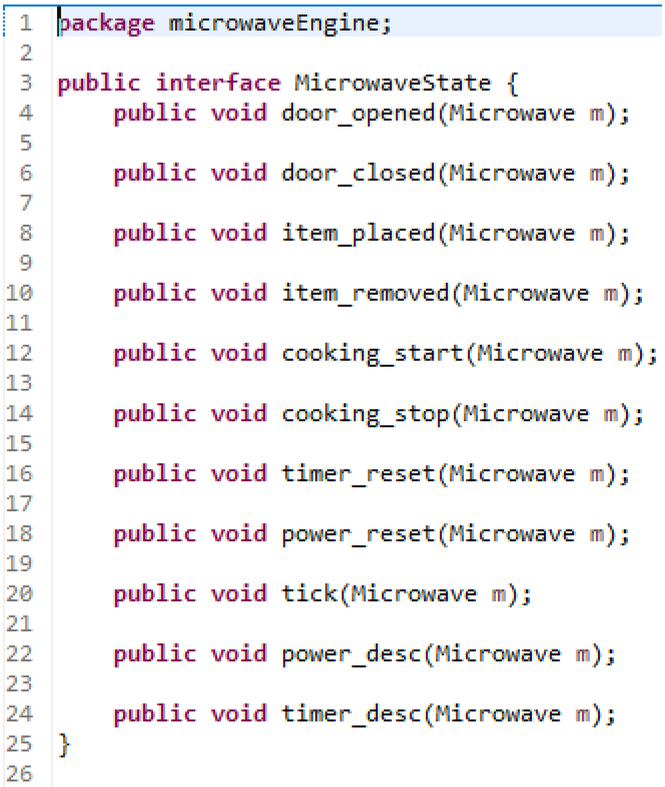
Se han implementado todos los métodos especificados en el diagrama UML del enunciado. No obstante, se han tenido que implementar getters y setters en la clase Microwave o bien porque es necesario que los estados del microondas dispongan de algún mecanismo que modifiquen atributos del microondas o bien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

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

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

Decisión de la interfaz de los estados del microondas

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



```java
package microwaveEngine;

public interface MicrowaveState {
    public void door_opened(Microwave m);

    public void door_closed(Microwave m);

    public void item_placed(Microwave m);

    public void item_removed(Microwave m);

    public void cooking_start(Microwave m);

    public void cooking_stop(Microwave m);

    public void timer_reset(Microwave m);

    public void power_reset(Microwave m);

    public void tick(Microwave m);

    public void power_desc(Microwave m);

    public void timer_desc(Microwave m);
}

```

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

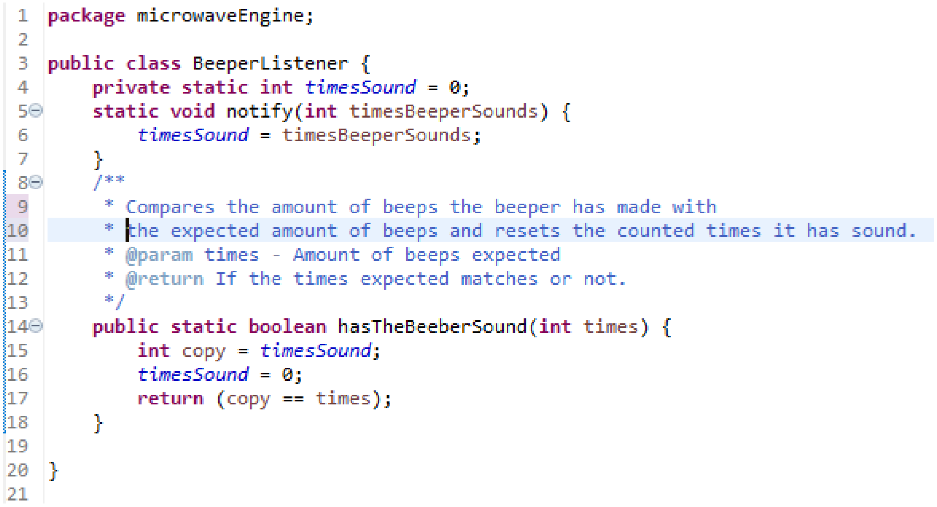
Podría sorprender la elección de los métodos **power_desc()**, **timer_desc()** ya que presentan, a priori, un carácter intrínseco al propio microondas y no tanto a sus estados. No obstante, estos métodos actúan diferentemente cuando el microondas está cocinando, ya que si la potencia o el tiempo tiene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

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

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

Representación del sonido de la bocina

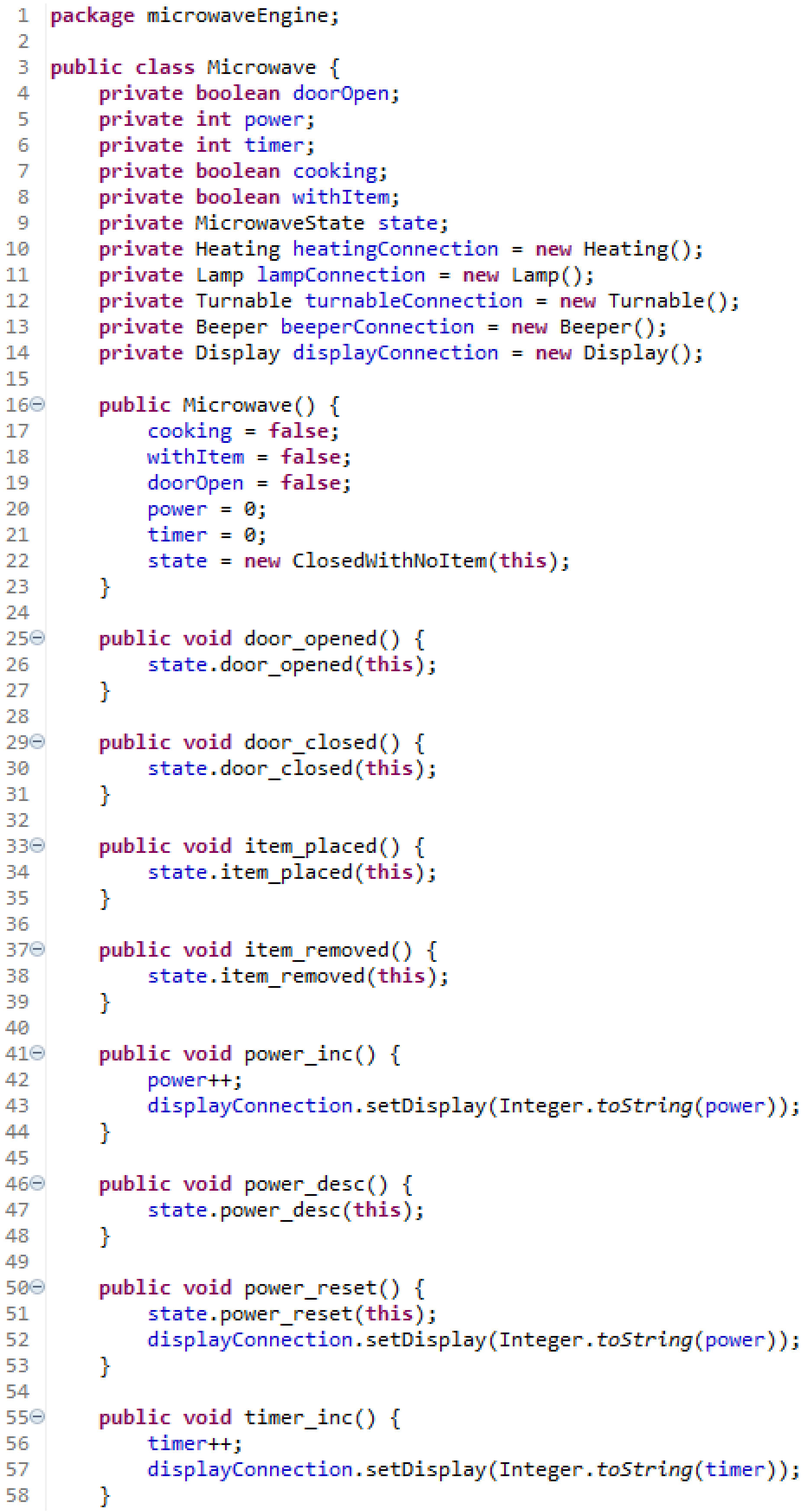
Ha sido bastante interesante determinar cómo representar el sonido del beeper (bocina) del microondas cuando este ha terminado de cocinar. Se ha optado usar el Patrón Observador y por ende, se ha creado una clase pública llamada **BeeperListener** (**vista**) con un método publico estático llamado **notify(int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



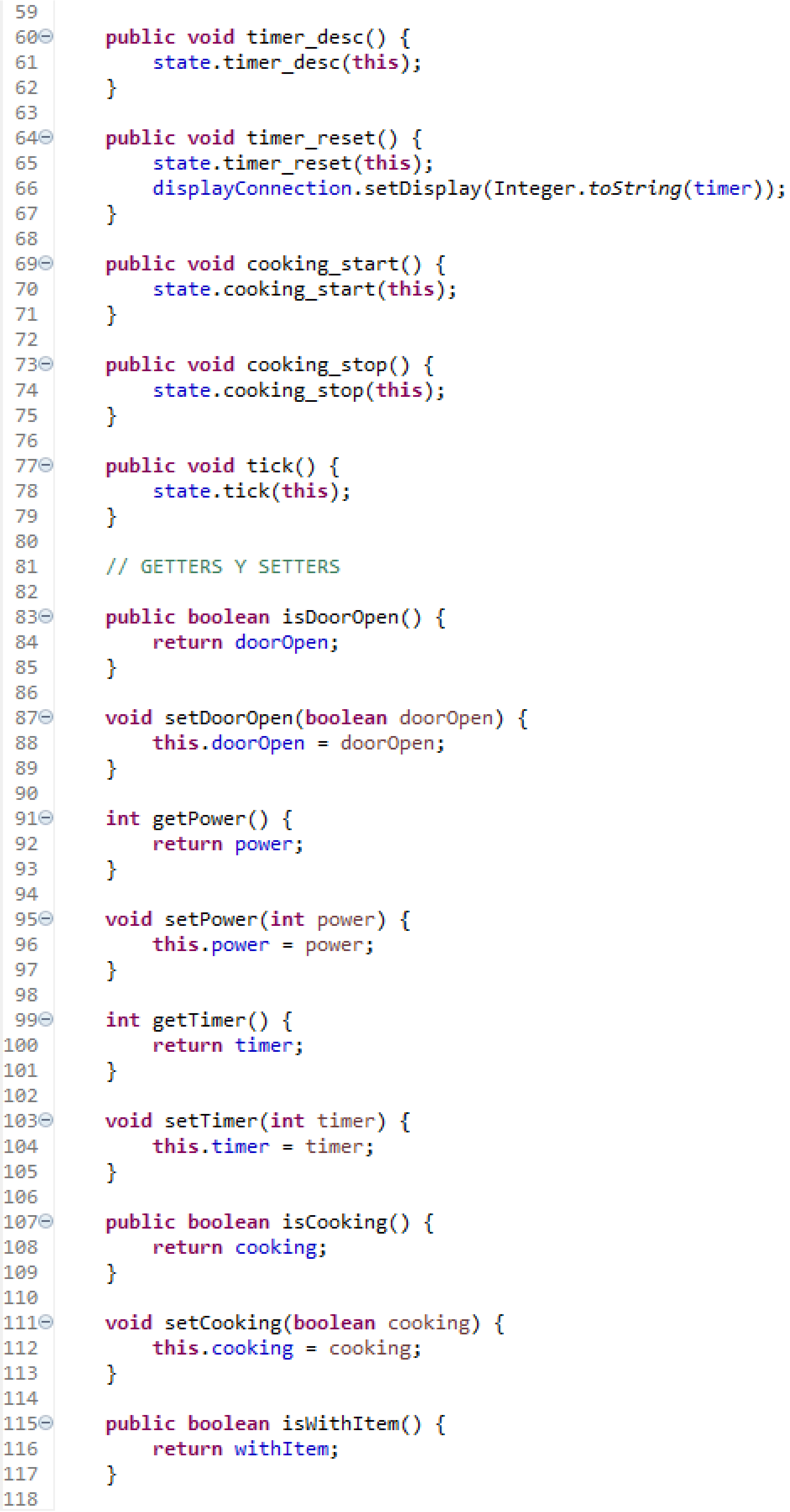
```java
package microwaveEngine;

public class BeeperListener {
    private static int timesSound = 0;
    static void notify(int timesBeeperSounds) {
        timesSound = timesBeeperSounds;
    }
    /**
     * Compares the amount of beeps the beeper has made with
     * the expected amount of beeps and resets the counted times it has sound.
     * @param times - Amount of beeps expected
     * @return If the times expected matches or not.
     */
    public static boolean hasTheBeeberSound(int times) {
        int copy = timesSound;
        timesSound = 0;
        return (copy == times);
    }

}
```

La implementación del microondas es:

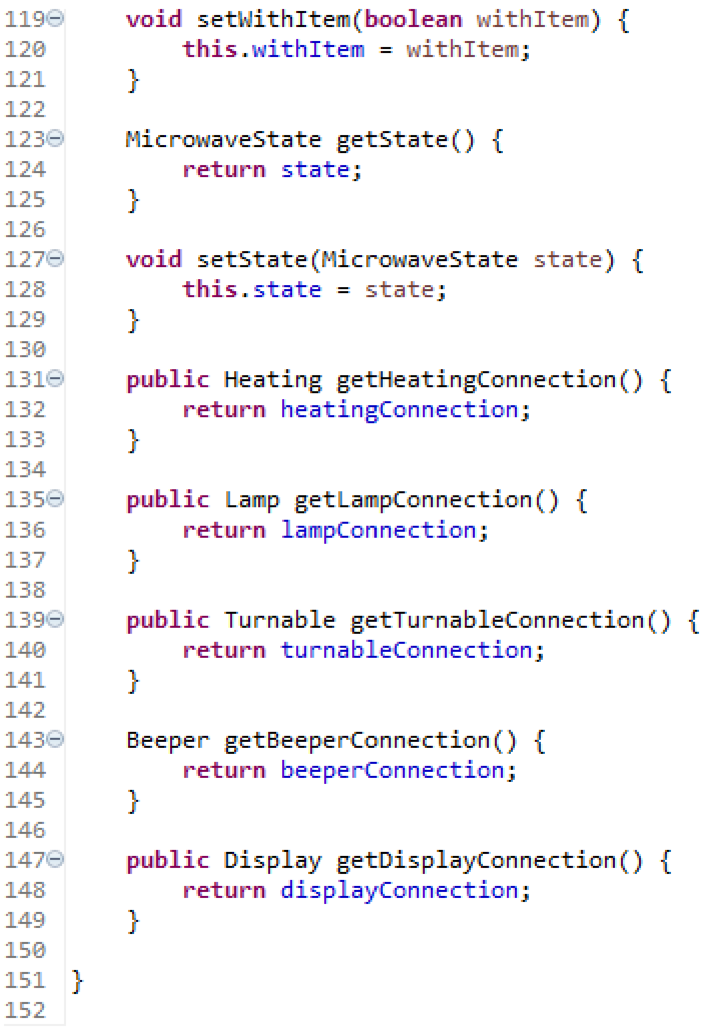
Nota: Se ha asumido que un microondas nuevo es un microondas cerrado sin alimentos y con temporizador y potencia a cero



```java
package microwaveEngine;

public class Microwave {
    private boolean doorOpen;
    private int power;
    private int timer;
    private boolean cooking;
    private boolean withItem;
    private MicrowaveState state;
    private Heating heatingConnection = new Heating();
    private Lamp lampConnection = new Lamp();
    private Turnable turnableConnection = new Turnable();
    private Beeper beeperConnection = new Beeper();
    private Display displayConnection = new Display();

    public Microwave() {
        cooking = false;
        withItem = false;
        doorOpen = false;
        power = 0;
        timer = 0;
        state = new ClosedWithNoItem(this);
    }

    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);
    }

    public void power_inc() {
        power++;
        displayConnection.setDisplay(Integer.toString(power));
    }

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

    public void power_reset() {
        state.power_reset(this);
        displayConnection.setDisplay(Integer.toString(power));
    }

    public void timer_inc() {
        timer++;
        displayConnection.setDisplay(Integer.toString(timer));
    }
```



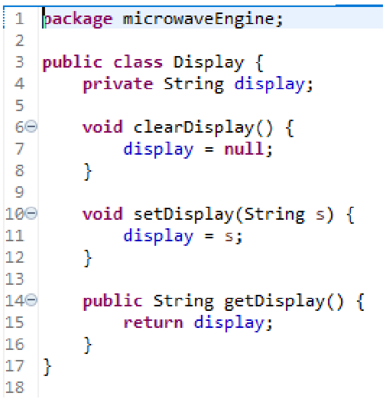
```java

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

    public void timer_reset() {
        state.timer_reset(this);
        displayConnection.setDisplay(Integer.toString(timer));
    }

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

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

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

    // GETTERS Y SETTERS

    public boolean isDoorOpen() {
        return doorOpen;
    }

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

    int getPower() {
        return power;
    }

    void setPower(int power) {
        this.power = power;
    }

    int getTimer() {
        return timer;
    }

    void setTimer(int timer) {
        this.timer = timer;
    }

    public boolean isCooking() {
        return cooking;
    }

    void setCooking(boolean cooking) {
        this.cooking = cooking;
    }

    public boolean isWithItem() {
        return withItem;
    }

```



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

    MicrowaveState getState() {
        return state;
    }

    void setState(MicrowaveState state) {
        this.state = state;
    }

    public Heating getHeatingConnection() {
        return heatingConnection;
    }

    public Lamp getLampConnection() {
        return lampConnection;
    }

    public Turnable getTurnableConnection() {
        return turnableConnection;
    }

    Beeper getBeeperConnection() {
        return beeperConnection;
    }

    public Display getDisplayConnection() {
        return displayConnection;
    }

}

```

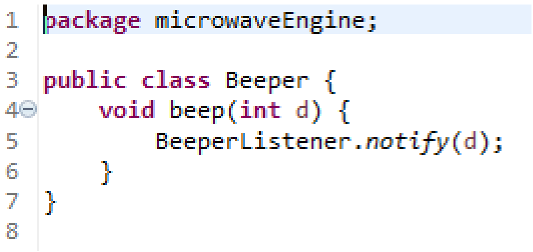
La implementación de la clase Display es:

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



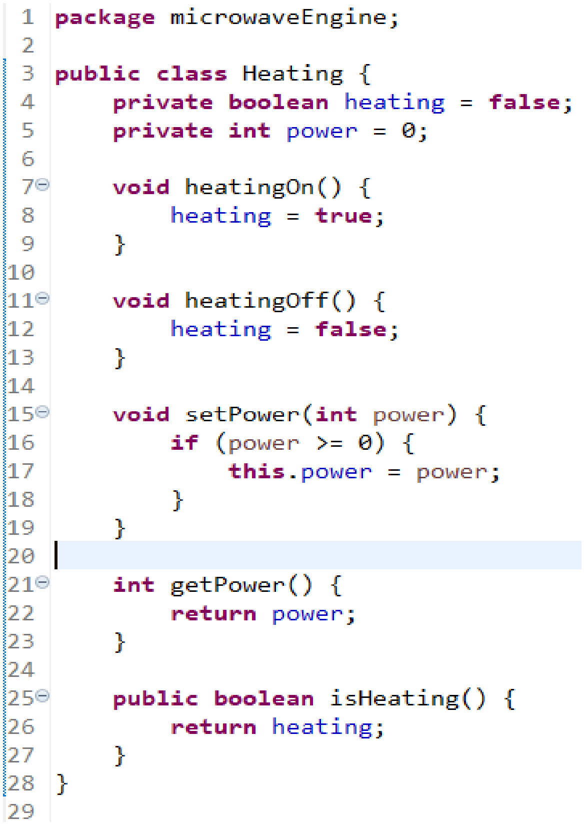
```java
package microwaveEngine;

public class Display {
    private String display;

    void clearDisplay() {
        display = null;
    }

    void setDisplay(String s) {
        display = s;
    }

    public String getDisplay() {
        return display;
    }
}

```

La implementación de la clase Beeper es:



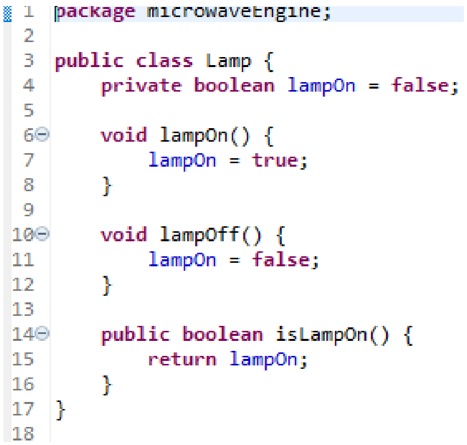
```java
package microwaveEngine;

public class Beeper {
    void beep(int d) {
        BeeperListener.notify(d);
    }
}

```

La implementación de la clase Heating es:



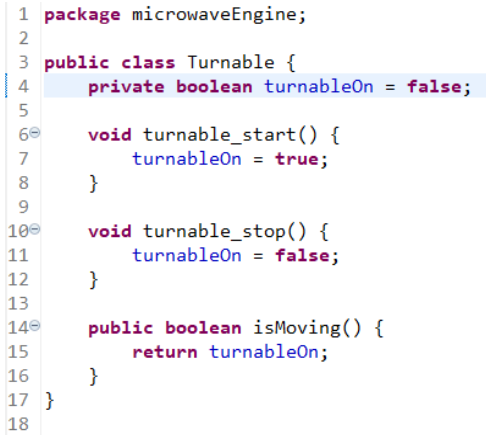
```java
package microwaveEngine;

public class Heating {
    private boolean heating = false;
    private int power = 0;

    void heatingOn() {
        heating = true;
    }

    void heatingOff() {
        heating = false;
    }

    void setPower(int power) {
        if (power >= 0) {
            this.power = power;
        }
    }

    int getPower() {
        return power;
    }

    public boolean isHeating() {
        return heating;
    }
}

```

La implementación de la clase Lamp es:



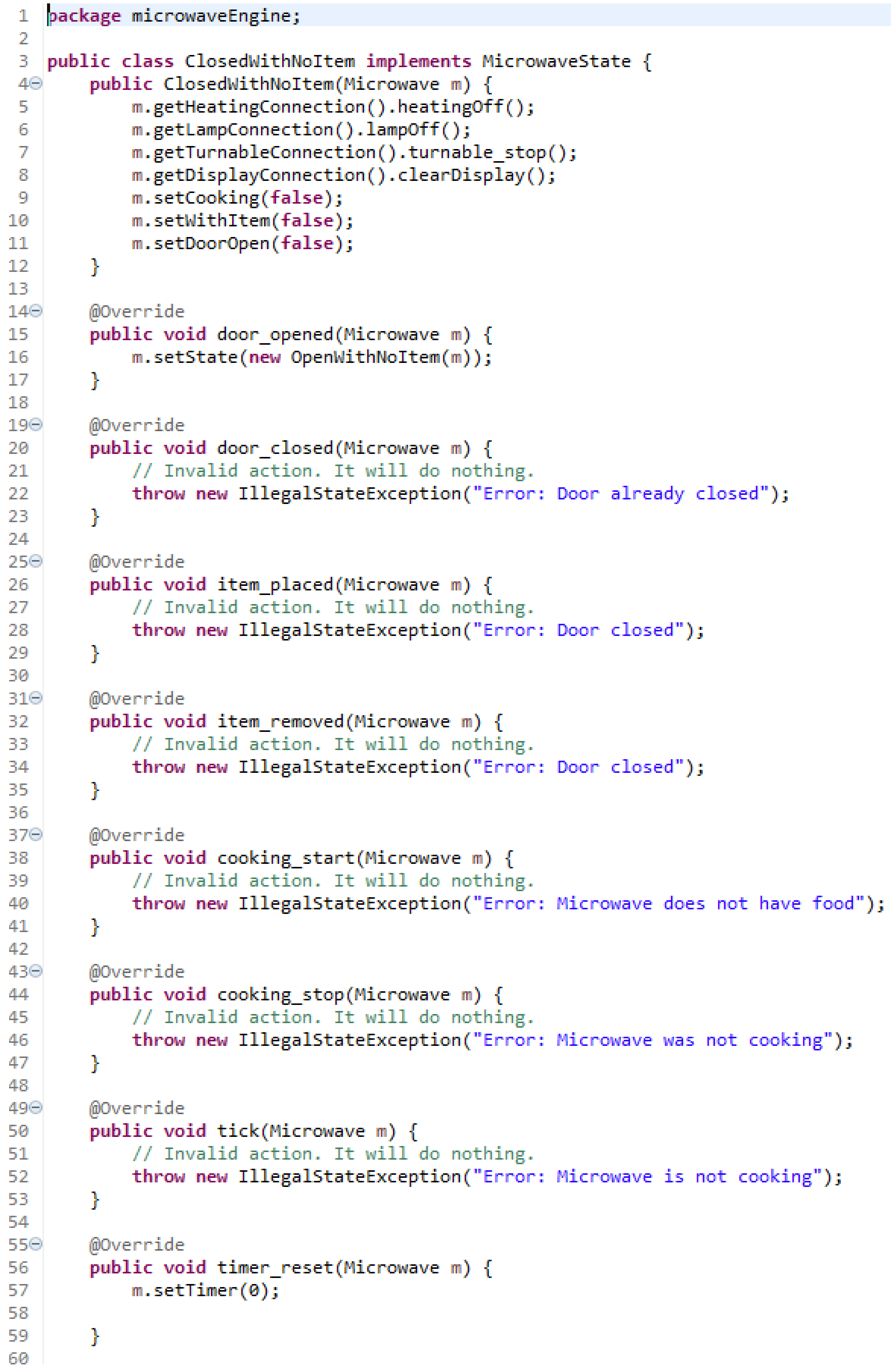
```java
package microwaveEngine;

public class Lamp {
    private boolean lampOn = false;

    void lampOn() {
        lampOn = true;
    }

    void lampOff() {
        lampOn = false;
    }

    public boolean isLampOn() {
        return lampOn;
    }
}

```

La implementación de la clase Turnable es:

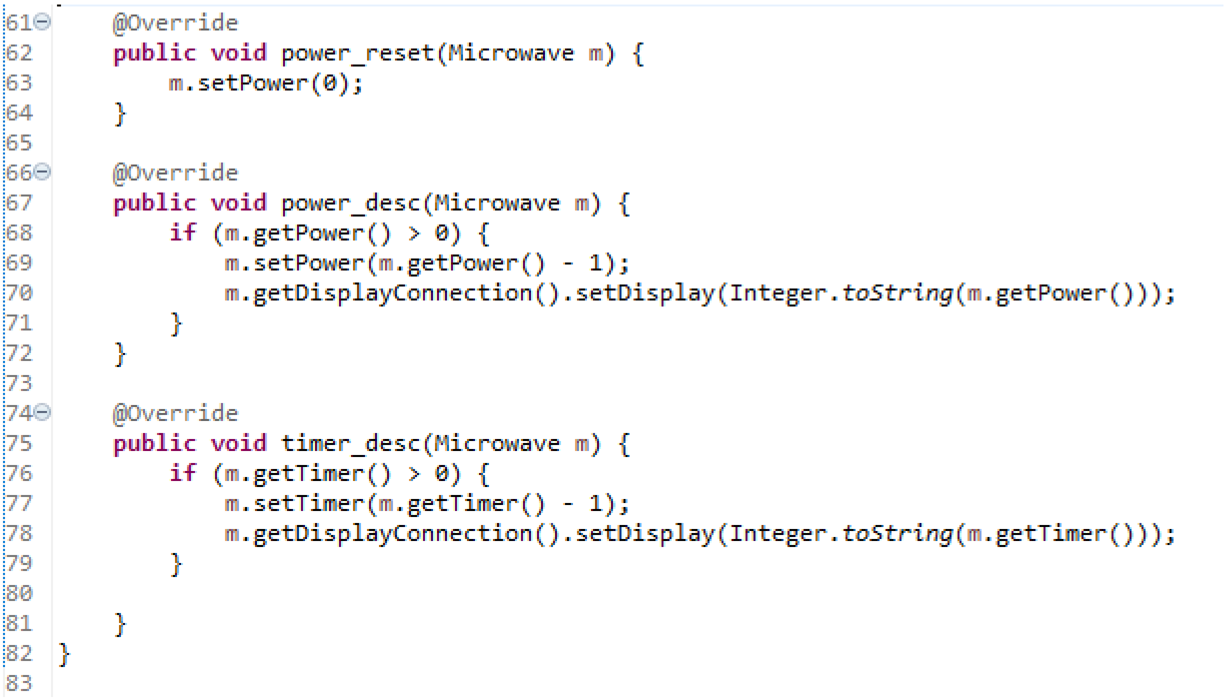


```java
package microwaveEngine;

public class Turnable {
    private boolean turnableOn = false;

    void turnable_start() {
        turnableOn = true;
    }

    void turnable_stop() {
        turnableOn = false;
    }

    public boolean isMoving() {
        return turnableOn;
    }
}

```

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

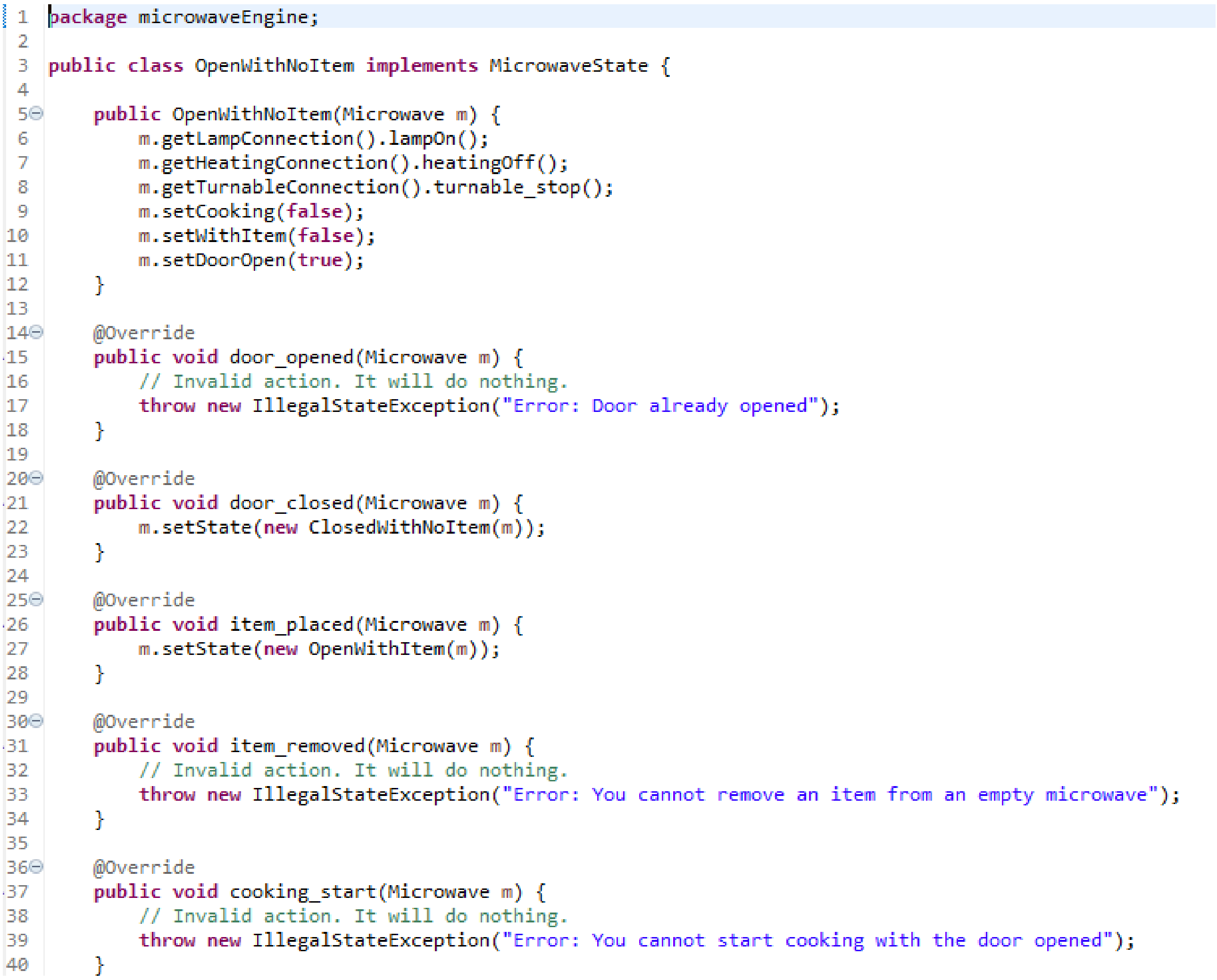


```java
package microwaveEngine;

public class ClosedWithNoItem implements MicrowaveState {
    public ClosedWithNoItem(Microwave m) {
        m.getHeatingConnection().heatingOff();
        m.getLampConnection().lampOff();
        m.getTurnableConnection().turnable_stop();
        m.getDisplayConnection().clearDisplay();
        m.setCooking(false);
        m.setWithItem(false);
        m.setDoorOpen(false);
    }

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

    @Override
    public void door_closed(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Door already closed");
    }

    @Override
    public void item_placed(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Door closed");
    }

    @Override
    public void item_removed(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Door closed");
    }

    @Override
    public void cooking_start(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Microwave does not have food");
    }

    @Override
    public void cooking_stop(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Microwave was not cooking");
    }

    @Override
    public void tick(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Microwave is not cooking");
    }

    @Override
    public void timer_reset(Microwave m) {
        m.setTimer(0);

    }

```

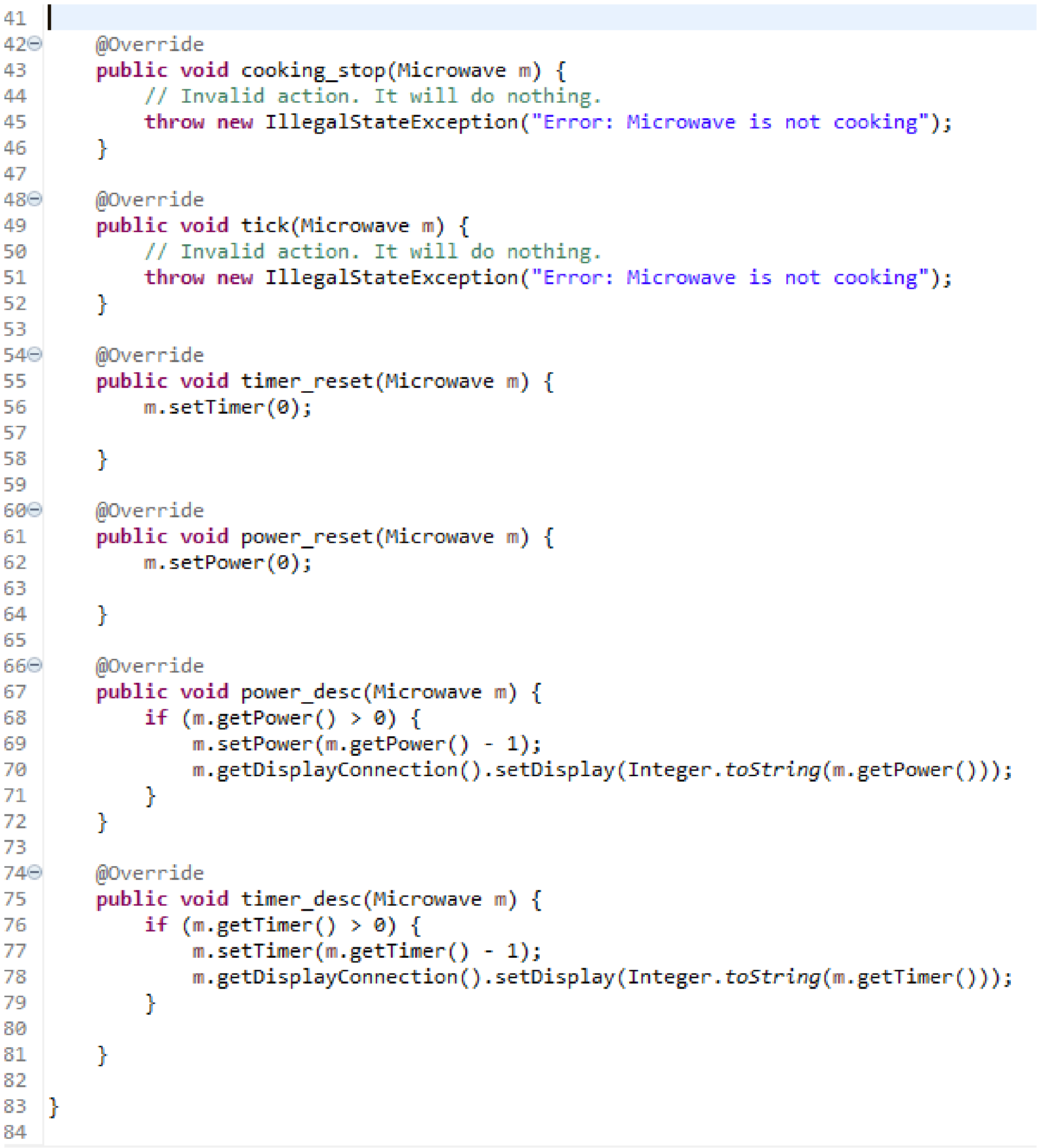


```java
    @Override
    public void power_reset(Microwave m) {
        m.setPower(0);
    }

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

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

    }
}

```

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

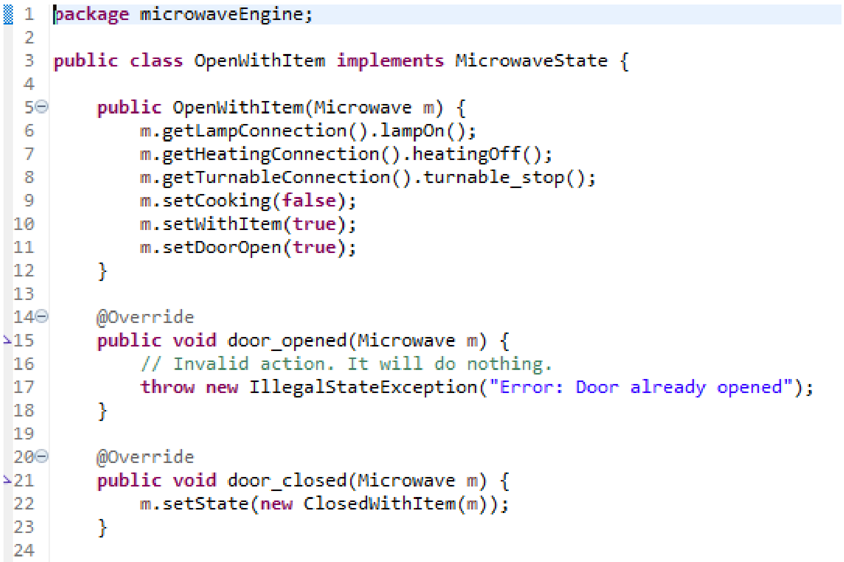


```java
package microwaveEngine;

public class OpenWithNoItem implements MicrowaveState {

    public OpenWithNoItem(Microwave m) {
        m.getLampConnection().lampOn();
        m.getHeatingConnection().heatingOff();
        m.getTurnableConnection().turnable_stop();
        m.setCooking(false);
        m.setWithItem(false);
        m.setDoorOpen(true);
    }

    @Override
    public void door_opened(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Door already opened");
    }

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

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

    @Override
    public void item_removed(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: You cannot remove an item from an empty microwave");
    }

    @Override
    public void cooking_start(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: You cannot start cooking with the door opened");
    }
```

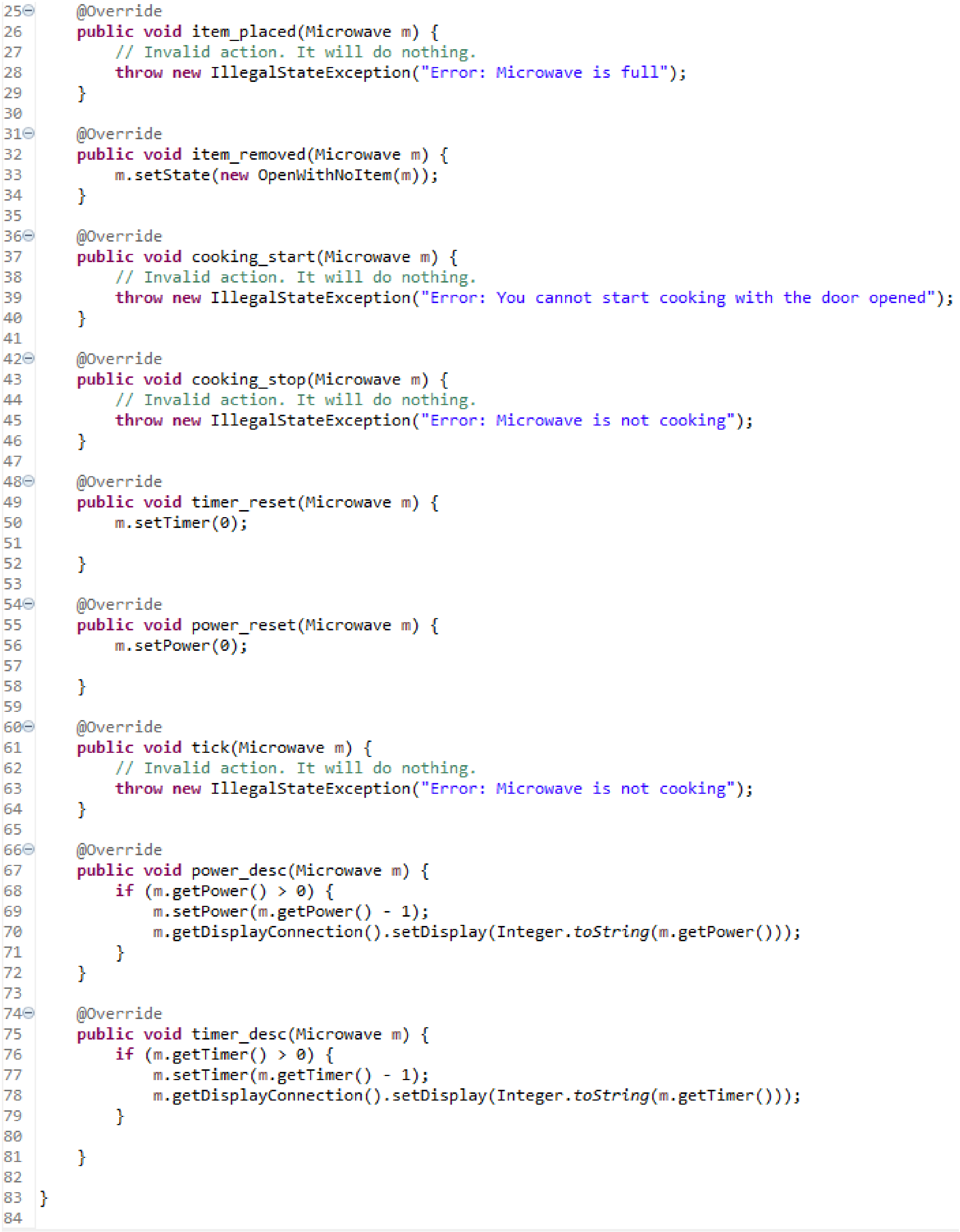


```java

    @Override
    public void cooking_stop(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Microwave is not cooking");
    }

    @Override
    public void tick(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Microwave is not cooking");
    }

    @Override
    public void timer_reset(Microwave m) {
        m.setTimer(0);

    }

    @Override
    public void power_reset(Microwave m) {
        m.setPower(0);

    }

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

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

    }

}

```

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

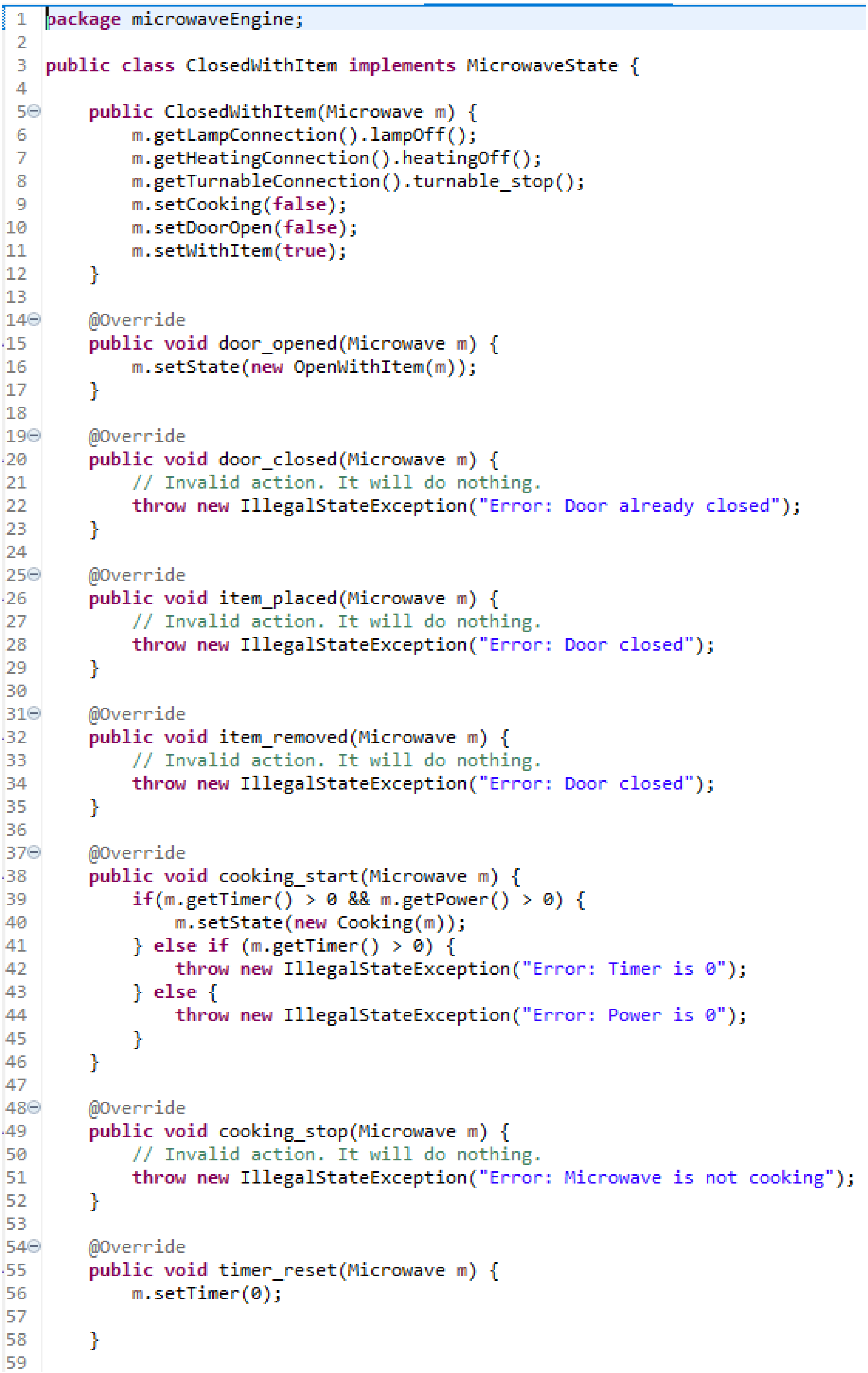


```java
package microwaveEngine;

public class OpenWithItem implements MicrowaveState {

    public OpenWithItem(Microwave m) {
        m.getLampConnection().lampOn();
        m.getHeatingConnection().heatingOff();
        m.getTurnableConnection().turnable_stop();
        m.setCooking(false);
        m.setWithItem(true);
        m.setDoorOpen(true);
    }

    @Override
    public void door_opened(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Door already opened");
    }

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

```

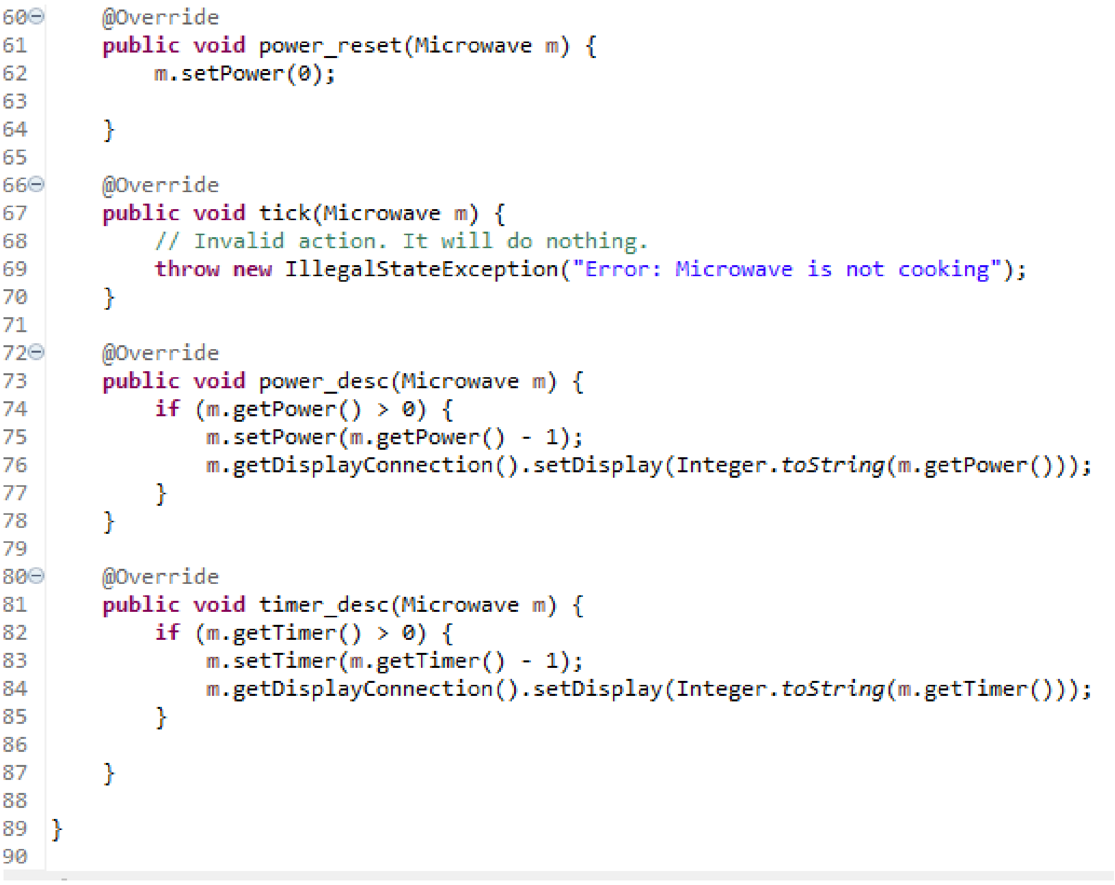


```
    @Override
    public void item_placed(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Microwave is full");
    }

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

    @Override
    public void cooking_start(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: You cannot start cooking with the door opened");
    }

    @Override
    public void cooking_stop(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Microwave is not cooking");
    }

    @Override
    public void timer_reset(Microwave m) {
        m.setTimer(0);

    }

    @Override
    public void power_reset(Microwave m) {
        m.setPower(0);

    }

    @Override
    public void tick(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Microwave is not cooking");
    }

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

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

    }

}
```

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

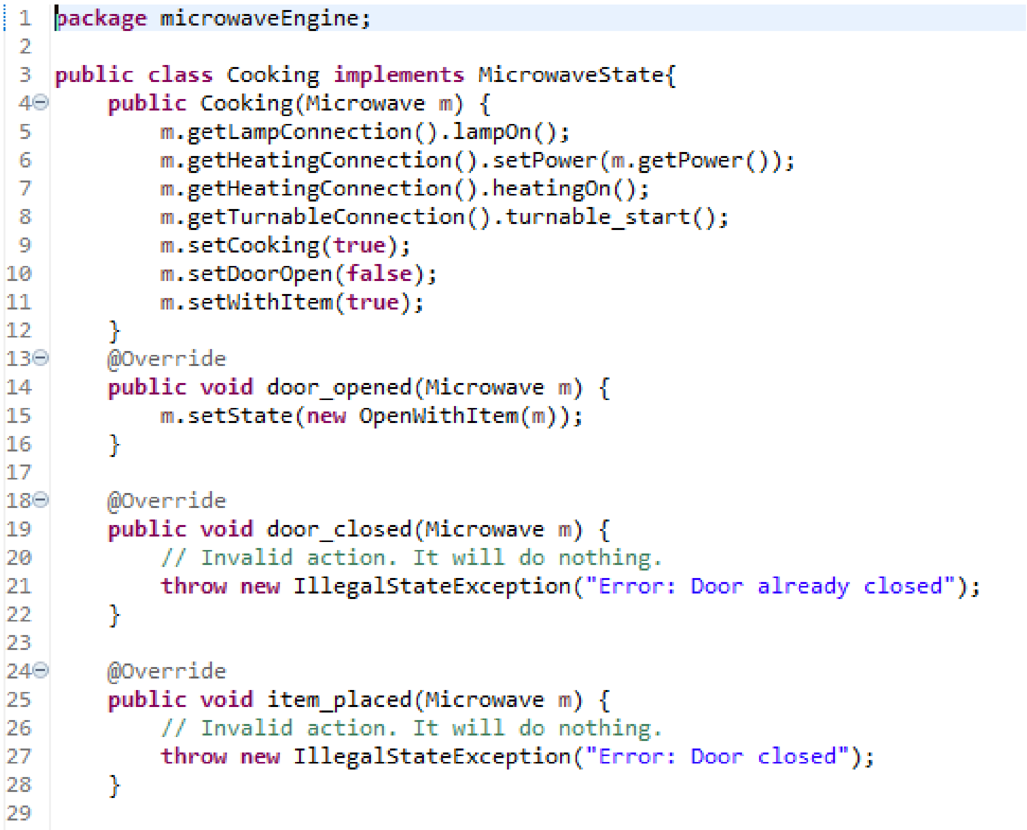


```java
package microwaveEngine;

public class ClosedWithItem implements MicrowaveState {

    public ClosedWithItem(Microwave m) {
        m.getLampConnection().lampOff();
        m.getHeatingConnection().heatingOff();
        m.getTurnableConnection().turnable_stop();
        m.setCooking(false);
        m.setDoorOpen(false);
        m.setWithItem(true);
    }

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

    @Override
    public void door_closed(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Door already closed");
    }

    @Override
    public void item_placed(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Door closed");
    }

    @Override
    public void item_removed(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Door closed");
    }

    @Override
    public void cooking_start(Microwave m) {
        if(m.getTimer() > 0 && m.getPower() > 0) {
            m.setState(new Cooking(m));
        } else if (m.getTimer() > 0) {
            throw new IllegalStateException("Error: Timer is 0");
        } else {
            throw new IllegalStateException("Error: Power is 0");
        }
    }

    @Override
    public void cooking_stop(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Microwave is not cooking");
    }

    @Override
    public void timer_reset(Microwave m) {
        m.setTimer(0);

    }

```

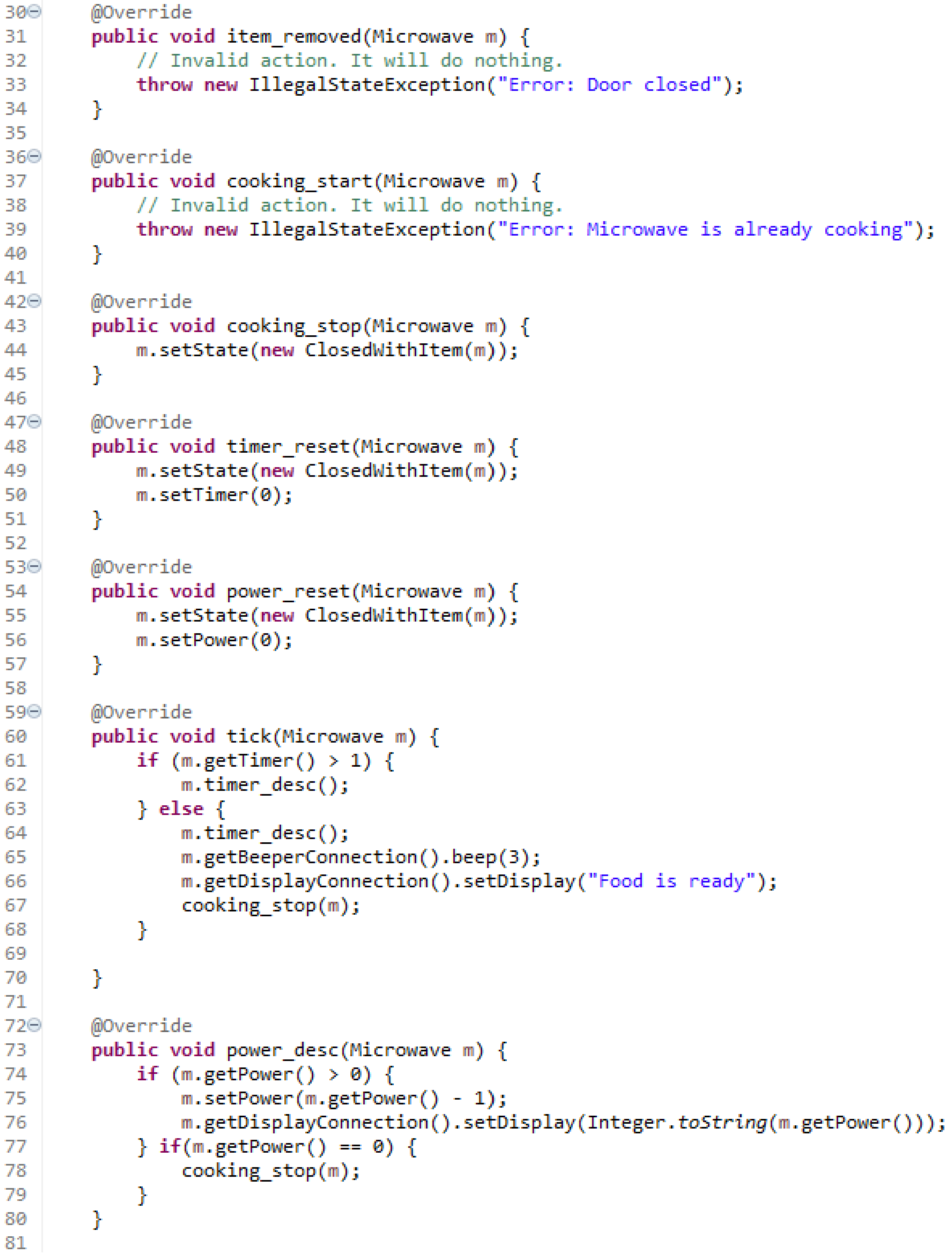


```java
    @Override
    public void power_reset(Microwave m) {
        m.setPower(0);

    }

    @Override
    public void tick(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Microwave is not cooking");
    }

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

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

    }

}

```

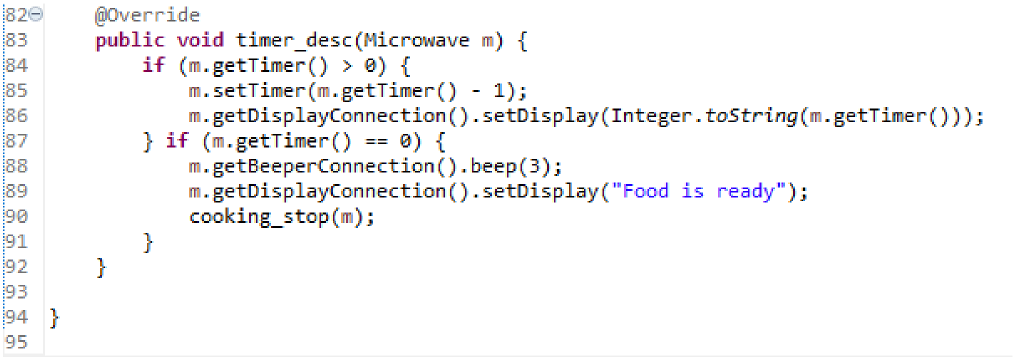
La implementación de la clase que define el estado del microondas cuando está cocinando es:



```java
package microwaveEngine;

public class Cooking implements MicrowaveState{
    public Cooking(Microwave m) {
        m.getLampConnection().lampOn();
        m.getHeatingConnection().setPower(m.getPower());
        m.getHeatingConnection().heatingOn();
        m.getTurnableConnection().turnable_start();
        m.setCooking(true);
        m.setDoorOpen(false);
        m.setWithItem(true);
    }
    @Override
    public void door_opened(Microwave m) {
        m.setState(new OpenWithItem(m));
    }

    @Override
    public void door_closed(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Door already closed");
    }

    @Override
    public void item_placed(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Door closed");
    }

```



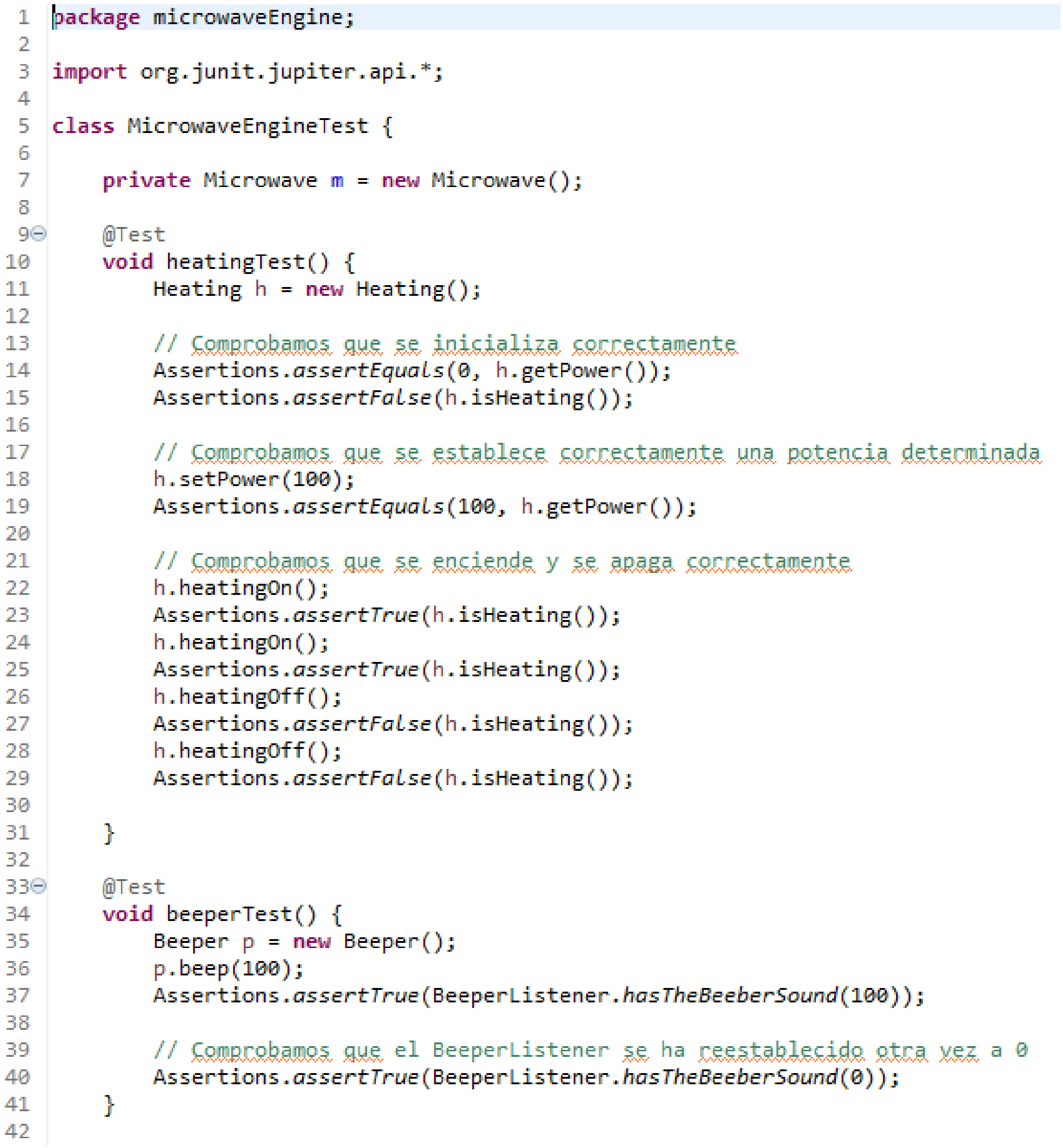
```java
    @Override
    public void item_removed(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Door closed");
    }

    @Override
    public void cooking_start(Microwave m) {
        // Invalid action. It will do nothing.
        throw new IllegalStateException("Error: Microwave is already cooking");
    }

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

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

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

    @Override
    public void tick(Microwave m) {
        if (m.getTimer() > 1) {
            m.timer_desc();
        } else {
            m.timer_desc();
            m.getBeeperConnection().beep(3);
            m.getDisplayConnection().setDisplay("Food is ready");
            cooking_stop(m);
        }

    }

    @Override
    public void power_desc(Microwave m) {
        if (m.getPower() > 0) {
            m.setPower(m.getPower() - 1);
            m.getDisplayConnection().setDisplay(Integer.toString(m.getPower()));
        } if(m.getPower() == 0) {
            cooking_stop(m);
        }
    }

```



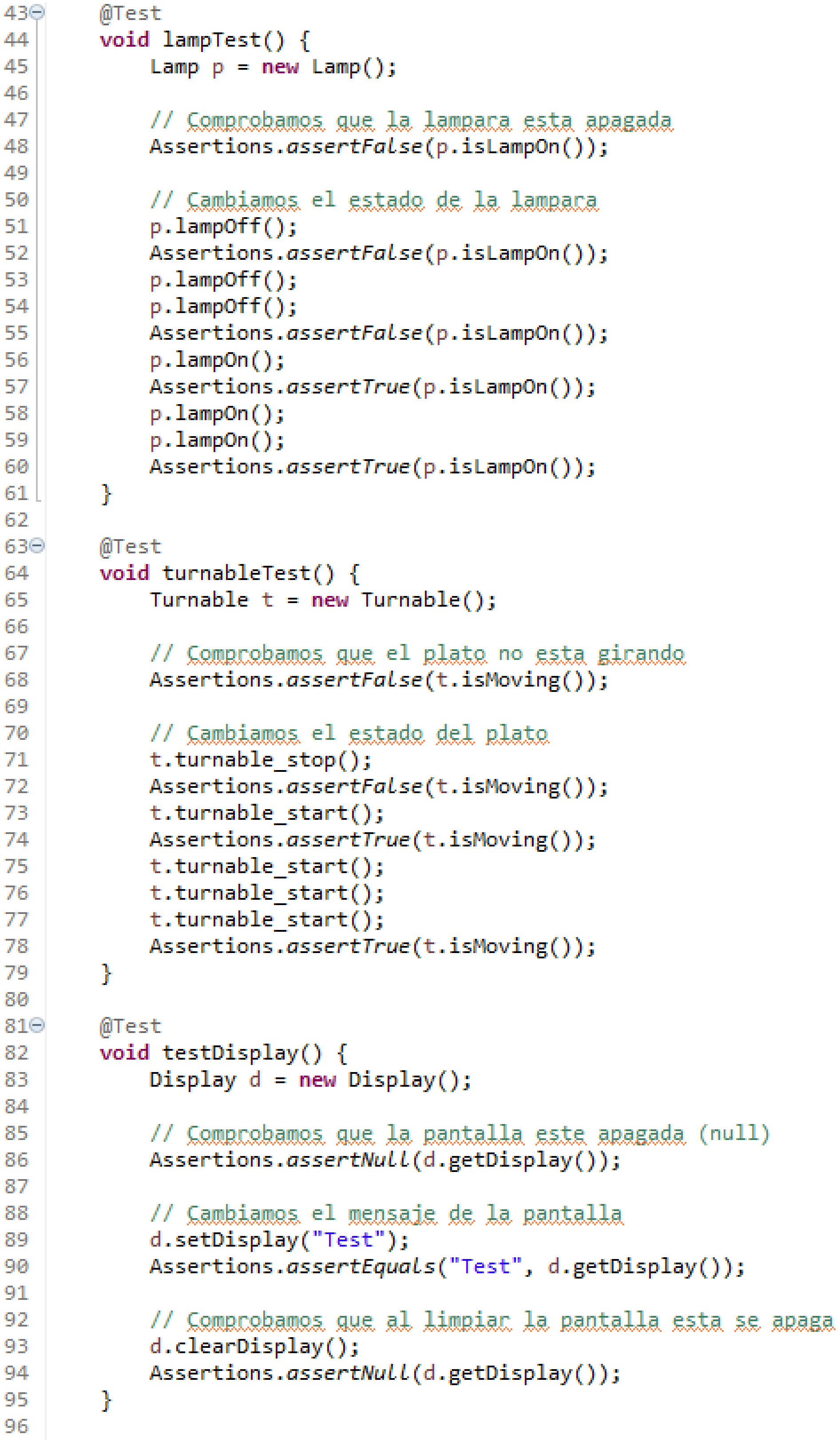
```java
    @Override
    public void timer_desc(Microwave m) {
        if (m.getTimer() > 0) {
            m.setTimer(m.getTimer() - 1);
            m.getDisplayConnection().setDisplay(Integer.toString(m.getTimer()));
        } if (m.getTimer() == 0) {
            m.getBeeperConnection().beep(3);
            m.getDisplayConnection().setDisplay("Food is ready");
            cooking_stop(m);
        }
    }

}

```

Apartado b

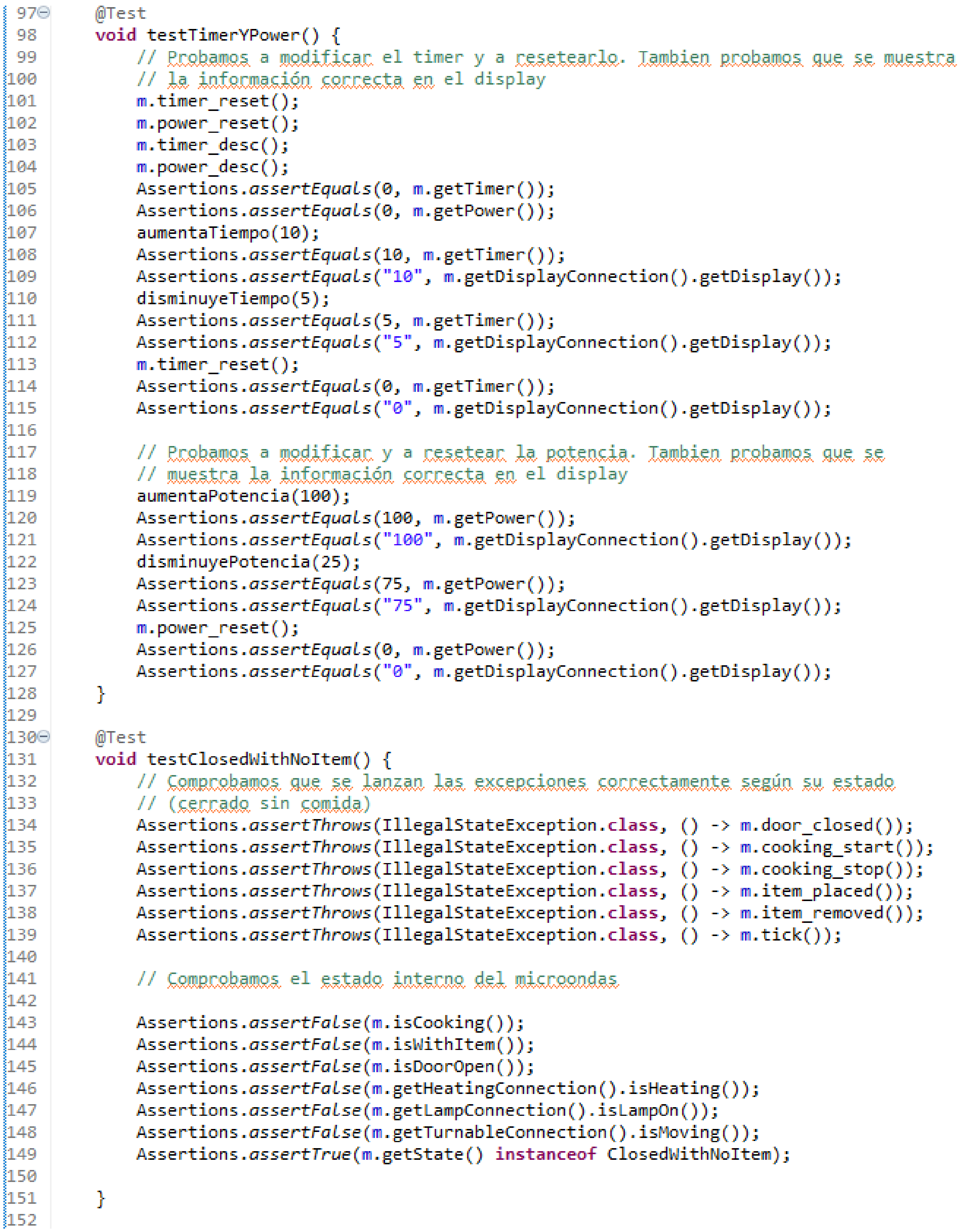
La implementación de los tests de JUnit es:



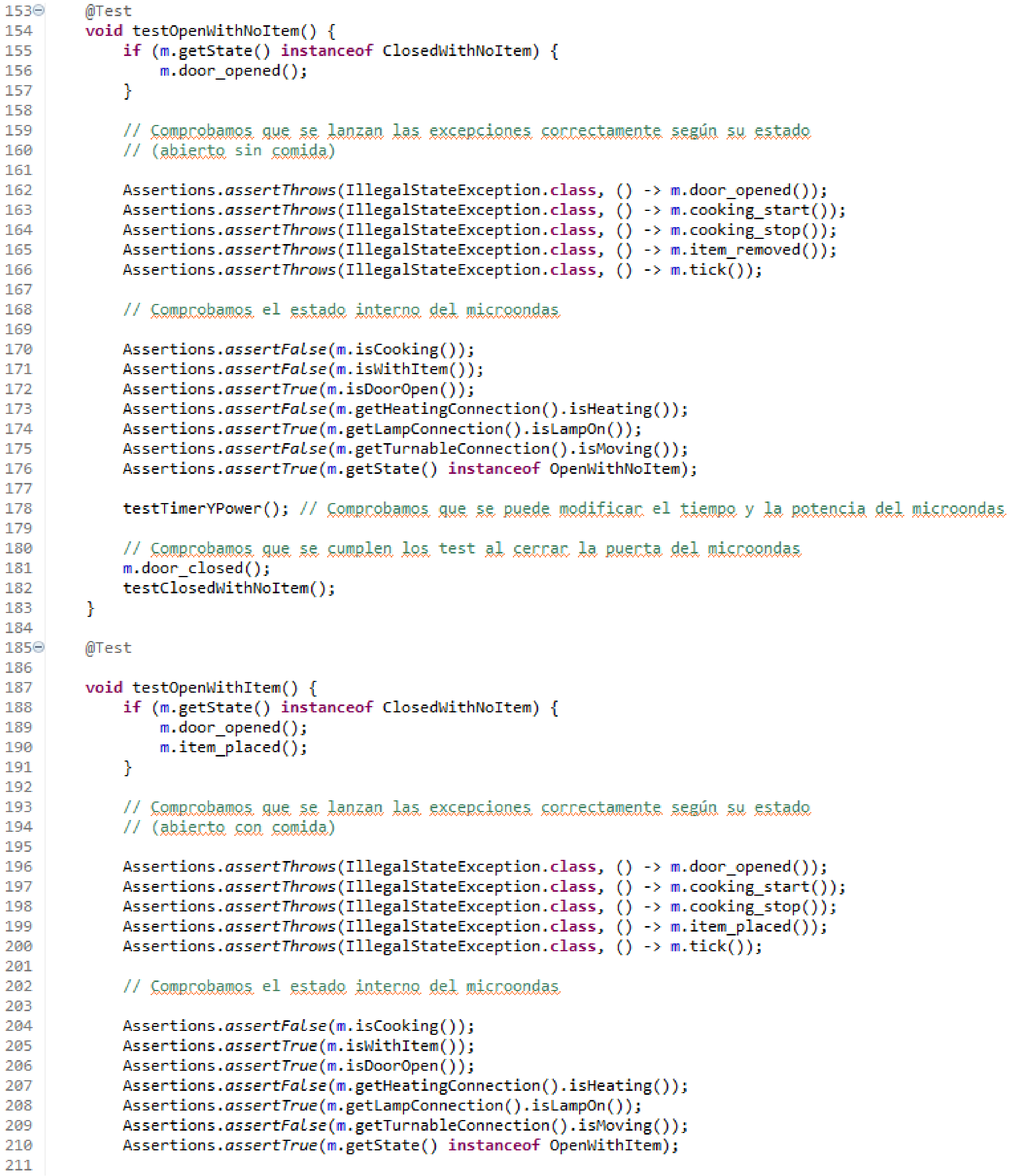
```java
package microwaveEngine;

import org.junit.jupiter.api.*;

class MicrowaveEngineTest {

    private Microwave m = new Microwave();

    @Test
    void heatingTest() {
        Heating h = new Heating();

        // Comprobamos que se inicializa correctamente
        Assertions.assertEquals(0, h.getPower());
        Assertions.assertFalse(h.isHeating());

        // Comprobamos que se establece correctamente una potencia determinada
        h.setPower(100);
        Assertions.assertEquals(100, h.getPower());

        // Comprobamos que se enciende y se apaga correctamente
        h.heatingOn();
        Assertions.assertTrue(h.isHeating());
        h.heatingOn();
        Assertions.assertTrue(h.isHeating());
        h.heatingOff();
        Assertions.assertFalse(h.isHeating());
        h.heatingOff();
        Assertions.assertFalse(h.isHeating());

    }

    @Test
    void beeperTest() {
        Beeper p = new Beeper();
        p.beep(100);
        Assertions.assertTrue(BeeperListener.hasTheBeeberSound(100));

        // Comprobamos que el BeeperListener se ha reestablecido otra vez a 0
        Assertions.assertTrue(BeeperListener.hasTheBeeberSound(0));
    }

```



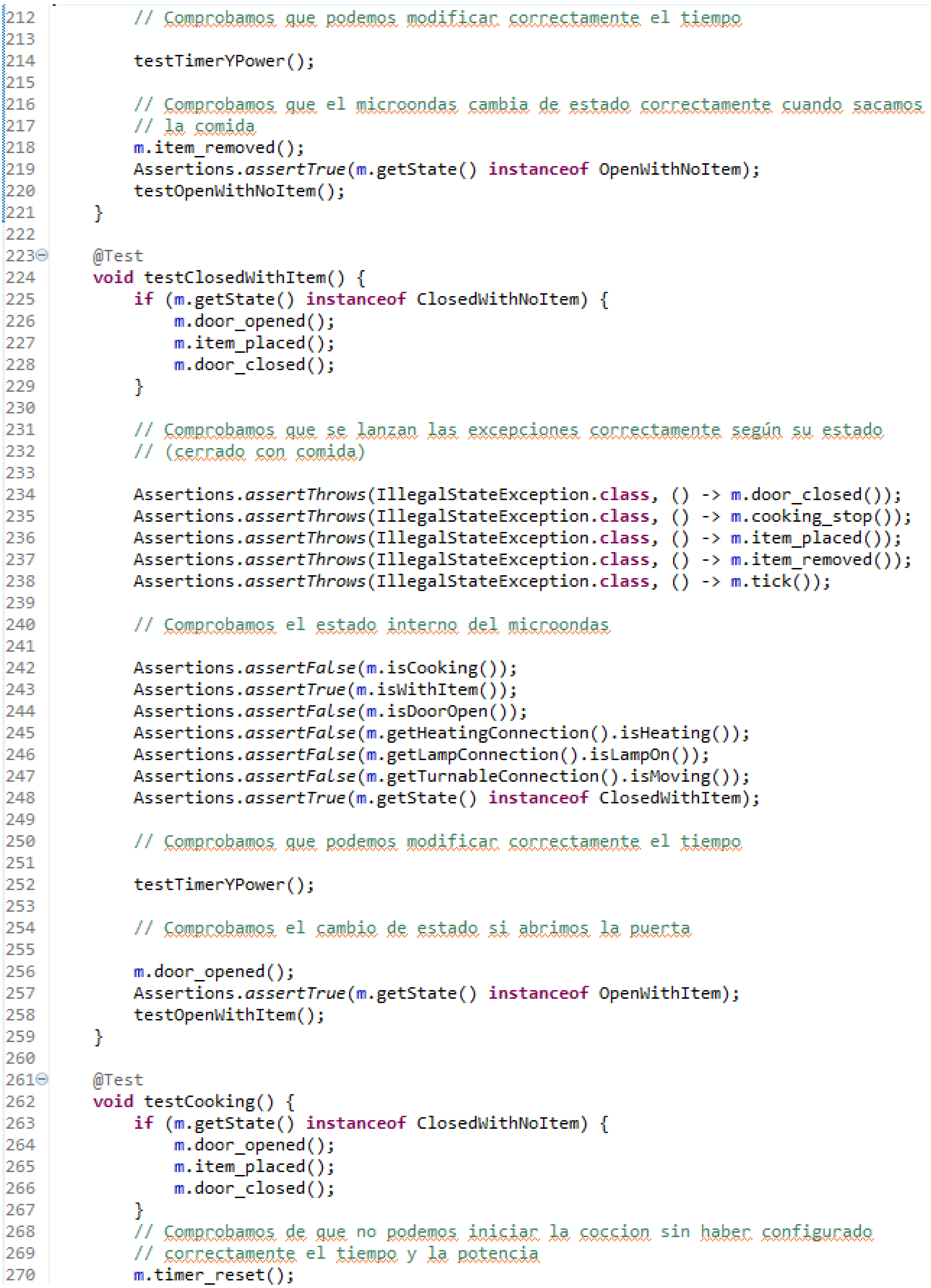
```java
    @Test
    void lampTest() {
        Lamp p = new Lamp();

        // Comprobamos que la lampara esta apagada
        Assertions.assertFalse(p.isLampOn());

        // Cambiamos el estado de la lampara
        p.lampOff();
        Assertions.assertFalse(p.isLampOn());
        p.lampOff();
        p.lampOff();
        Assertions.assertFalse(p.isLampOn());
        p.lampOn();
        Assertions.assertTrue(p.isLampOn());
        p.lampOn();
        p.lampOn();
        Assertions.assertTrue(p.isLampOn());
    }

    @Test
    void turnableTest() {
        Turnable t = new Turnable();

        // Comprobamos que el plato no esta girando
        Assertions.assertFalse(t.isMoving());

        // Cambiamos el estado del plato
        t.turnable_stop();
        Assertions.assertFalse(t.isMoving());
        t.turnable_start();
        Assertions.assertTrue(t.isMoving());
        t.turnable_start();
        t.turnable_start();
        t.turnable_start();
        Assertions.assertTrue(t.isMoving());
    }

    @Test
    void testDisplay() {
        Display d = new Display();

        // Comprobamos que la pantalla este apagada (null)
        Assertions.assertNull(d.getDisplay());

        // Cambiamos el mensaje de la pantalla
        d.setDisplay("Test");
        Assertions.assertEquals("Test", d.getDisplay());

        // Comprobamos que al limpiar la pantalla esta se apaga
        d.clearDisplay();
        Assertions.assertNull(d.getDisplay());
    }

```



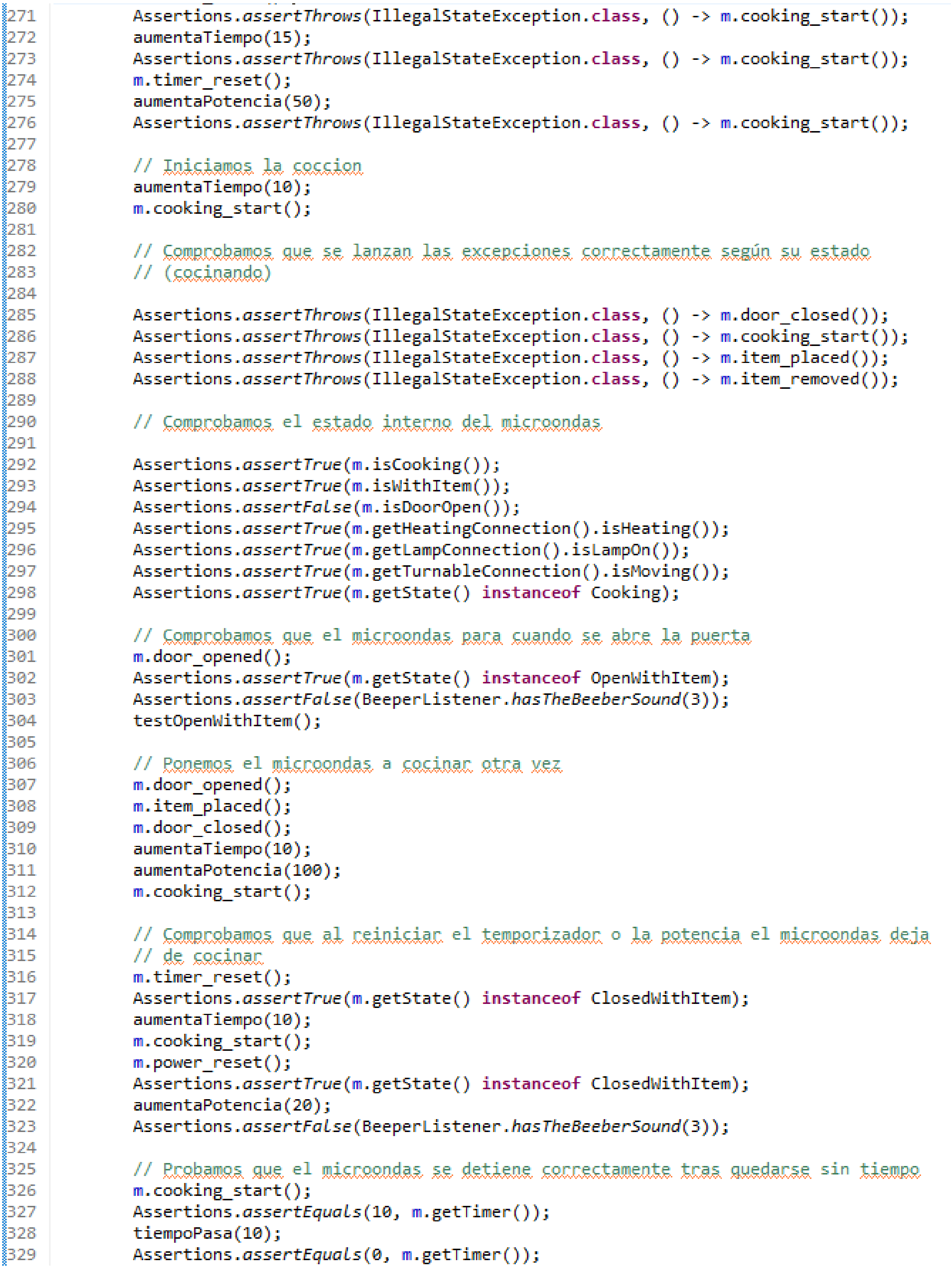
```java
    @Test
    void testTimerYPower() {
        // Probamos a modificar el timer y a resetearlo. Tambien probamos que se muestra
        // la información correcta en el display
        m.timer_reset();
        m.power_reset();
        m.timer_desc();
        m.power_desc();
        Assertions.assertEquals(0, m.getTimer());
        Assertions.assertEquals(0, m.getPower());
        aumentaTiempo(10);
        Assertions.assertEquals(10, m.getTimer());
        Assertions.assertEquals("10", m.getDisplayConnection().getDisplay());
        disminuyeTiempo(5);
        Assertions.assertEquals(5, m.getTimer());
        Assertions.assertEquals("5", m.getDisplayConnection().getDisplay());
        m.timer_reset();
        Assertions.assertEquals(0, m.getTimer());
        Assertions.assertEquals("0", m.getDisplayConnection().getDisplay());

        // Probamos a modificar y a resetear la potencia. Tambien probamos que se
        // muestra la información correcta en el display
        aumentaPotencia(100);
        Assertions.assertEquals(100, m.getPower());
        Assertions.assertEquals("100", m.getDisplayConnection().getDisplay());
        disminuyePotencia(25);
        Assertions.assertEquals(75, m.getPower());
        Assertions.assertEquals("75", m.getDisplayConnection().getDisplay());
        m.power_reset();
        Assertions.assertEquals(0, m.getPower());
        Assertions.assertEquals("0", m.getDisplayConnection().getDisplay());
    }

    @Test
    void testClosedWithNoItem() {
        // Comprobamos que se lanzan las excepciones correctamente según su estado
        // (cerrado sin comida)
        Assertions.assertThrows(IllegalStateException.class, () -> m.door_closed());
        Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
        Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_stop());
        Assertions.assertThrows(IllegalStateException.class, () -> m.item_placed());
        Assertions.assertThrows(IllegalStateException.class, () -> m.item_removed());
        Assertions.assertThrows(IllegalStateException.class, () -> m.tick());

        // Comprobamos el estado interno del microondas

        Assertions.assertFalse(m.isCooking());
        Assertions.assertFalse(m.isWithItem());
        Assertions.assertFalse(m.isDoorOpen());
        Assertions.assertFalse(m.getHeatingConnection().isHeating());
        Assertions.assertFalse(m.getLampConnection().isLampOn());
        Assertions.assertFalse(m.getTurnableConnection().isMoving());
        Assertions.assertTrue(m.getState() instanceof ClosedWithNoItem);

    }
```



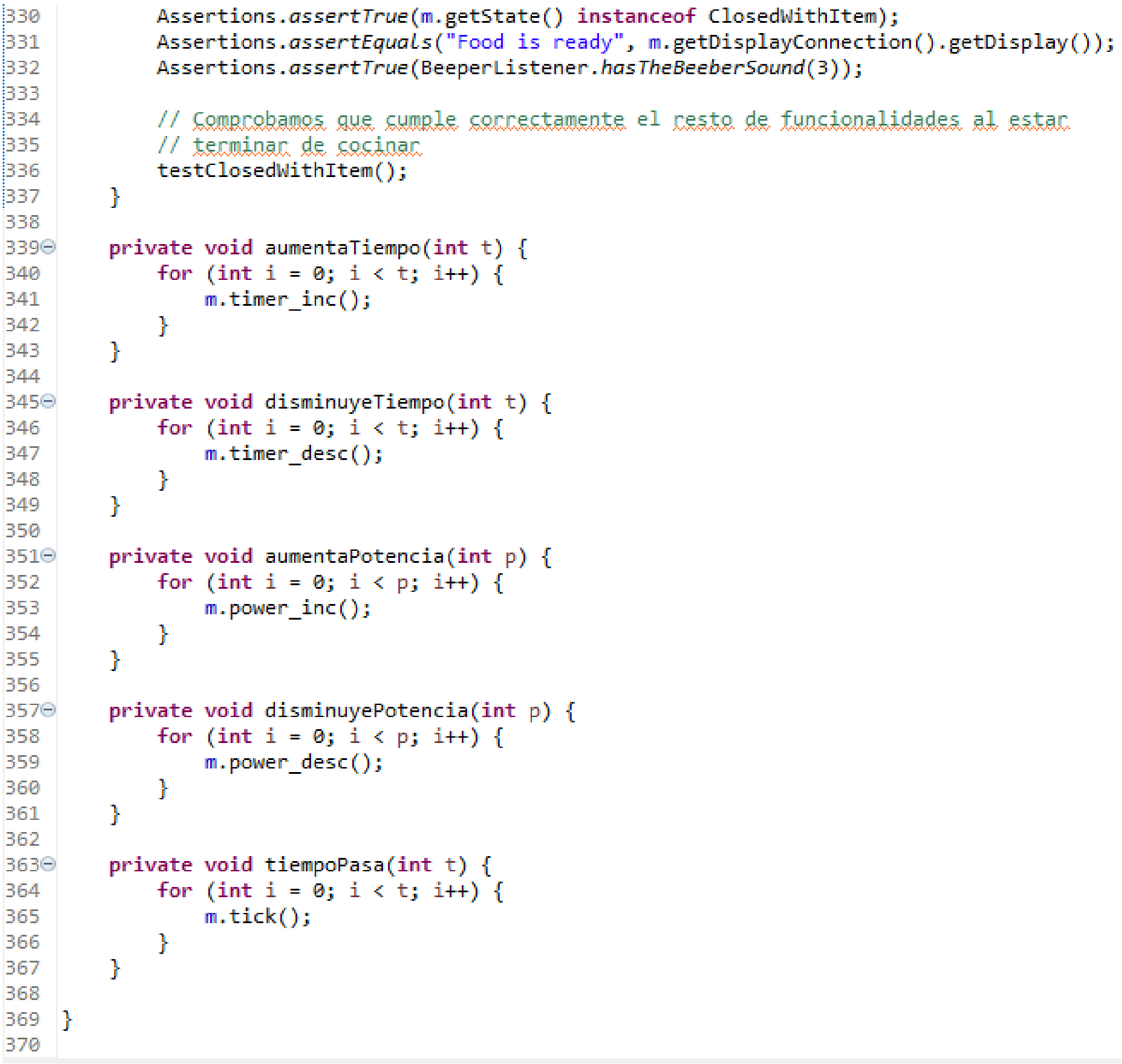
```java
    @Test
    void testOpenWithNoItem() {
        if (m.getState() instanceof ClosedWithNoItem) {
            m.door_opened();
        }

        // Comprobamos que se lanzan las excepciones correctamente según su estado
        // (abierto sin comida)

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

        // Comprobamos el estado interno del microondas

        Assertions.assertFalse(m.isCooking());
        Assertions.assertFalse(m.isWithItem());
        Assertions.assertTrue(m.isDoorOpen());
        Assertions.assertFalse(m.getHeatingConnection().isHeating());
        Assertions.assertTrue(m.getLampConnection().isLampOn());
        Assertions.assertFalse(m.getTurnableConnection().isMoving());
        Assertions.assertTrue(m.getState() instanceof OpenWithNoItem);

        testTimerYPower(); // Comprobamos que se puede modificar el tiempo y la potencia del microondas

        // Comprobamos que se cumplen los test al cerrar la puerta del microondas
        m.door_closed();
        testClosedWithNoItem();
    }

    @Test

    void testOpenWithItem() {
        if (m.getState() instanceof ClosedWithNoItem) {
            m.door_opened();
            m.item_placed();
        }

        // Comprobamos que se lanzan las excepciones correctamente según su estado
        // (abierto con comida)

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

        // Comprobamos el estado interno del microondas

        Assertions.assertFalse(m.isCooking());
        Assertions.assertTrue(m.isWithItem());
        Assertions.assertTrue(m.isDoorOpen());
        Assertions.assertFalse(m.getHeatingConnection().isHeating());
        Assertions.assertTrue(m.getLampConnection().isLampOn());
        Assertions.assertFalse(m.getTurnableConnection().isMoving());
        Assertions.assertTrue(m.getState() instanceof OpenWithItem);

```



```
        // Comprobamos que podemos modificar correctamente el tiempo

        testTimerYPower();

        // Comprobamos que el microondas cambia de estado correctamente cuando sacamos
        // la comida
        m.item_removed();
        Assertions.assertTrue(m.getState() instanceof OpenWithNoItem);
        testOpenWithNoItem();
    }

    @Test
    void testClosedWithItem() {
        if (m.getState() instanceof ClosedWithNoItem) {
            m.door_opened();
            m.item_placed();
            m.door_closed();
        }

        // Comprobamos que se lanzan las excepciones correctamente según su estado
        // (cerrado con comida)

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

        // Comprobamos el estado interno del microondas

        Assertions.assertFalse(m.isCooking());
        Assertions.assertTrue(m.isWithItem());
        Assertions.assertFalse(m.isDoorOpen());
        Assertions.assertFalse(m.getHeatingConnection().isHeating());
        Assertions.assertFalse(m.getLampConnection().isLampOn());
        Assertions.assertFalse(m.getTurnableConnection().isMoving());
        Assertions.assertTrue(m.getState() instanceof ClosedWithItem);

        // Comprobamos que podemos modificar correctamente el tiempo

        testTimerYPower();

        // Comprobamos el cambio de estado si abrimos la puerta

        m.door_opened();
        Assertions.assertTrue(m.getState() instanceof OpenWithItem);
        testOpenWithItem();
    }

    @Test
    void testCooking() {
        if (m.getState() instanceof ClosedWithNoItem) {
            m.door_opened();
            m.item_placed();
            m.door_closed();
        }
        // Comprobamos de que no podemos iniciar la coccion sin haber configurado
        // correctamente el tiempo y la potencia
        m.timer_reset();
```

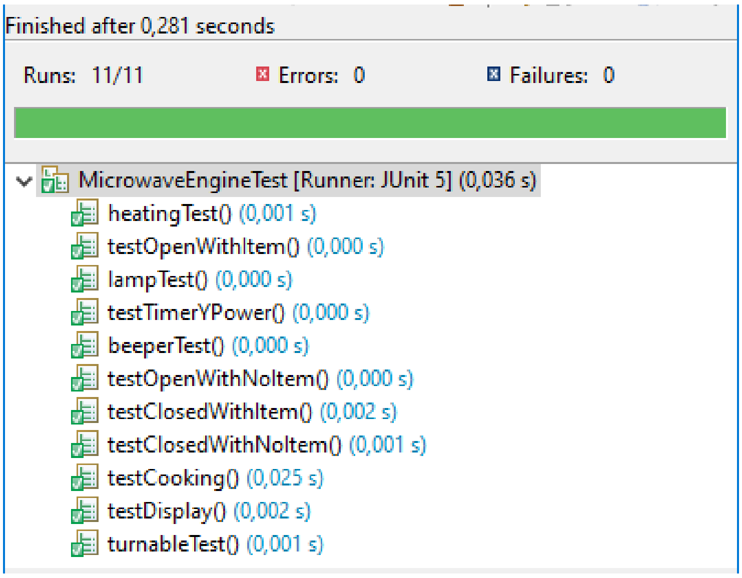


```
        Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
        aumentaTiempo(15);
        Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
        m.timer_reset();
        aumentaPotencia(50);
        Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());

        // Iniciamos la coccion
        aumentaTiempo(10);
        m.cooking_start();

        // Comprobamos que se lanzan las excepciones correctamente según su estado
        // (cocinando)

        Assertions.assertThrows(IllegalStateException.class, () -> m.door_closed());
        Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
        Assertions.assertThrows(IllegalStateException.class, () -> m.item_placed());
        Assertions.assertThrows(IllegalStateException.class, () -> m.item_removed());

        // Comprobamos el estado interno del microondas

        Assertions.assertTrue(m.isCooking());
        Assertions.assertTrue(m.isWithItem());
        Assertions.assertFalse(m.isDoorOpen());
        Assertions.assertTrue(m.getHeatingConnection().isHeating());
        Assertions.assertTrue(m.getLampConnection().isLampOn());
        Assertions.assertTrue(m.getTurnableConnection().isMoving());
        Assertions.assertTrue(m.getState() instanceof Cooking);

        // Comprobamos que el microondas para cuando se abre la puerta
        m.door_opened();
        Assertions.assertTrue(m.getState() instanceof OpenWithItem);
        Assertions.assertFalse(BeeperListener.hasTheBeeberSound(3));
        testOpenWithItem();

        // Ponemos el microondas a cocinar otra vez
        m.door_opened();
        m.item_placed();
        m.door_closed();
        aumentaTiempo(10);
        aumentaPotencia(100);
        m.cooking_start();

        // Comprobamos que al reiniciar el temporizador o la potencia el microondas deja
        // de cocinar
        m.timer_reset();
        Assertions.assertTrue(m.getState() instanceof ClosedWithItem);
        aumentaTiempo(10);
        m.cooking_start();
        m.power_reset();
        Assertions.assertTrue(m.getState() instanceof ClosedWithItem);
        aumentaPotencia(20);
        Assertions.assertFalse(BeeperListener.hasTheBeeberSound(3));

        // Probamos que el microondas se detiene correctamente tras quedarse sin tiempo
        m.cooking_start();
        Assertions.assertEquals(10, m.getTimer());
        tiempoPasa(10);
        Assertions.assertEquals(0, m.getTimer());
```



```java
        Assertions.assertTrue(m.getState() instanceof ClosedWithItem);
        Assertions.assertEquals("Food is ready", m.getDisplayConnection().getDisplay());
        Assertions.assertTrue(BeeperListener.hasTheBeeberSound(3));

        // Comprobamos que cumple correctamente el resto de funcionalidades al estar
        // terminar de cocinar
        testClosedWithItem();
    }

    private void aumentaTiempo(int t) {
        for (int i = 0; i < t; i++) {
            m.timer_inc();
        }
    }

    private void disminuyeTiempo(int t) {
        for (int i = 0; i < t; i++) {
            m.timer_desc();
        }
    }

    private void aumentaPotencia(int p) {
        for (int i = 0; i < p; i++) {
            m.power_inc();
        }
    }

    private void disminuyePotencia(int p) {
        for (int i = 0; i < p; i++) {
            m.power_desc();
        }
    }

    private void tiempoPasa(int t) {
        for (int i = 0; i < t; i++) {
            m.tick();
        }
    }

}
```

Resultado del test:



```
Finished after 0,281 seconds

Runs:  11/11          ☒ Errors:  0          ☒ Failures:  0

MicrowaveEngineTest [Runner: JUnit 5] (0,036 s)
    heatingTest() (0,001 s)
    testOpenWithItem() (0,000 s)
    lampTest() (0,000 s)
    testTimerYPower() (0,000 s)
    beeperTest() (0,000 s)
    testOpenWithNoItem() (0,000 s)
    testClosedWithItem() (0,002 s)
    testClosedWithNoItem() (0,001 s)
    testCooking() (0,025 s)
    testDisplay() (0,002 s)
    turnableTest() (0,001 s)
```

Apartado c

Se han implementado cuatro características (features) diferentes.

ClosedEmptyMicrowave.feature

```
Feature: Using a closed microwave with no item on it

  Scenario: Open a closed microwave
    Given a closed empty microwave
    When I open the door
    Then the door opens
    And the light turns on
    And the plate is not turning
    And the microwave is not heating

  Scenario: Trying to reset time in a microwave
    Given a closed empty microwave
    When I press reset timer button
    Then the timer must be set to zero

  Scenario: Trying to reset power in a microwave
    Given a closed empty microwave
    When I press reset power button
    Then the power must be set to zero

  Scenario Outline: Trying to set time
    Given a closed empty microwave
    When I set time to <a> seconds
    Then the microwave must display "<b>"

    Examples:
      | a   | b   |
      |  -1 |   0 |
      |   0 |   0 |
      |   1 |   1 |
      |  15 |  15 |
      | 100 | 100 |
      |  40 |  40 |

  Scenario Outline: Trying to set power
    Given a closed empty microwave
    When I set the power to <a>
    Then the microwave must display "<b>"

    Examples:
      | a   | b   |
      |  -1 |   0 |
      |   0 |   0 |
      |  20 |  20 |
      | 200 | 200 |
      | 201 | 201 |
      |   2 |   2 |
  Scenario: Trying to cook in a closed microwave with no item
    Given a closed empty microwave
    When I set time to 20 seconds
    And I set the power to 100
    And I press start cooking button
    Then the microwave must not start cooking
```

OpenedMicrowave.feature

```
Feature: Using an opened microwave

  Scenario: Open a closed microwave
    Given a closed empty microwave
    When I open the door
    Then the door opens
    And the light turns on
    And the plate is not turning
    And the microwave is not heating

  Scenario: Placing food in a microwave
    Given an opened empty microwave
    When I insert food
    Then the food is placed

  Scenario: Removing food from a microwave
    Given an opened full microwave
    When I retrieve the food
    Then the microwave is empty

  Scenario: Trying to reset time in a microwave
    Given an opened empty microwave
    When I press reset timer button
    Then the timer must be set to zero

  Scenario: Trying to reset power in a microwave
    Given an opened empty microwave
    When I press reset power button
    Then the power must be set to zero

  Scenario: Trying to cook in a opened microwave
    Given an opened empty microwave
    When I set time to 20 seconds
    And I set the power to 100
    And I press start cooking button
    Then the microwave must not start cooking

  Scenario Outline: Trying to set time
    Given an opened empty microwave
    When I set time to <a> seconds
    Then the microwave must display "<b>"
    Examples:
      | a   | b   |
      |  -1 |   0 |
      |   0 |   0 |
      |   1 |   1 |
      |  15 |  15 |
      | 100 | 100 |
      |  40 |  40 |

  Scenario Outline: Trying to set power
    Given an opened empty microwave
    When I set the power to <a>
    Then the microwave must display "<b>"

    Examples:
      | a   | b   |
      |  -1 |   0 |
      |   0 |   0 |
      |  20 |  20 |
      | 200 | 200 |
      | 201 | 201 |
      |   2 |   2 |
```

ClosedFullMicrowave.feature

```
Feature: Using a closed Microwave with an item inside

  Scenario: Close the door of a full door-opened microwave
    Given an opened empty microwave
    And I insert food
    When I close the door
    Then the door is closed
    And the light are not on
    And the plate is not turning
    And the microwave is not heating

  Scenario: Trying to reset time in a microwave
    Given a closed empty microwave
    When I press reset timer button
    Then the timer must be set to zero

  Scenario: Trying to reset power in a microwave
    Given a closed empty microwave
    When I press reset power button
    Then the power must be set to zero

  Scenario Outline: Trying to set time
    Given a closed empty microwave
    When I set time to <a> seconds
    Then the microwave must display "<b>"
    Examples:
      | a   | b   |
      |  -1 |   0 |
      |   0 |   0 |
      |   1 |   1 |
      |  15 |  15 |
      | 100 | 100 |
      |  40 |  40 |

  Scenario Outline: Trying to set power
    Given a closed empty microwave
    When I set the power to <a>
    Then the microwave must display "<b>"

    Examples:
      | a   | b   |
      |  -1 |   0 |
      |   0 |   0 |
      |  20 |  20 |
      | 200 | 200 |
      | 201 | 201 |
      |   2 |   2 |
```

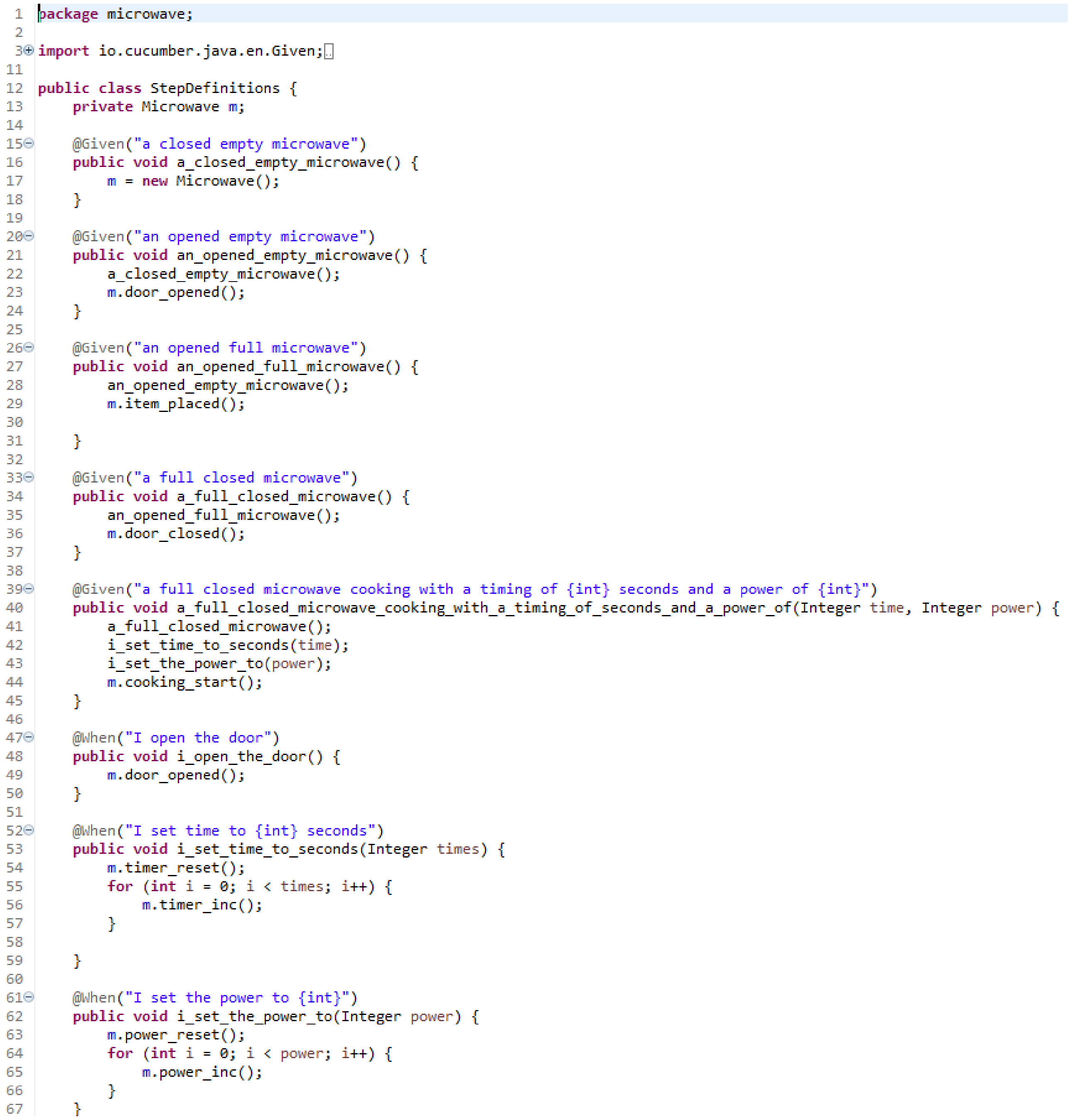
CookingWithMicrowave.feature

```
Feature: Cooking with a closed full microwave

  Scenario: Start cooking with a microwave
    Given a full closed microwave
    When I set time to 20 seconds
    And I set the power to 100
    And I press start cooking button
    Then the microwave must start cooking
    And the light turns on
    And the plate is turning
    And the microwave is heating

  Scenario Outline: Cooking time finishes correctly
    Given a full closed microwave
    When I set time to <a> seconds
    And I set the power to 100
    And I press start cooking button
    And it passes <b> seconds
    Then the microwave is not cooking
    And the light are not on
    And the plate is not turning
    And the microwave is not heating
    And the beeper sounds 3 times
    And the microwave must display "Food is ready"
    Examples:
      | a   | b   |
      |   5 |   5 |
      |   7 |   7 |
      |  10 |  10 |
      |  15 |  15 |
      |  40 |  40 |
      | 120 | 120 |

  Scenario: Increase time during cooking process
    Given a full closed microwave cooking with a timing of 10 seconds and a power of 100
    When I press the increase timer button
    Then the microwave must display "11"
    And the cooking time must be 11

  Scenario: Decrease time during cooking process
    Given a full closed microwave cooking with a timing of 10 seconds and a power of 100
    When I press the decrease timer button
    Then the microwave must display "9"
    And the cooking time must be 9

  Scenario: Increase power during cooking process
    Given a full closed microwave cooking with a timing of 10 seconds and a power of 100
    When I press the increase power button
    Then the microwave must display "101"

  Scenario: Decrease power during cooking process
    Given a full closed microwave cooking with a timing of 10 seconds and a power of 100
    When I press the decrease power button
    Then the microwave must display "99"

  Scenario: Abort cooking process by opening the door
    Given a full closed microwave cooking with a timing of 5 seconds and a power of 50
    When I open the door
    Then the door opens
    And the microwave is not cooking
    And the light turns on
    And the plate is not turning
    And the microwave is not heating

  Scenario: Start again cooking after opening the microwave while it was cooking
    Given a full closed microwave cooking with a timing of 3 seconds and a power of 50
    When I open the door
    And I close the door
    And I press the increase timer button
    Then the microwave must display "4"
```

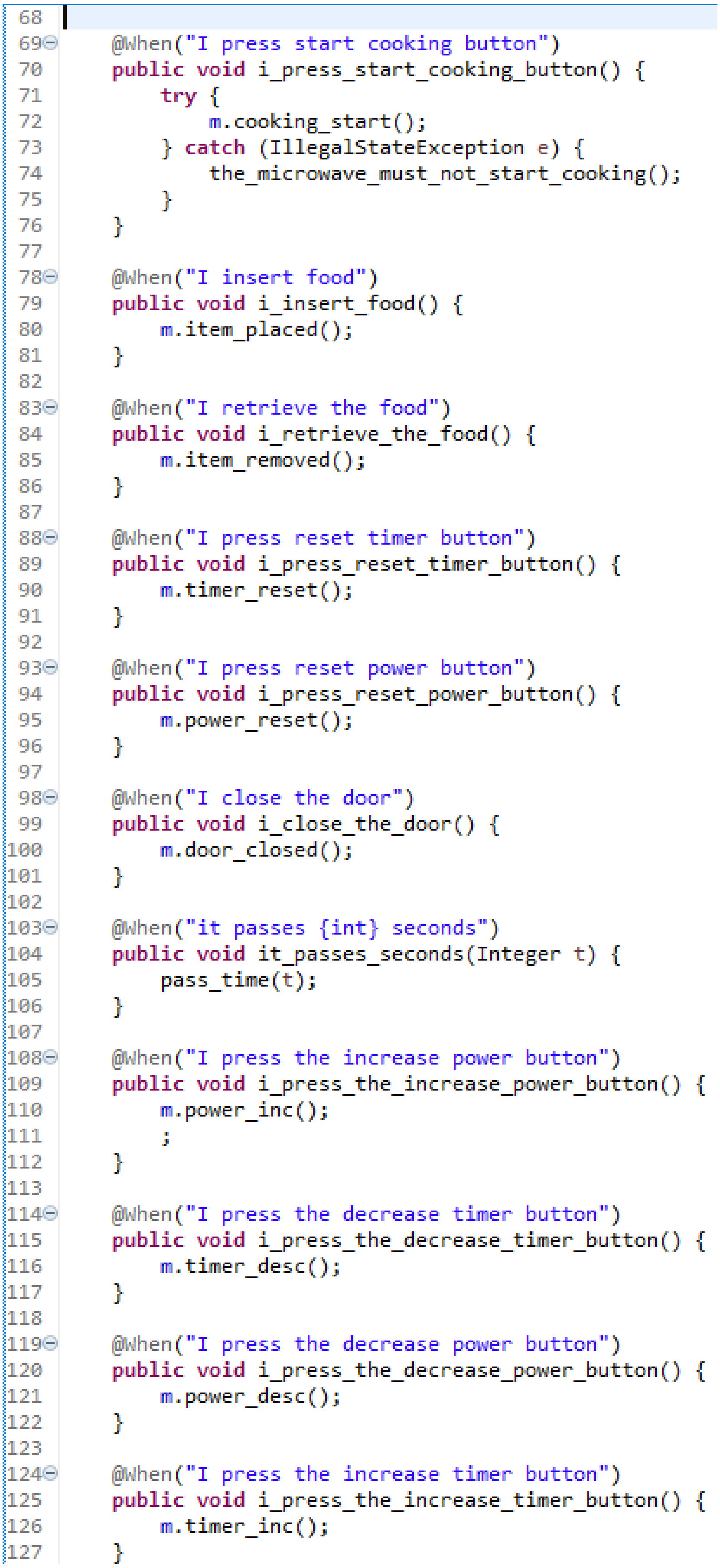
```
Scenario: Abort cooking process by resetting the timer
  Given a full closed microwave cooking with a timing of 1 seconds and a power of 2
  When I press reset timer button
  Then the microwave is not cooking
  And the light are not on
  And the plate is not turning
  And the microwave is not heating
  And the microwave must display "0"

Scenario: Abort cooking process by resetting the power
  Given a full closed microwave cooking with a timing of 11 seconds and a power of 600
  When I press reset power button
  Then the microwave is not cooking
  And the light are not on
  And the plate is not turning
  And the microwave is not heating
  And the microwave must display "0"

Scenario: Abort cooking process by pressing descrease time button when the time is 1
  Given a full closed microwave cooking with a timing of 1 seconds and a power of 10
  When I press the decrease timer button
  Then the microwave is not cooking
  And the light are not on
  And the plate is not turning
  And the microwave is not heating
  And the microwave must display "Food is ready"
  And the beeper sounds 3 times

Scenario: Abort cooking process by pressing descrease power button when the power is 1
  Given a full closed microwave cooking with a timing of 10 seconds and a power of 1
  When I press the decrease power button
  Then the microwave is not cooking
  And the light are not on
  And the plate is not turning
  And the microwave is not heating
  And the microwave must display "0"
  And the beeper sounds 0 times
```

StepDefinitions

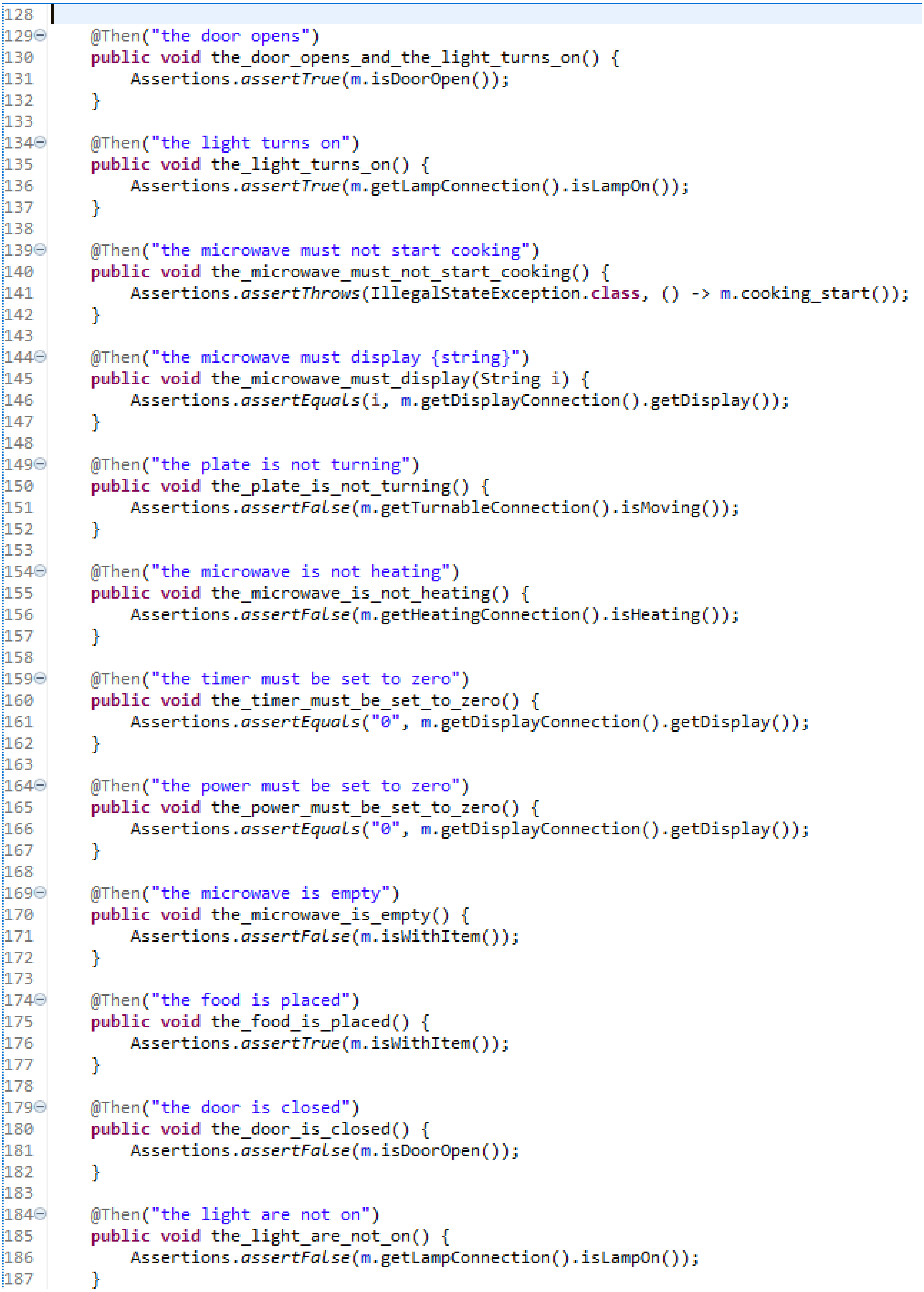
En la clase StepDefinitions se ha definido cada uno de los métodos que se invocarán con los escenarios.



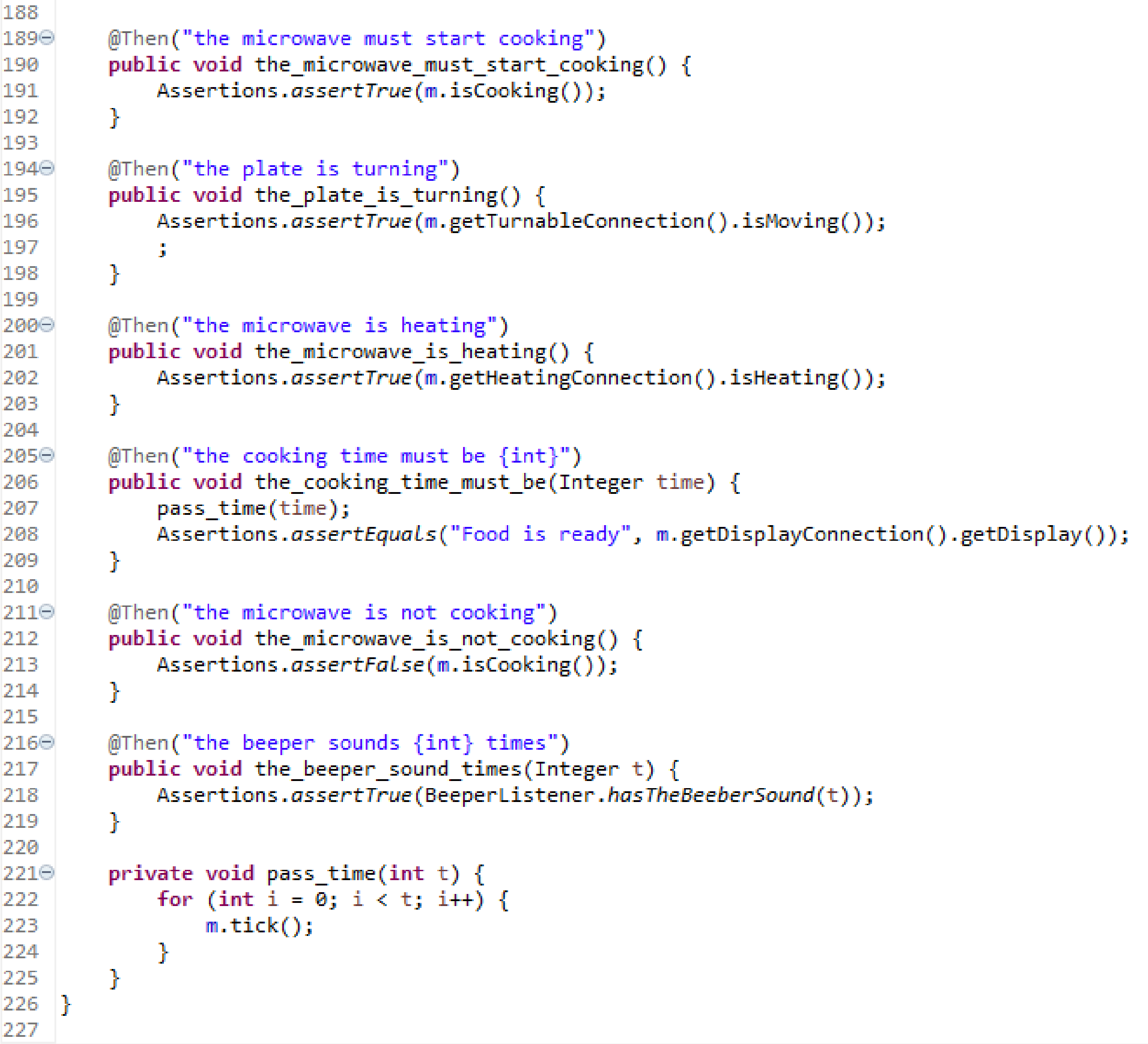
```java
package microwave;

import io.cucumber.java.en.Given;

public class StepDefinitions {
    private Microwave m;

    @Given("a closed empty microwave")
    public void a_closed_empty_microwave() {
        m = new Microwave();
    }

    @Given("an opened empty microwave")
    public void an_opened_empty_microwave() {
        a_closed_empty_microwave();
        m.door_opened();
    }

    @Given("an opened full microwave")
    public void an_opened_full_microwave() {
        an_opened_empty_microwave();
        m.item_placed();

    }

    @Given("a full closed microwave")
    public void a_full_closed_microwave() {
        an_opened_full_microwave();
        m.door_closed();
    }

    @Given("a full closed microwave cooking with a timing of {int} seconds and a power of {int}")
    public void a_full_closed_microwave_cooking_with_a_timing_of_seconds_and_a_power_of(Integer time, Integer power) {
        a_full_closed_microwave();
        i_set_time_to_seconds(time);
        i_set_the_power_to(power);
        m.cooking_start();
    }

    @When("I open the door")
    public void i_open_the_door() {
        m.door_opened();
    }

    @When("I set time to {int} seconds")
    public void i_set_time_to_seconds(Integer times) {
        m.timer_reset();
        for (int i = 0; i < times; i++) {
            m.timer_inc();
        }

    }

    @When("I set the power to {int}")
    public void i_set_the_power_to(Integer power) {
        m.power_reset();
        for (int i = 0; i < power; i++) {
            m.power_inc();
        }
    }
```



```java

    @When("I press start cooking button")
    public void i_press_start_cooking_button() {
        try {
            m.cooking_start();
        } catch (IllegalStateException e) {
            the_microwave_must_not_start_cooking();
        }
    }

    @When("I insert food")
    public void i_insert_food() {
        m.item_placed();
    }

    @When("I retrieve the food")
    public void i_retrieve_the_food() {
        m.item_removed();
    }

    @When("I press reset timer button")
    public void i_press_reset_timer_button() {
        m.timer_reset();
    }

    @When("I press reset power button")
    public void i_press_reset_power_button() {
        m.power_reset();
    }

    @When("I close the door")
    public void i_close_the_door() {
        m.door_closed();
    }

    @When("it passes {int} seconds")
    public void it_passes_seconds(Integer t) {
        pass_time(t);
    }

    @When("I press the increase power button")
    public void i_press_the_increase_power_button() {
        m.power_inc();
        ;
    }

    @When("I press the decrease timer button")
    public void i_press_the_decrease_timer_button() {
        m.timer_desc();
    }

    @When("I press the decrease power button")
    public void i_press_the_decrease_power_button() {
        m.power_desc();
    }

    @When("I press the increase timer button")
    public void i_press_the_increase_timer_button() {
        m.timer_inc();
    }
```

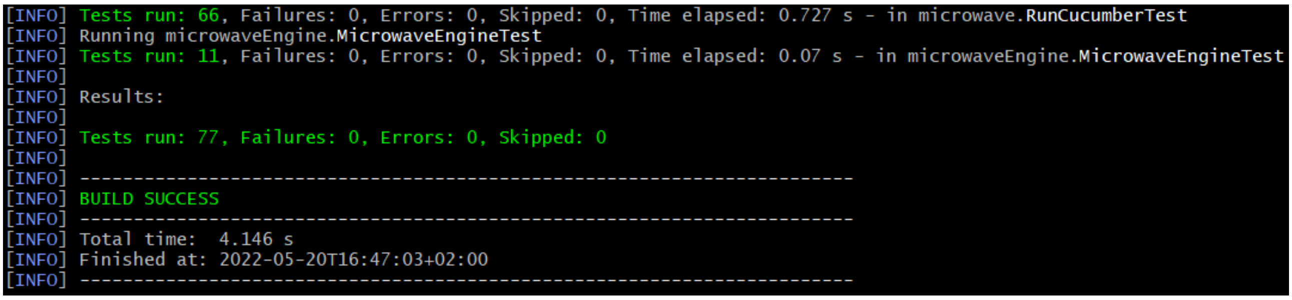


```
    @Then("the door opens")
    public void the_door_opens_and_the_light_turns_on() {
        Assertions.assertTrue(m.isDoorOpen());
    }

    @Then("the light turns on")
    public void the_light_turns_on() {
        Assertions.assertTrue(m.getLampConnection().isLampOn());
    }

    @Then("the microwave must not start cooking")
    public void the_microwave_must_not_start_cooking() {
        Assertions.assertThrows(IllegalStateException.class, () -> m.cooking_start());
    }

    @Then("the microwave must display {string}")
    public void the_microwave_must_display(String i) {
        Assertions.assertEquals(i, m.getDisplayConnection().getDisplay());
    }

    @Then("the plate is not turning")
    public void the_plate_is_not_turning() {
        Assertions.assertFalse(m.getTurnableConnection().isMoving());
    }

    @Then("the microwave is not heating")
    public void the_microwave_is_not_heating() {
        Assertions.assertFalse(m.getHeatingConnection().isHeating());
    }

    @Then("the timer must be set to zero")
    public void the_timer_must_be_set_to_zero() {
        Assertions.assertEquals("0", m.getDisplayConnection().getDisplay());
    }

    @Then("the power must be set to zero")
    public void the_power_must_be_set_to_zero() {
        Assertions.assertEquals("0", m.getDisplayConnection().getDisplay());
    }

    @Then("the microwave is empty")
    public void the_microwave_is_empty() {
        Assertions.assertFalse(m.isWithItem());
    }

    @Then("the food is placed")
    public void the_food_is_placed() {
        Assertions.assertTrue(m.isWithItem());
    }

    @Then("the door is closed")
    public void the_door_is_closed() {
        Assertions.assertFalse(m.isDoorOpen());
    }

    @Then("the light are not on")
    public void the_light_are_not_on() {
        Assertions.assertFalse(m.getLampConnection().isLampOn());
    }
```



```java

    @Then("the microwave must start cooking")
    public void the_microwave_must_start_cooking() {
        Assertions.assertTrue(m.isCooking());
    }

    @Then("the plate is turning")
    public void the_plate_is_turning() {
        Assertions.assertTrue(m.getTurnableConnection().isMoving());
        ;
    }

    @Then("the microwave is heating")
    public void the_microwave_is_heating() {
        Assertions.assertTrue(m.getHeatingConnection().isHeating());
    }

    @Then("the cooking time must be {int}")
    public void the_cooking_time_must_be(Integer time) {
        pass_time(time);
        Assertions.assertEquals("Food is ready", m.getDisplayConnection().getDisplay());
    }

    @Then("the microwave is not cooking")
    public void the_microwave_is_not_cooking() {
        Assertions.assertFalse(m.isCooking());
    }

    @Then("the beeper sounds {int} times")
    public void the_beeper_sound_times(Integer t) {
        Assertions.assertTrue(BeeperListener.hasTheBeeberSound(t));
    }

    private void pass_time(int t) {
        for (int i = 0; i < t; i++) {
            m.tick();
        }
    }
}

```

Resultados de ejecutar dichos tests



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

Enlace a GitHub

https://github.com/FlorinUMA/Microwave