

Universidad de Málaga

Ingeniería de la Salud

Proyecto Microondas

Repositorio

https://github.com/GitHubAlejandroDR/Microondas_ADominguez.git

Autor

Alejandro Domínguez Recio

Curso

Ingeniería del Software Avanzada

Profesor

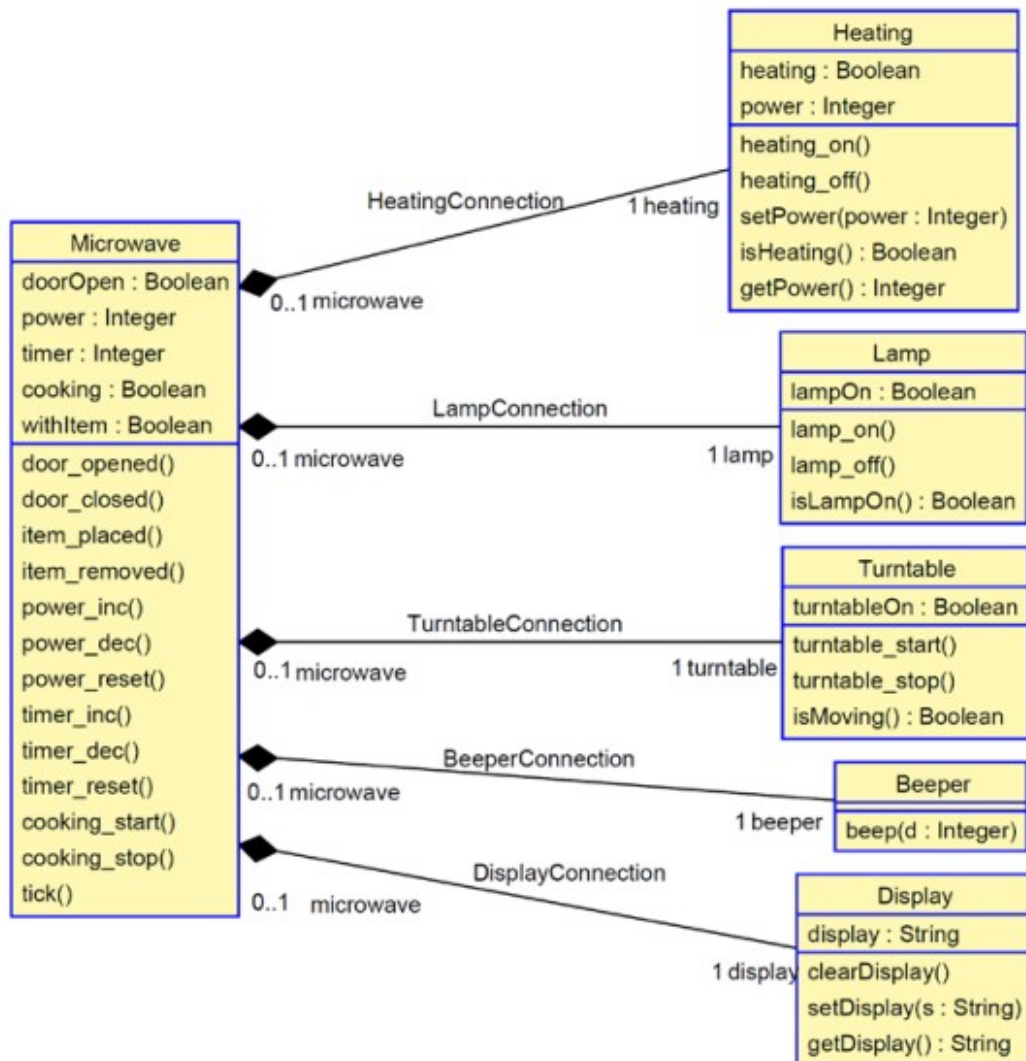
Antonio Jesus Vallecillo Moreno

Índice

ENUNCIADO.....	2
DIAGRAMA DE CLASES – PATRÓN ESTADO.....	5
COMPONENTES.....	6
Clase Heating.....	6
Clase Lam.....	7
Clase Turntable.....	8
Clase Beeper.....	9
Clase Display.....	10
MICROWAVE.....	11
Clase Microwave.....	11
Clase Estado Microwave.....	15
Clase ClosedWithNoItem.....	16
Clase OpenWithNoItem.....	19
Clase OpenWithItem.....	22
Clase CloseWithItem.....	25
Clase Cooking.....	28
JUNIT TEST.....	31
CUCUMBER.....	35
Scenario_Microwave.feature.....	35
StepDefinitions.....	40

ENUNCIADO

Suponemos un horno de microondas que contiene diferentes componentes que interactúan a las órdenes de un componente principal, Microwave, mediante una comunicación basada en operaciones. Los componentes que forman parte del horno, y a los que el propio microondas invoca sus funciones, son los que se muestran en el modelo UML y se describen a continuación:



- Plato giratorio (Turntable). Se activa mediante la operación `turntable_start()` cuando el microondas está en funcionamiento y se para (mediante la operación `turntable_stop()`) cuando se abre la puerta o se acaba tiempo de cocinado. Implementa una operación de consulta `isMoving()` que permite saber en todo momento si el plato está girando o no.

- Campana (Beeper). Avisa cuando el temporizador haya llegado a cero. Su operación `beep(d:Integer)` hace que la campana suene tantas veces como indica el parámetro `d`.
- Lámpara (Lamp) que se enciende (`lamp_on()`) o apaga (`lamp_off()`) en función de distintos eventos, como pueden ser que la puerta esté abierta o que el microondas esté funcionando. Una operación de consulta, `isLampOn()` permite conocer al microondas si la luz está dada o no.
- Unidad de calor (Heating) que es el dispositivo de magnetrón que emite las microondas, encargado de calentar la comida a una determinada potencia (`power`). El microondas lo enciende y apaga usando las operaciones `heating_on()` y `heating_off()`, y también puede conocer si está encendido o no con la operación de consulta `isHeating()`. El componente microondas también puede establecer la potencia y conocerla con las operaciones `setPower()` y `getPower()`.
- Pantalla (Display) que le permite al microondas mostrar distintos mensajes (por ejemplo, “La comida está lista”). En este caso su función principal es mostrar el tiempo restante del temporizador. La operación `clearDisplay()` borra el contenido de la pantalla y la apaga. La pantalla se vuelve a encender cuando se invoca la operación `setDisplay()`

Los atributos que se muestran en los componentes son para su uso interno, a su valor solo puede conocerse y establecerse a través de las correspondientes operaciones.

Por su parte, el propio microondas admite una serie de operaciones que el usuario puede invocar a través de una Interfaz de Usuario (UI), y que sirven para controlar tanto su estado como su comportamiento. Dicha interfaz no se muestra en el diagrama de clases, sino que inicialmente supondremos que es el propio usuario 2 quien invoca las operaciones del microondas, bien mediante acciones físicas (abrir y cerrar la puerta, meter un alimento o retirarlo) o a través de un panel de control que le permite incrementar o disminuir el tiempo de cocinado y la potencia de calentamiento, así como iniciar o parar el cocinado. El estado interno del microondas viene determinado por sus atributos, que solo pueden ser modificados a través de dichas operaciones. Finalmente, un reloj externo es el encargado de invocar la operación `tick()` del microondas cada segundo, permitiéndole conocer así el paso del tiempo.

El modo de funcionamiento normal del microondas es que el usuario abre la puerta, coloca un artículo dentro, cierra la puerta, establece la potencia y el tiempo de cocinado, y presiona el botón de `start_cooking()`. A partir de ese momento el microondas pone a girar el plato, apaga la luz, muestra en el display el tiempo que queda por cocinar, y da la orden a la unidad de calor para que empiece a calentar. Cada vez que se reciba un `tick()` del reloj, el horno decrementa el tiempo restante, y actualiza el valor mostrado en el display. Cuando el timer llega a 0, el horno da la orden a la unidad de calor para que se detenga, la campana suena tres veces, el plato giratorio se para, se

enciende la luz, y en el display se muestra un mensaje que indica que la comida está lista. Si el horno está funcionando y se abre la puerta, el resultado es el mismo, salvo que el temporizador deja de avanzar, manteniendo el valor actual. En cualquier momento, independientemente de si la puerta está abierta o cerrada o el horno está cocinando o no, es posible modificar los valores del temporizador o la potencia de cocinado.

Por supuesto, las operaciones del microondas no pueden invocarse en cualquier orden. Por ejemplo, si la puerta está cerrada no pueden invocarse las operaciones que permiten meter o sacar la comida (`item_removed()` e `item_placed()`). Por razones de seguridad, cuando el usuario pulsa el botón de cocinado (operación `start_cooking()`) y la puerta está abierta, dicha operación no tiene efecto alguno. Igualmente, si no hay ningún artículo en el horno, dicha acción tampoco tiene efecto, aunque la puerta esté cerrada. El siguiente diagrama de estados especifica los comportamientos válidos en cada estado, así como los invariantes que se cumplen en cada uno de los estados.

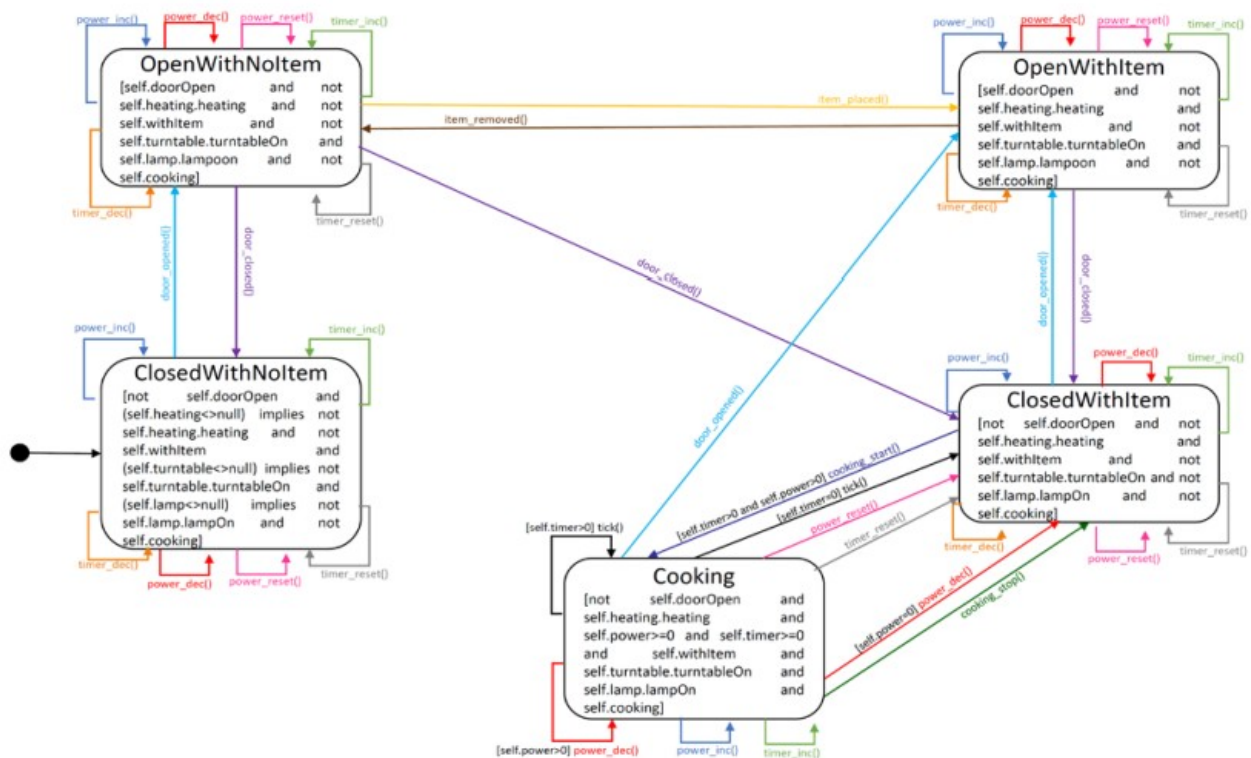
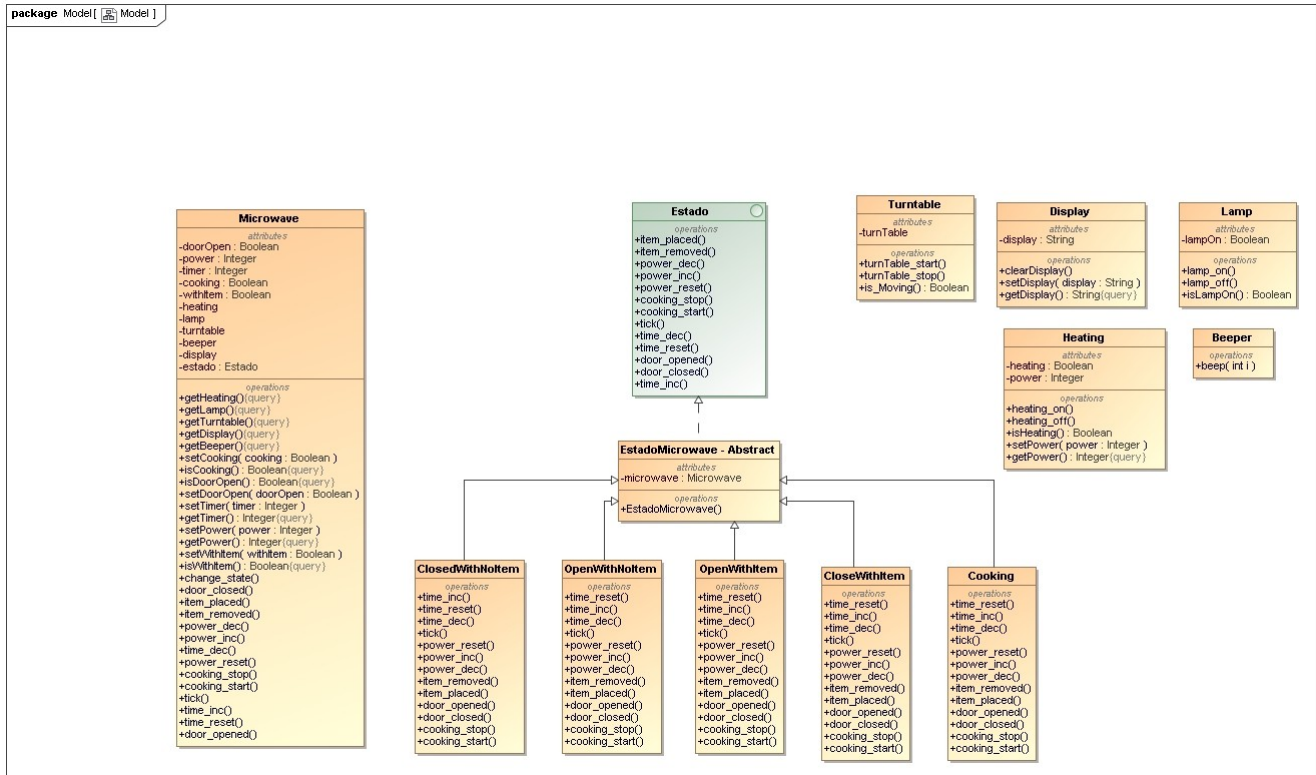


DIAGRAMA DE CLASES – PATRÓN ESTADO



COMPONENTES

Los métodos y variables de los distintos componentes siguen las especificaciones del enunciado. Los componentes no son afectados por ninguna situación que obligen a modificar la lógica de sus métodos.

- **Clase Heating**

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

- **Clase Lam**

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


- **Clase Turntable**

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

- **Clase Beeper**

```
import java.util.concurrent.TimeUnit;

public class Beeper {

    public void beep(int d) {

        for (int i = 1; i <= d; i++) {
            System.out.println("Beep");
            try {
                TimeUnit.SECONDS.sleep(1);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

- **Clase Display**

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

MICROWAVE

La clase microwave almacenará los componentes de los que esta formado. Las variables que representan los componentes tendrán sus respectivos getters posibilitando su acceso desde los distintos estados del microondas.

Los métodos propios del microondas referenciarán su acción al objeto estado. El comportamiento de los métodos del microondas irá acorde al estado en el que se encuentre.

Además se han creado getters/setters que permiten acceder o modificar las variables propias del microondas.

Los métodos que no estén permitidos de ejecutar en los diferentes estados del microondas por situaciones no reales o cuestiones de seguridad lanzarán su correspondiente excepción. Las excepciones lanzadas en un entorno real debería de ser recogidas y tratadas en tiempo de ejecución.

La lógica de estados sigue el mismo criterio que el expuesto en el diagrama de máquinas de estado del enunciado.

Acompañando a cada método hay pequeños comentarios explicando el resultado de su ejecución.

- **Clase Microwave**

```
public class Microwave {

    private Estado estado;

    private Boolean doorOpen;
    private Integer power;
    private Integer timer;
    private Boolean cooking;
    private Boolean withItem;

    private Heating heating;
    private Lamp lamp;
    private Turntable turntable;
    private Beeper beeper;
    private Display display;

    public Microwave() {

        // Inicializamos los componentes
        this.estado = new ClosedWithNoItem(this);
        this.heating = new Heating();
        this.lamp = new Lamp();
        this.turntable = new Turntable();
        this.beeper = new Beeper();
    }
}
```

```

        this.display = new Display();

        // Inicializamos las variables
        power = 0;
        doorOpen = false;
        timer = 0;
        cooking = false;
        withItem = false;
        lamp.lamp_off();
        turntable.turntable_stop();
        heating.heating_off();
        display.setDisplay("");
    }

    // Getters Componentes Microwave////////////////////////////////////

    public Heating getHeating() {
        return heating;
    }

    public Lamp getLamp() {
        return lamp;
    }

    public Turntable getTurntable() {
        return turntable;
    }

    public Beeper getBeeper() {
        return beeper;
    }

    public Display getDisplay() {
        return display;
    }

    // Métodos Getters y Setters Atributos Microwave////////////////////////////////////

    public Boolean getDoorOpen() {
        return doorOpen;
    }

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

    public Integer getPower() {
        return power;
    }

    public void setPower(Integer power) {
        this.power = power;
        heating.setPower(power);
        display.setDisplay("Power:" + power.toString());
    }

```

```

public Integer getTimer() {
    return timer;
}

public void setTimer(Integer timer) {
    this.timer = timer;
    display.setDisplay("Time:" + timer.toString());
}

public Boolean getCooking() {
    return cooking;
}

public void setCooking(Boolean cooking) {
    this.cooking = cooking;
    if (cooking == true) {
        turntable.turntable_start();
        heating.heating_on();
        lamp.lamp_off();
    } else {
        turntable.turntable_stop();
        heating.heating_off();
        lamp.lamp_on();
    }
}

public Boolean getWithItem() {
    return withItem;
}

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

// Métodos Interfaz Microwave //////////////////////////////////////

public void door_opened() {
    estado.door_opened();
}

public void change_state(Estado estado) {
    this.estado = estado;
}

public void door_closed() {
    estado.door_closed();
}

public void item_placed() {
    estado.item_placed();
}

public void item_removed() {
    estado.item_removed();
}

public void power_dec() {

```

```
        estado.power_dec();
    }

    public void power_inc() {
        estado.power_inc();
    }

    public void power_reset() {
        estado.power_reset();
    }

    public void time_inc() {
        estado.time_inc();
    }

    public void time_dec() {
        estado.time_dec();
    }

    public void time_reset() {
        estado.time_reset();
    }

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

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

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

- **Clase Estado Microwave**

```
public abstract class Estado_Microwave implements Estado {  
    protected Microwave microwave;  
    public Estado_Microwave(Microwave microwave) {  
        this.microwave = microwave;  
    }  
}
```


- **Clase ClosedWithNoItem**

```

public class ClosedWithNoItem extends Estado_Microwave {

    public ClosedWithNoItem(Microwave microwave) {
        super(microwave);
        // TODO Auto-generated constructor stub
    }

    @Override
    public void door_opened() {

        // Actualizamos estados de las variables
        microwave.getLamp().lamp_on();
        microwave.setDoorOpen(true);

        // Se pasa al estado OpenWithNoItem
        Estado openwithnoitem = new OpenWithNoItem(microwave);
        microwave.change_state(openwithnoitem);

    }

    @Override
    public void door_closed() {

        // Cerrar la puerta con la puerta cerrada no involucra ninguna acción
        throw new RuntimeException("Door currently closed");

    }

    @Override
    public void item_placed() {

        // Meter comida con la puerta cerrada no involucra ninguna acción
        throw new RuntimeException("Door closed. Action not possible");

    }

    @Override
    public void item_removed() {

        // Sacar comida con la puerta cerrada no involucra ninguna acción
        throw new RuntimeException("Door closed. Action not possible");

    }

    @Override
    public void power_dec() {

        // Casos:
        // Power = 0 -> No acción -> Display "Power: 0"
        // Power > 0 -> Decremento power -> Display "Power: valorActual"
        if (microwave.getPower() == 0) {
            microwave.getDisplay().setDisplay("Power: 0");
        } else {

```

```

        microwave.setPower(microwave.getPower() - 1);
    }

}

@Override
public void power_inc() {

    // Incremento power
    microwave.setPower(microwave.getPower() + 1);

}

@Override
public void power_reset() {

    // Reset power
    microwave.setPower(0);
    microwave.getDisplay().setDisplay("Power Reset");

}

@Override
public void time_inc() {

    // Incremento time
    microwave.setTimer(microwave.getTimer() + 1);

}

@Override
public void time_dec() {

    // Casos:
    // Timer = 0 -> No acción -> Display "Time: 0"
    // Timer > 0 -> Decremento timer -> Display "Time: valorActual"
    if (microwave.getTimer() == 0) {
        microwave.getDisplay().setDisplay("Time: 0");
    } else {
        microwave.setTimer(microwave.getTimer() - 1);
    }
    ;

}

@Override
public void time_reset() {

    // Reset power
    microwave.setTimer(0);
    ;
    microwave.getDisplay().setDisplay("Timer Reset");

}

@Override
public void cooking_start() {

```

```
    // Cooking con la puerta con la puerta cerrada sin comida no involucra  
ninguna    // acción  
            throw new RuntimeException("Action not possible in this state");  
        }  
  
        @Override  
        public void cooking_stop() {  
  
            // Stop Cooking con la puerta con la puerta cerrada sin comida no  
involucra    // ninguna acción  
            throw new RuntimeException("Action not possible in this state");  
        }  
  
        @Override  
        public void tick() {  
  
            // Tick con la puerta con la puerta cerrada sin comida no involucra  
ninguna    // acción  
            throw new RuntimeException("Action not possible in this state");  
        }  
    }  
}
```

- **Clase OpenWithNoItem**

```
public class OpenWithNoItem extends Estado_Microwave {

    public OpenWithNoItem(Microwave microwave) {
        super(microwave);
        // TODO Auto-generated constructor stub
    }

    @Override
    public void door_opened() {

        // Abrir la puerta con la puerta abierta no involucra ninguna acción
        throw new RuntimeException("Door already open");

    }

    @Override
    public void door_closed() {

        // Actualizamos estados de las variables
        microwave.getLamp().lamp_off();
        microwave.setDoorOpen(false);

        // Se pasa al estado ClosedWithNoItem
        Estado closedwithnoitem = new ClosedWithNoItem(microwave);
        microwave.change_state(closedwithnoitem);

    }

    @Override
    public void item_placed() {

        // Actualizamos valor withitem
        microwave.setWithItem(true);

        // Se pasa al estado OpenWithItem
        Estado openwithitem = new OpenWithItem(microwave);
        microwave.change_state(openwithitem);

    }

    @Override
    public void item_removed() {

        // Sacar comida en este estado no involucra ninguna acción
        throw new RuntimeException("No food yet");

    }

    @Override
    public void power_dec() {

        // Casos:
        // Power = 0 -> No acción -> Display "Power: 0"
    }
}
```

```

        // Power > 0 -> Decremento power -> Display "Power: valorActual"
        if (microwave.getPower() == 0) {
            microwave.getDisplay().setDisplay("Power: 0");
        } else {
            microwave.setPower(microwave.getPower() - 1);
        }
    }

    @Override
    public void power_inc() {

        // Incremento power
        microwave.setPower(microwave.getPower() + 1);
    }

    @Override
    public void power_reset() {

        // Reset power
        microwave.setPower(0);
        microwave.getDisplay().setDisplay("Power Reset");
    }

    @Override
    public void time_inc() {

        // Incremento time
        microwave.setTimer(microwave.getTimer() + 1);
    }

    @Override
    public void time_dec() {

        // Casos:
        // Timer = 0 -> No acción -> Display "Time: 0"
        // Timer > 0 -> Decremento timer -> Display "Time: valorActual"
        if (microwave.getTimer() == 0) {
            microwave.getDisplay().setDisplay("Time: 0");
        } else {
            microwave.setTimer(microwave.getTimer() - 1);
        }
    }

    @Override
    public void time_reset() {

        // Reset power
        microwave.setTimer(0);
        microwave.getDisplay().setDisplay("Timer Reset");
    }
}

```

```
@Override
public void cooking_start() {

    // Cooking con la puerta con la puerta abierta sin comida no involucra
    ninguna
    // acción
    throw new RuntimeException("Action not possible in this state");

}

@Override
public void cooking_stop() {

    // Stop Cooking con la puerta con la puerta abierta sin comida no
    involucra
    // ninguna acción
    throw new RuntimeException("Action not possible in this state");

}

@Override
public void tick() {

    // Tick con la puerta con la puerta abierta sin comida no involucra
    ninguna
    // acción
    throw new RuntimeException("Action not possible in this state");

}

}
```

- **Clase OpenWithItem**

```

public class OpenWithItem extends Estado_Microwave {

    public OpenWithItem(Microwave microwave) {
        super(microwave);
        // TODO Auto-generated constructor stub
    }

    @Override
    public void door_opened() {

        // Abrir la puerta con la puerta abierta no involucra ninguna acción
        throw new RuntimeException("Door currently open");

    }

    @Override
    public void door_closed() {

        // Al cerrar la puerta se debe de apagar la lámpara
        microwave.getLamp().lamp_off();
        microwave.setDoorOpen(false);

        // Se pasa al estado ClosednWithItem
        Estado closedwithnoitem = new ClosedWithItem(microwave);
        microwave.change_state(closedwithnoitem);

    }

    @Override
    public void item_placed() {

        // Meter comida con comida dentro no involucra ninguna acción
        throw new RuntimeException("Item currently inside");

    }

    @Override
    public void item_removed() {

        // Actualizamos valor withItem
        microwave.setWithItem(false);

        // Se pasa al estado OpenWithNoItem
        Estado openwithnoitem = new OpenWithNoItem(microwave);
        microwave.change_state(openwithnoitem);

    }

    @Override
    public void power_dec() {

        // Casos:

```

```

    // Power = 0 -> No acción -> Display "Power: 0"
    // Power > 0 -> Decremento power -> Display "Power: valorActual"
    if (microwave.getPower() == 0) {
        microwave.getDisplay().setDisplay("Power: 0");
    } else {
        microwave.setPower(microwave.getPower() - 1);
    }
}

@Override
public void power_inc() {

    // Incremento power
    microwave.setPower(microwave.getPower() + 1);

}

@Override
public void power_reset() {

    // Reset power
    microwave.setPower(0);
    microwave.getDisplay().setDisplay("Power Reset");

}

@Override
public void time_inc() {

    // Incremento time
    microwave.setTimer(microwave.getTimer() + 1);

}

@Override
public void time_dec() {

    // Casos:
    // Timer = 0 -> No acción -> Display "Time: 0"
    // Timer > 0 -> Decremento timer -> Display "Time: valorActual"
    if (microwave.getTimer() == 0) {
        microwave.getDisplay().setDisplay("Time: 0");
    } else {
        microwave.setTimer(microwave.getTimer() - 1);
    }

}

@Override
public void time_reset() {

    // Reset power
    microwave.setTimer(0);
    microwave.getDisplay().setDisplay("Timer Reset");

}

```



```
@Override
public void cooking_start() {

    // Cooking con la puerta con la puerta abierta no involucra ninguna acción
    throw new RuntimeException("Action not possible in this state");

}

@Override
public void cooking_stop() {

    // Stop Cooking con la puerta con la puerta abierta no involucra ninguna
    acción throw new RuntimeException("Action not possible in this state");

}

@Override
public void tick() {

    // Tick con la puerta con la puerta abierta no involucra ninguna acción
    throw new RuntimeException("Action not possible in this state");

}

}
```

• Clase CloseWithItem

```
public class ClosedWithItem extends Estado_Microwave {

    public ClosedWithItem(Microwave microwave) {
        super(microwave);
        // TODO Auto-generated constructor stub
    }

    @Override
    public void door_opened() {

        // Al abrir la puerta se debe de encender la lámpara
        microwave.getLamp().lamp_on();
        microwave.setDoorOpen(true);

        // Se pasa al estado OpenWithNoItem
        Estado openwithitem = new OpenWithItem(microwave);
        microwave.change_state(openwithitem);
    }

    @Override
    public void door_closed() {

        // Cerrar la puerta con la puerta cerrada no involucra ninguna accin
        throw new RuntimeException("Door currently closed");
    }

    @Override
    public void item_placed() {

        // Meter comida con la puerta cerrada no involucra ninguna accin
        throw new RuntimeException("Door currently closed");
    }

    @Override
    public void item_removed() {

        // Sacar comida con la puerta cerrada no involucra ninguna accin
        throw new RuntimeException("Door currently closed");
    }

    @Override
    public void power_dec() {

        // Casos:
        // Power = 0 -> No acción -> Display "Power: 0"
        // Power > 0 -> Decremento power -> Display "Power: valorActual"
        if (microwave.getPower() == 0) {
            microwave.getDisplay().setDisplay("Power: 0");
        } else {
```

```

        microwave.setPower(microwave.getPower() - 1);
    }

}

@Override
public void power_inc() {

    // Incremento power
    microwave.setPower(microwave.getPower() + 1);
}

@Override
public void power_reset() {

    // Reset power
    microwave.setPower(0);
    microwave.getDisplay().setDisplay("Power Reset");
}

@Override
public void time_inc() {

    // Incremento time
    microwave.setTimer(microwave.getTimer() + 1);
}

@Override
public void time_dec() {

    // Casos:
    // Timer = 0 -> No acción -> Display "Time: 0"
    // Timer > 0 -> Decremento timer -> Display "Time: valorActual"
    if (microwave.getTimer() == 0) {
        microwave.getDisplay().setDisplay("Time: 0");
    } else {
        microwave.setTimer(microwave.getTimer() - 1);
    }
}

@Override
public void time_reset() {

    // Reset power
    microwave.setTimer(0);
    microwave.getDisplay().setDisplay("Timer Reset");
}

@Override
public void cooking_start() {

    if (microwave.getPower() > 0 & microwave.getTimer() > 0) {

```

```
// Se actualiza estado de variables
microwave.setCooking(true);

// Se pasa al estado Cooking
Estado cooking = new Cooking(microwave);
microwave.change_state(cooking);
} else {
    microwave.getDisplay().setDisplay("Error Configuration");
}

}

@Override
public void cooking_stop() {

    // Cooking Stop en este estado no involucra ninguna accion
    throw new RuntimeException("Action not possible in this state");

}

@Override
public void tick() {

    // Cooking Stop en este estado no involucra ninguna accion
    throw new RuntimeException("Action not possible in this state");

}

}
```

- **Clase Cooking**

```

public class Cooking extends Estado_Microwave {

    public Cooking(Microwave microwave) {
        super(microwave);
    }

    @Override
    public void door_opened() {

        // Al abrir la puerta se debe de encender la lámpara
        microwave.setDoorOpen(true);
        microwave.setCooking(false);

        // Se pasa al estado OpenWithNoItem
        Estado openwithitem = new OpenWithItem(microwave);
        microwave.change_state(openwithitem);

    }

    @Override
    public void door_closed() {

        // Cerrar la puerta con la puerta cerrada no involucra ninguna acción
        throw new RuntimeException("Door currently closed");

    }

    @Override
    public void item_placed() {

        // Meter comida con la puerta cerrada no involucra ninguna acción
        throw new RuntimeException("Item currently inside");

    }

    @Override
    public void item_removed() {

        // Sacar comida con la puerta cerrada no involucra ninguna acción
        throw new RuntimeException("Action not possible in this state");

    }

    @Override
    public void power_dec() {

        // Casos:
        // Power = 0 -> No acción -> Display "Power: 0"
        // Power > 0 -> Decremento power -> Display "Power: valorActual"
        if (microwave.getPower() == 0) {
            microwave.getDisplay().setDisplay("Power: 0");
        }
    }
}

```

```

        microwave.setCooking(false);
        Estado closedwithitem = new ClosedWithItem(microwave);
        microwave.change_state(closedwithitem);
    } else {
        microwave.setPower(microwave.getPower() - 1);
    }
}

@Override
public void power_inc() {

    microwave.setPower(microwave.getPower() + 1);

}

@Override
public void power_reset() {

    // Reset power
    microwave.setPower(0);
    microwave.getDisplay().setDisplay("Power Reset");
    microwave.setCooking(false);
    Estado closedwithitem = new ClosedWithItem(microwave);
    microwave.change_state(closedwithitem);
}

@Override
public void time_inc() {

    // Incremento time
    microwave.setTimer(microwave.getTimer() + 1);

}

@Override
public void time_dec() {

    // Casos:
    // Timer = 0 -> No acción -> Display "Time: 0" -> Se pasa al estado
    // OpenWithNoItem
    // Timer > 0 -> Decremento timer -> Display "Time: valorActual"
    if (microwave.getTimer() > 1) {
        microwave.setTimer(microwave.getTimer() - 1);
        System.out.println(microwave.getTimer().toString());
    } else if (microwave.getTimer() == 1) {
        microwave.setTimer(microwave.getTimer() - 1);
        microwave.getBeeper().beep(3);
        microwave.getDisplay().setDisplay("Food ready!!");
        this.cooking_stop();
    }

}

@Override
public void time_reset() {

```

```
// Reset power
microwave.setTimer(0);
microwave.getDisplay().setDisplay("Timer Reset");
microwave.setCooking(false);
Estado closedwithitem = new ClosedWithItem(microwave);
microwave.change_state(closedwithitem);

}

@Override
public void cooking_start() {

    // Cooking en el estado cooking on no involucra ninguna acción
    throw new RuntimeException("Microwave currently working");

}

@Override
public void cooking_stop() {

    // Se actualiza cooking
    microwave.setCooking(false);

    // Se pasa al estado OpenWithNoItem
    Estado closedwithitem = new ClosedWithItem(microwave);
    microwave.change_state(closedwithitem);

}

@Override
public void tick() {

    this.time_dec();

}

}
```

JUNIT TEST

Las distintas pruebas realizadas tienen el fin de verificar el correcto funcionamiento del microondas en distintos tipos de situaciones o estados. Se comprobarán que las variables y métodos cambian acorde al estado del microondas.

```
public class Test_Microwave {

    @Test
    public void tests(){

        // Comprobamos las variables y operaciones en el estado ClosedWithNoItem
        Microwave microwave = new Microwave();
        assertEquals(0,microwave.getPower());
        assertEquals(0,microwave.getTimer());
        assertEquals(false,microwave.getWithItem());
        assertEquals(false,microwave.getDoorOpen());
        assertEquals(false,microwave.getCooking());
        assertEquals(false,microwave.getLamp().isLampOn());
        assertEquals(false,microwave.getLamp().isLampOn());
        assertThrows(RuntimeException.class, () -> { microwave.cooking_start();});
        assertThrows(RuntimeException.class, () -> { microwave.tick();});
        assertThrows(RuntimeException.class, () -> { microwave.door_closed();});
        assertThrows(RuntimeException.class, () -> { microwave.cooking_stop();});

        // Comprobamos que no se permiten valores negativos de time y power
        microwave.power_dec();
        microwave.time_dec();
        assertEquals(0,microwave.getPower());
        assertEquals(0,microwave.getTimer());

        // Comprobamos variables y operaciones en el estado OpenWithNoItem
        microwave.door_opened();
        assertEquals(0,microwave.getPower());
        assertEquals(0,microwave.getTimer());
        assertEquals(false,microwave.getWithItem());
        assertEquals(true,microwave.getDoorOpen());
        assertEquals(false,microwave.getCooking());
        assertEquals(true,microwave.getLamp().isLampOn());
        assertThrows(RuntimeException.class, () -> { microwave.cooking_start();});
        assertThrows(RuntimeException.class, () -> { microwave.tick();});
        assertThrows(RuntimeException.class, () -> { microwave.door_opened();});
        assertThrows(RuntimeException.class, () -> { microwave.cooking_stop();});
        assertThrows(RuntimeException.class, () -> { microwave.item_removed();});

        // Comprobamos variables y operaciones en el estado OpenWithItem
        microwave.item_placed();
        assertEquals(0,microwave.getPower());
        assertEquals(0,microwave.getTimer());
        assertEquals(true,microwave.getWithItem());
```



```

assertEquals(true,microwave.getDoorOpen());
assertEquals(false,microwave.getCooking());
assertEquals(true,microwave.getLamp().isLampOn());
assertThrows(RuntimeException.class, () -> { microwave.cooking_start();});
assertThrows(RuntimeException.class, () -> { microwave.tick();});
assertThrows(RuntimeException.class, () -> { microwave.door_opened();});
assertThrows(RuntimeException.class, () -> { microwave.cooking_stop();});
assertThrows(RuntimeException.class, () -> { microwave.item_placed();});

```

```

// Comprobamos variables y operaciones en el estado ClosedWithItem

```

```

microwave.door_closed();
assertEquals(0,microwave.getPower());
assertEquals(0,microwave.getTimer());
assertEquals(true,microwave.getWithItem());
assertEquals(false,microwave.getDoorOpen());
assertEquals(false,microwave.getCooking());
assertEquals(false,microwave.getLamp().isLampOn());
assertThrows(RuntimeException.class, () -> { microwave.tick();});
assertThrows(RuntimeException.class, () -> { microwave.door_closed();});
assertThrows(RuntimeException.class, () -> { microwave.cooking_stop();});
assertThrows(RuntimeException.class, () -> { microwave.item_placed();});

```

```

// Comprobamos variables en el estado ClosedWithItem con configuración no

```

permitida

```

microwave.cooking_start();
assertEquals(0,microwave.getPower());
assertEquals(0,microwave.getTimer());
assertEquals(true,microwave.getWithItem());
assertEquals(false,microwave.getDoorOpen());
assertEquals(false,microwave.getCooking());
assertEquals(false,microwave.getLamp().isLampOn());

```

```

// Comprobamos variables y operaciones en el estado Cooking

```

```

microwave.time_inc();
microwave.power_inc();
assertEquals(1,microwave.getPower());
assertEquals(1,microwave.getTimer());
microwave.cooking_start();
microwave.tick();
assertEquals(1,microwave.getPower());
assertEquals(0,microwave.getTimer());
assertEquals(true,microwave.getWithItem());
assertEquals(false,microwave.getDoorOpen());
assertEquals(false,microwave.getCooking());
assertEquals(true,microwave.getLamp().isLampOn());

```

```

// Comprobamos variables y operaciones en el estado Cooking abriendo la
puerta antes de terminar

```

```

microwave.setTimer(20);
assertEquals(1, microwave.getPower());
microwave.cooking_start();
// "Reloj externo" -> Simulamos 15 segundos
for(int i = 1;i <= 15;i++) {
    microwave.tick();
}
microwave.door_opened();

```

```

assertEquals(1, microwave.getPower());
assertEquals(5, microwave.getTimer());
assertEquals(true, microwave.getWithItem());
assertEquals(true, microwave.getDoorOpen());
assertEquals(false, microwave.getCooking());
assertEquals(true, microwave.getLamp().isLampOn());
assertThrows(RuntimeException.class, () -> { microwave.tick();});
assertThrows(RuntimeException.class, () -> { microwave.door_opened();});
assertThrows(RuntimeException.class, () -> { microwave.cooking_stop();});
assertThrows(RuntimeException.class, () -> { microwave.item_placed();});

// Comprobamos la posibilidad de aumentar la potencia el tiempo y potencia
mientras el microondas está calentado
microwave.door_closed();
microwave.setTimer(15);
microwave.cooking_start();
// "Reloj externo" -> Simulamos 15 segundos
for(int i = 1; i <= 10; i++) {
    microwave.tick();
}
microwave.time_inc();
microwave.power_inc();
assertEquals(2, microwave.getPower());
assertEquals(6, microwave.getTimer());
// "Reloj externo" -> Simulamos los 6 segundos restantes
for(int i = 1; i <= 6; i++) {
    microwave.tick();
}
assertEquals(2, microwave.getPower());
assertEquals(0, microwave.getTimer());
assertEquals(true, microwave.getWithItem());
assertEquals(false, microwave.getDoorOpen());
assertEquals(false, microwave.getCooking());
assertEquals(true, microwave.getLamp().isLampOn());
assertThrows(RuntimeException.class, () -> { microwave.tick();});
assertThrows(RuntimeException.class, () -> { microwave.door_closed();});
assertThrows(RuntimeException.class, () -> { microwave.cooking_stop();});
assertThrows(RuntimeException.class, () -> { microwave.item_placed();});

//Volvemos en los estados desde Cooking
//Comprobamos variables estado OpenWithItem
microwave.door_opened();
assertEquals(2, microwave.getPower());
assertEquals(0, microwave.getTimer());
assertEquals(true, microwave.getWithItem());
assertEquals(true, microwave.getDoorOpen());
assertEquals(false, microwave.getCooking());
assertEquals(false, microwave.getTurntable().isMoving());
assertEquals(true, microwave.getLamp().isLampOn());
assertThrows(RuntimeException.class, () -> { microwave.tick();});
assertThrows(RuntimeException.class, () -> { microwave.door_opened();});
assertThrows(RuntimeException.class, () -> { microwave.cooking_stop();});
assertThrows(RuntimeException.class, () -> { microwave.item_placed();});

// Comprobamos variables estado OpenWithNoItem
microwave.item_removed();
assertEquals(2, microwave.getPower());

```

```

    assertEquals(0,microwave.getTimer());
    assertEquals(false,microwave.getWithItem());
    assertEquals(true,microwave.getDoorOpen());
assertEquals(false,microwave.getCooking());
    assertEquals(true,microwave.getLamp().isLampOn());
    assertThrows(RuntimeException.class, () -> { microwave.tick();});
    assertThrows(RuntimeException.class, () -> { microwave.door_opened();});
    assertThrows(RuntimeException.class, () -> { microwave.cooking_stop();});
    assertThrows(RuntimeException.class, () -> { microwave.item_removed();});

    // Comprobamos variables estado ClosedWithNoItem
    microwave.door_closed();
    assertEquals(2,microwave.getPower());
    assertEquals(0,microwave.getTimer());
    assertEquals(false,microwave.getWithItem());
    assertEquals(false,microwave.getDoorOpen());
    assertEquals(false,microwave.getCooking());
    assertEquals(false,microwave.getLamp().isLampOn());
    assertThrows(RuntimeException.class, () -> { microwave.tick();});
    assertThrows(RuntimeException.class, () -> { microwave.door_closed();});
    assertThrows(RuntimeException.class, () -> { microwave.cooking_stop();});
    assertThrows(RuntimeException.class, () -> { microwave.item_removed();});

}

}

```

CUCUMBER

- **Scenario_Microwave.feature**

Feature: Funcionamiento de estados del microondas y componentes asociados.

@ScenarioOpenWithNoItem

Scenario: Como usuario quiero encender el microondas y abrir la puerta

Given probar el microondas
When pulso abrir la puerta
Then la puerta deberá de abrirse
And la luz deberá encenderse

@ScenarioClosedWithItem

Scenario: Como usuario quiero encender el microondas, abrir la puerta, introducir un alimento y cerrar la puerta

Given probar el microondas
When pulso abrir la puerta
And introduzco un alimento
And cierro la puerta
Then la puerta deberá de estar cerrada
And la luz deberá apagarse
And deberá de mostrar la presencia de un alimento en el interior

@ScenarioOpenClosedPowerTimeInc

Scenario: Como usuario quiero encender el microondas, abrir la puerta, introducir un alimento y incrementar el tiempo y potencia

Given probar el microondas
When pulso abrir la puerta
And introduzco un alimento
And incremento la potencia 1.0
And incremento el tiempo 1.0 segundos
Then la puerta deberá de abrirse
And deberá de mostrar la presencia de un alimento en el interior
And el tiempo deber ser 1.0
And la potencia debe ser 1.0

@ScenarioCookingFinish

Scenario: Como usuario quiero encender el microondas, abrir la puerta, introducir un alimento, cerrar la puerta, incrementar el tiempo y potencia y cocinar el alimento

Given probar el microondas
When pulso abrir la puerta
And introduzco un alimento
And incremento la potencia 1.0
And incremento el tiempo 1.0 segundos
And cierro la puerta
And pulso start cooking
And y espero 1.0 segundo
Then deberá de mostrar la presencia de un alimento en el interior
And la puerta deberá de estar cerrada
And el tiempo deber ser 0.0

And la potencia debe ser 1.0
 And la luz deberá encenderse

@ScenarioStopCooking

Scenario: Como usuario quiero encender el microondas, abrir la puerta, introducir un alimento, cerrar la puerta, incrementar el tiempo y potencia, cocinar el alimento y para la coción antes de terminar

Given probar el microondas
 When pulso abrir la puerta
 And introduzco un alimento
 And incremento la potencia 1.0
 And incremento el tiempo 3.0 segundos
 And cierro la puerta
 And pulso start cooking
 And y espero 1.0 segundo
 And pulso stop cooking
 Then deberá de mostrar la presencia de un alimento en el interior
 And la puerta deberá de estar cerrada
 And el tiempo deber ser 2.0
 And la potencia debe ser 1.0
 And la luz deberá encenderse

@ScenarioHeatingStop

Scenario: Como usuario quiero encender el microondas, abrir la puerta, introducir un alimento, cerrar la puerta, incrementar el tiempo y potencia, cocinar el alimento, para la coción antes de terminar y verificar que el plato no esta girando ni el microondas calentando

Given probar el microondas
 When pulso abrir la puerta
 And introduzco un alimento
 And incremento la potencia 1.0
 And incremento el tiempo 3.0 segundos
 And cierro la puerta
 And pulso start cooking
 And y espero 1.0 segundo
 And pulso stop cooking
 Then deberá de mostrar la presencia de un alimento en el interior
 And la puerta deberá de estar cerrada
 And el tiempo deber ser 2.0
 And la potencia debe ser 1.0
 And la luz deberá encenderse
 And el microondas parar de calentar

@ScenarioDisplay

Scenario: Como usuario quiero encender el microondas, abrir la puerta, introducir un alimento, cerrar la puerta, incrementar el tiempo y potencia y ver cuanto se ha incrementado

Given probar el microondas
 When pulso abrir la puerta
 And introduzco un alimento
 And cierro la puerta
 And incremento la potencia 1.0
 And incremento el tiempo 1.0 segundos
 Then la puerta deberá de estar cerrada
 And la luz deberá apagarse
 And deberá de mostrar la presencia de un alimento en el interior
 And el display deberá de mostrar "Time:1"

`@ScenarioOpenDoorCooking`

Scenario: Como usuario quiero encender el microondas, abrir la puerta, introducir un alimento, cerrar la puerta, incrementar el tiempo y potencia, cocinar el alimento y abrir la puerta antes de terminar

```

Given probar el microondas
When pulso abrir la puerta
And introduzco un alimento
And incremento la potencia 1.0
And incremento el tiempo 3.0 segundos
And cierro la puerta
And pulso start cooking
And y espero 1.0 segundo
And pulso abrir la puerta
Then deberá de mostrar la presencia de un alimento en el interior
And la puerta deberá de abrirse
And la luz deberá encenderse
And el tiempo deber ser 2.0
And la potencia debe ser 1.0
And la luz deberá encenderse

```

`@ScenarioStopCooking_outline`

Scenario: Como usuario quiero encender el microondas, abrir la puerta, introducir un alimento, cerrar la puerta, incrementar el tiempo y potencia, cocinar el alimento y terminar la coción

```

Given probar el microondas
When pulso abrir la puerta
And introduzco un alimento
And incremento la potencia 1.0
And incremento el tiempo <value1> segundos
And cierro la puerta
And pulso start cooking
And y espero <value2> segundo
Then deberá de mostrar la presencia de un alimento en el interior
And la puerta deberá de estar cerrada
And el tiempo deber ser <status>
And la potencia debe ser 1.0

```

Examples:

value1	value2	status			
		2.0		2.0	0.0
	1.0	0.0		1.0	
	3.0	3.0		0.0	
	2.0	-2.0		2.0	

`@ScenarioCookingDoorOpen_outline`

Scenario: Como usuario quiero encender el microondas, abrir la puerta, introducir un alimento, cerrar la puerta, incrementar el tiempo y potencia, cocinar el alimento y para la coción en varios instantes

```

Given probar el microondas
When pulso abrir la puerta
And introduzco un alimento
And incremento la potencia 1.0
And incremento el tiempo <value1> segundos

```

```

And cierro la puerta
And pulso start cooking
And y espero <value2> segundo
And pulso abrir la puerta
Then deberá de mostrar la presencia de un alimento en el interior
And la puerta deberá de abrirse
And el tiempo deber ser <status>
And la potencia debe ser 1.0
And la luz deberá encenderse

```

Examples:

value1	value2	status			
		2.0		2.0	0.0
	1.0	0.0		1.0	
	3.0	3.0		0.0	
	2.0	-2.0		2.0	

@ScenarioCookingPauseOpenClose_outline

Scenario: Como usuario quiero encender el microondas, abrir la puerta, introducir un alimento, cerrar la puerta, incrementar el tiempo y potencia, cocinar el alimento y para la coción en varios instantes

```

Given probar el microondas
When pulso abrir la puerta
And introduzco un alimento
And incremento la potencia 1.0
And incremento el tiempo <value1> segundos
And cierro la puerta
And pulso start cooking
And y espero <value2> segundo
And pulso abrir la puerta
And cierro la puerta
And pulso start cooking
And y espero <value3> segundo
Then deberá de mostrar la presencia de un alimento en el interior
And el tiempo deber ser <status>
And la potencia debe ser 1.0
And la luz deberá encenderse

```

Examples:

value1	value2	value3	status			
		2.0		1.0	1.0	0.0
	1.0	0.0		1.0	0.0	
	3.0	1.0		2.0	0.0	
	2.0	-2.0		2.0	0.0	

@ScenarioCookingPauseOpenClose_outline

Scenario: Como usuario quiero encender el microondas, abrir la puerta, introducir un alimento, cerrar la puerta, incrementar el tiempo y potencia, cocinar el alimento y para la coción en varios instantes

```

Given probar el microondas
When pulso abrir la puerta
And introduzco un alimento
And incremento la potencia 1.0
And incremento el tiempo <value1> segundos
And cierro la puerta
And pulso start cooking

```

```

And y espero <value2> segundo
And pulso abrir la puerta
And cierro la puerta
And pulso start cooking
And y espero <value3> segundo
Then deberá de mostrar la presencia de un alimento en el interior
And el tiempo deber ser <status>
And la potencia debe ser 1.0

```

Examples:

value1	value2	value3	status				
		2.0		1.0	1.0	0.0	
	1.0	0.0	1.0	0.0			
	3.0	1.0	0.0	2.0			
	2.0	-2.0	2.0	0.0			

• StepDefinitions

```

import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Assertions.*;

public class StepDefinitions {

    private Microwave microwave;

    @Given("probar el microondas")
    public void probar_el_microondas() {
        microwave = new Microwave();
    }

    @Then("la puerta deberá de abrirse")
    public void la_puerta_deberá_de_abrirse() {
        assertEquals(true, microwave.getDoorOpen());
    }

    @Then("la luz deberá encenderse")
    public void y_la_luz_deberá_de_encenderse() {
        assertEquals(true, microwave.getLamp().isLampOn());
    }

    @When("pulso abrir la puerta")
    public void pulso_abrir_la_puerta() {
        microwave.door_opened();
    }

    @When("introduzco un alimento")
    public void introduzco_un_alimento() {
        microwave.item_placed();
    }

    @When("cierro la puerta")
    public void cierro_la_puerta() {
        microwave.door_closed();
    }

    @Then("la puerta deberá de estar cerrada")
    public void la_puerta_deberá_de_estar_cerrada() {
        assertEquals(false, microwave.getDoorOpen());
    }

    @Then("la luz deberá apagarse")
    public void la_luz_apagada() {
        assertEquals(false, microwave.getLamp().isLampOn());
    }
}

```

```

}

@Then("deberá de mostrar la presencia de un alimento en el interior")
public void deberá_de_mostrar_la_presencia_de_un_alimento_en_el_interior() {
    assertEquals(true, microwave.getWithItem());
}

@When("incremento la potencia {double}")
public void incremento_la_potencia(Double double1) {
    microwave.setPower(double1.intValue());
}

@When("incremento el tiempo {double} segundos")
public void incremento_el_tiempo_segundo(Double double1) {
    microwave.setTimer(double1.intValue());
}

@Then("la deberá de abrirse")
public void la_deberá_de_abrirse() {
    assertEquals(true, microwave.getDoorOpen());
}

@Then("el tiempo deber ser {double}")
public void el_tiempo_deber_ser(Double double1) {
    assertEquals(double1.intValue(), microwave.getTimer());
}

@Then("la potencia debe ser {double}")
public void la_potencia_debe_ser(Double double1) {
    assertEquals(double1.intValue(), microwave.getPower());
}

@When("pulso start cooking")
public void pulso_start_cooking() {
    microwave.cooking_start();
}

@When("y espero {double} segundo")
public void y_espero_segundo(Double double1) {
    for(int i = 1; i <= double1; i++) {
        microwave.tick();
    }
}

@When("pulso stop cooking")
public void pulso_stop_cooking() {
    microwave.cooking_stop();
}

@Then("el plato deberá de pararse")
public void el_plato_deberá_de_pararse() {
    assertEquals(false, microwave.getTurntable().isMoving());
}

@Then("el microondas parar de calentar")
public void el_microondas_parar_de_calentar() {

```

```
        assertEquals(false, microwave.getHeating().isHeating());
    }

    @Then("el display deberá de mostrar {string}")
    public void el_display_deberá_de_mostrar(String string) {
        assertEquals(string, microwave.getDisplay().getDisplay());
    }

}
```