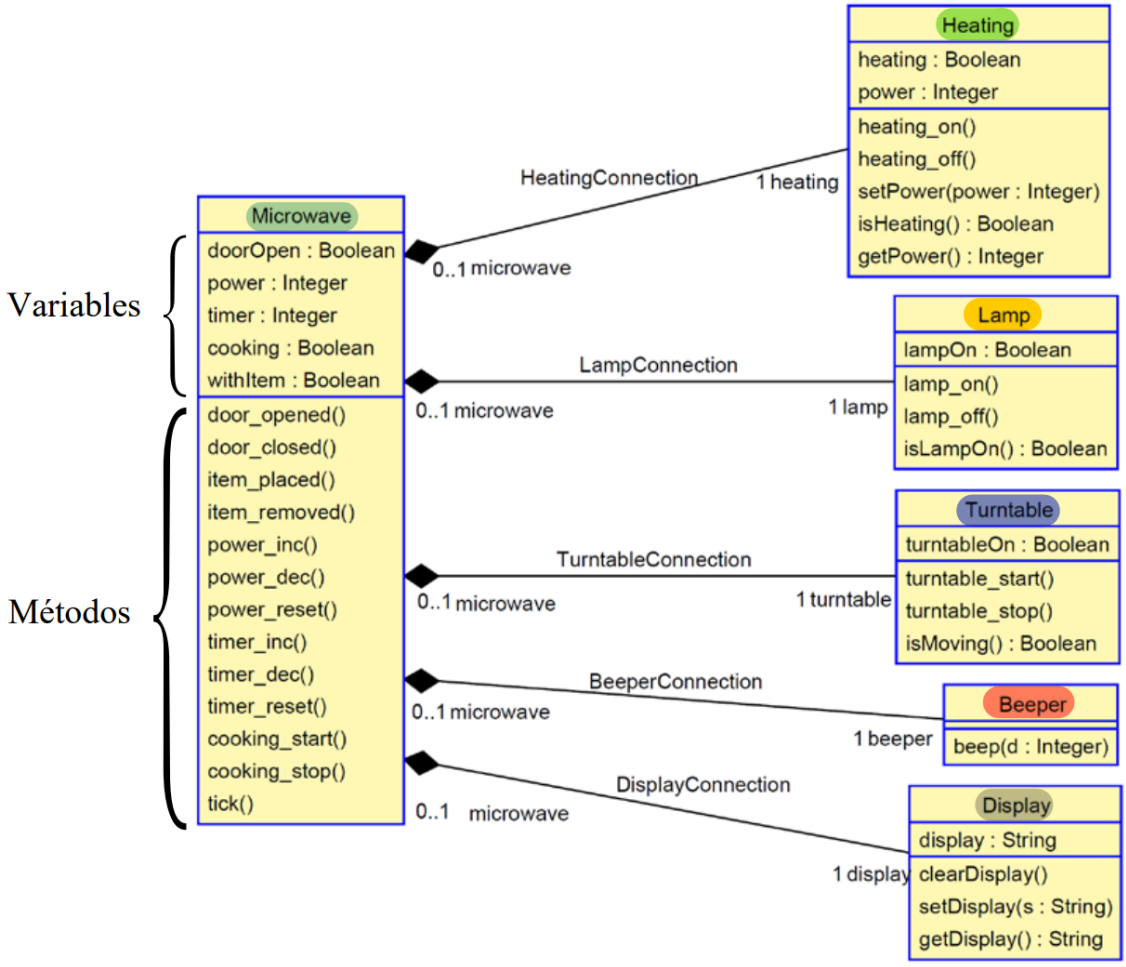
Ingeniería del Software Avanzada

Práctica Final: Horno Microondas



Autor: David Ramírez Arco

Fecha: 10 de junio de 2022

GitHub: https://github.com/Archerd6/Proyecto_Microondas

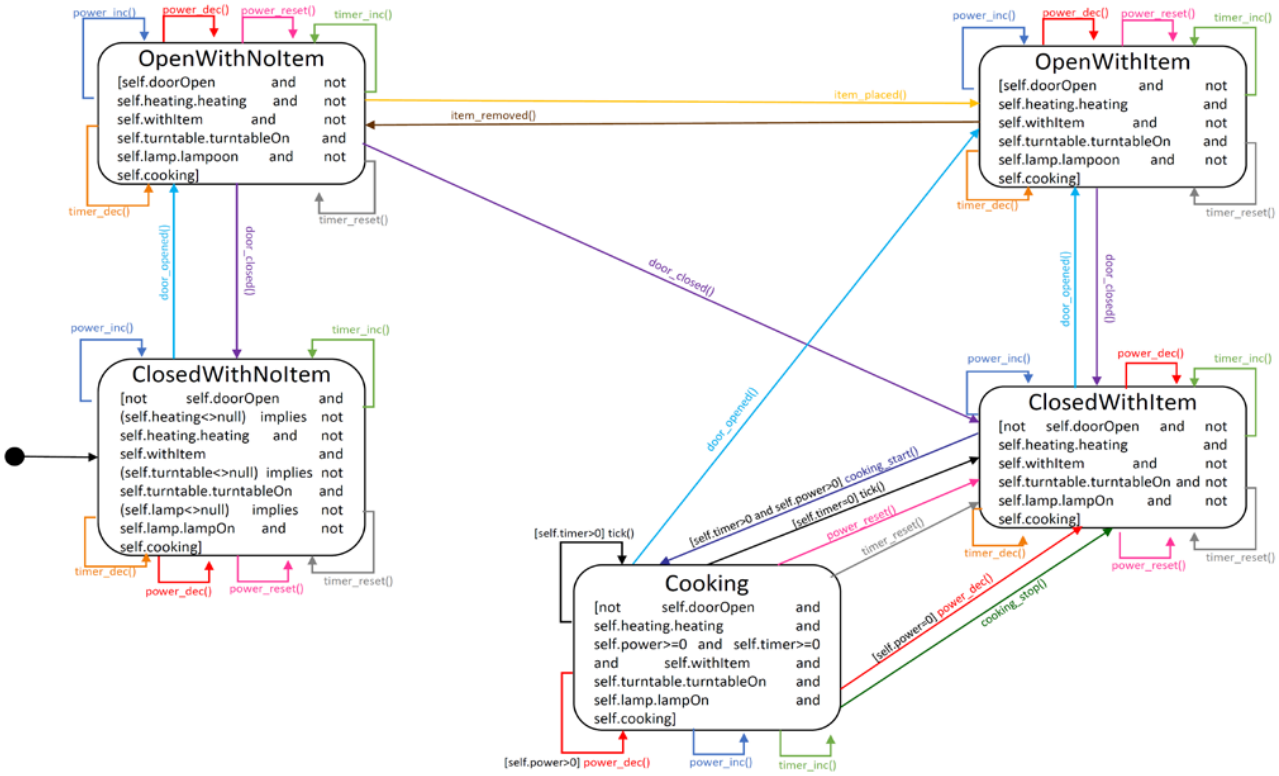
Índice

Apartados

Apartado A

Implementar el sistema en Java, usando el patrón de diseño Estado

El patrón diseño Estado se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado del mismo, y en este caso, se nos muestra cuales son:



He seguido también el Diagrama UML aunque he necesitado crear algunos métodos mas para poder realizar las comprobaciones y acciones.

Todas las clases creadas del microondas (sin las pruebas de Junit y Gherkin-Cucumber) están en la directorio microwave

(**Enlaze** al directorio GitHub)

La interfaz de microondas que he creado si sigue el Diagrama:

```
1   package microwave;
2
3   public interface Microwave_Interface
4   {
5           /**Increment on 10w the power*/
6           public void power_inc(Microwave mw);
7           /**Decrement on 10w the power*/
8           public void power_dec(Microwave mw);
9           /**Set power to 0w*/
10          public void power_reset(Microwave mw);
11          /**Increase the timer*/
12          public void timer_inc(Microwave mw);
13          /**Decrease the timer*/
14          public void timer_dec(Microwave mw);
15          /**Set the timer to 0*/
16          public void timer_reset(Microwave mw);
17          /**Try to start the microwave*/
18
19          /**Try to open the door*/
20          public void door_opened(Microwave mw);
21          /**Try to close the door*/
22          public void door_closed(Microwave mw);
23          /**Try to place a item*/
24          public void item_placed(Microwave mw);
25          /**Try to remove a item*/
26          public void item_removed(Microwave mw);
27
28          public void cooking_start(Microwave mw);
29          /**Try to stop the microwave*/
30          public void cooking_stop(Microwave mw);
31          /**Once invoked this reduce the time remaining in timer one unit*/
32          public void tick(Microwave mw);
33  }
```

Beeper

15 lines (14 sloc) | 279 Bytes

Raw Blame

```
package microwave;

/**Bell that warns when the timer has reached zero
 * @author ap27r
 * */
public class Beeper
{
        /**Makes the bell ring as many times as indicated by the parameter
         * @param d - Times to beep
         * */
        public void beep(int d)
        {
                BeeperCounter.listen(d);
        }
}
```

BeeperCounter

He creado una clase pública llamada BeeperCounter para recibir el parámetro utilizado en la clase Beeper, lo guarda y comprueba para la hora de hacer las distintas pruebas en el proyecto.

31 lines (28 sloc) | 632 Bytes

Raw Blame

```
package microwave;
/**Listener to the Beeper class, as Beeper don't have variables to save in the UML diagram
 * @author ap27r
 * */
public class BeeperCounter
{
        /**Variables to save the number of beeps*/
        private static int num_beeps;

        /**Main metod used by the Beeper class
         * @param times_beeped
         * */
        public static void listen(int times_beeped)
        {
                num_beeps = times_beeped;
        }

        /**Has the beep beeped?*/
        public static boolean isBeeped(int t)
        {
                return (clear() == t);
        }

        /**Reset number of beep and return old value*/
        private static int clear()
        {
                int bp = num_beeps;
                num_beeps = 0;
                return bp;
        }
}
```

Display

26 lines (24 sloc) | 579 Bytes

Raw Blame

```java
package microwave;

/**Display that allows the microwave to show different messages (for example, “The food is ready”)*/
public class Display
{
	//Variable
	private String display;

	/**The operation clearDisplay() clears the contents of the screen and turns it off*/
	public void clearDisplay()
	{
		display = null;
	}
	/**The screen turns back on when calls the setDisplay() operation
	 * @param s - String that will show the GUI
	 * */
	public void setDisplay(String s)
	{
		display = s;
	}
	/**Getter of the display*/
	public String getDisplay()
	{
		return display;
	}
}
```

Heating

47 lines (42 sloc) | 989 Bytes

Raw Blame

```java
package microwave;
/**Magnetron device that emits microwaves, in charge of heat the food to a certain power (<b>power</b>) <br>
 * <br>
 * The microwave turns it on and off using the <b>heating_on()</b> and <b>heating_off()</b> operations <br>
 * You can also know whether it is on or not with the <b>isHeating()</b> query operation <br>
 * The microwave component can also set the power and meet it with the <b>setPower()</b> and <b>getPower()</b> operations <br>
 *
 * @author ap27r
 */
public class Heating
{
	// Variables
	private boolean heating = false;
	private int power = 0;

	/**Setter to true*/
	public void heating_on()
	{
		heating = true;
	}

	/**Setter to false*/
	public void heating_off()
	{
		heating = false;
	}

	/**Setter to value*/
	public void setPower(int power)
	{
		if (power >= 0) // Can't give negative power
		{
			this.power = power;
		}
	}

	// Getters
	public boolean isHeating()
	{
		return heating;
	}

	public int getPower()
	{
		return power;
	}
}
```

Lamp

31 lines (27 sloc) | 532 Bytes

Raw Blame

```
package microwave;

/** Lamp that turns on (<b>lamp_on()</b>) or off (<b>lamp_off()</b>) based on different events, such as: <br>
 * <br>
 * - Door open <br>
 * - Microwave working a query operation
 *
 * @author ap27r
 */
public class Lamp
{
        private boolean lampOn = false;

        /**Setter to true*/
        public void lamp_on()
        {
                lampOn = true;
        }

        /**Setter to false*/
        public void lamp_off()
        {
                lampOn = false;
        }

        /**Allows the microwave to know if the light is on or not*/
        public boolean isLampOn()
        {
                return lampOn;
        }
}
```

Turntable

30 lines (26 sloc) | 745 Bytes

Raw Blame

```
package microwave;

/**Turntable triggered by the <b>turntable_start()</b> operation when the microwave is running and
 * stops (via the turntable_stop() operation) when the door is opened or cooking time is running out.<br>
 * <br>
 * It implements an <b>isMoving()</b> query operation that lets you know where all the time whether the platter is spinning or not.
 * @author ap27r
 * */
public class Turntable
{
        private boolean turntableOn = false;

        /**Setter to true*/
        public void turntable_start()
        {
                turntableOn = true;
        }

        /**Setter to false*/
        public void turntable_stop()
        {
                turntableOn = false;
        }

        /**Allows the microwave to know if the platterplatter is spinning*/
        public boolean isMoving()
        {
                return turntableOn;
        }
}
```

Microwave

205 lines (167 sloc) | 3.26 KB

Raw Blame

```java
package microwave;

/**This class represent a microwave, that might have a set of states, depending of de actions performed on it
 * @author David RA
 * @version 1.0
 * */
public class Microwave
{
        // Variables
        private int power;
        private int timer;
        private boolean doorOpen;
        private boolean cooking;
        private boolean withItem;

        // Variables componentes
        private Heating heatingComponent = new Heating();
        private Lamp lampComponent = new Lamp();
        private Turntable turnableComponent = new Turntable();
        private Beeper beeperComponent = new Beeper();
        private Display displayComponent = new Display();
        /**Class that implement the microwave_interface*/
        private Microwave_Interface state;

        /**Main constructor: creates a microwave closed with no item
         *
         * Must verify that the power and timer are 0
         *                 the door, cook, item variables are false
         *  */
        public Microwave()
        {
                state = new MW_ClosedWithNoItem(this);

                power = 0;
                timer = 0;
                doorOpen = false;
                cooking = false;
                withItem = false;
        }

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

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

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

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

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

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

```

```java

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

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

        /**An external clock is in charge of invoking the tick() operation of the microwave every second, allowing you to know the passage of time*/
        public void tick()
        {
                state.tick(this);
        }

```

```java

        // Not represented in the main UML diagram but necessary

        public int getPower()
        {
                return power;
        }

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

        public int getTime()
        {
                return timer;
        }

        public void setTime(int timer)
        {
                if(timer > 0)
                {
                        this.timer = timer;
                }
                else
                {
                        this.timer = 0; // Not negative numbers in the timer
                }
        }

```

```java

        public boolean isOpen()
        {
                return doorOpen;
        }

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

        public boolean isItem()
        {
                return withItem;
        }

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

        public boolean isCooking()
        {
                return cooking;
        }

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

        public Microwave_Interface getState()
        {
                return state;
        }

        public Heating getHeatComponent()
        {
                return heatingComponent;
        }

        public Lamp getLampComponent()
        {
                return lampComponent;
        }

        public Turntable getTurntComponent()
        {
                return turnableComponent;
        }

        public Beeper getBeepComponent()
        {
                return beeperComponent;
        }

        public Display getDisplayComponent()
        {
                return displayComponent;
        }

        /**Method to change the state automatically -Needed for some tests*/
        public void setState(Microwave_Interface status)
        {
                this.state = status;
        }
}
```

MW_ClosedWithNoItem

119 lines (103 sloc) | 3.14 KB

Raw Blame

```java
package microwave;

/**This class represent a microwave that is closed, with no item inside of it
 * @author David RA
 * @version 1.1
 * */
public class MW_ClosedWithNoItem implements Microwave_Interface
{
        /**The constructor must verify that the lamp, heating, turnable, Cooking and the door are off
         * An must have a item inside
         *  */
        public MW_ClosedWithNoItem(Microwave m)
        {
                m.getLampComponent().lamp_off(); // Check lamp
                m.getHeatComponent().heating_off(); // Check heat
                m.getTurntComponent().turntable_stop(); // Check rotation
                m.setCooking(false); // Check cooking
                m.setOpen(false); // Check door
                m.setItem(false); // Check item
                m.getDisplayComponent().clearDisplay(); // Display restarted, as this is the state created by default in the microwave
        }

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

        @Override
        public void door_closed(Microwave m)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You cant close the door of a already closed microwave");
        }

        @Override
        public void item_placed(Microwave m)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You cant place a item in a closed microwave");
        }

        @Override
        public void item_removed(Microwave m)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You cant remove a item having the door closed");
        }

        @Override
        public void power_inc(Microwave m)
        {
                m.setPower(m.getPower() + 10); // Increment on 10w the power as said in interface
                m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
        }

        @Override
        public void power_dec(Microwave m)
        {
                if (m.getPower() > 0)
                {
                        m.setPower(m.getPower() - 10); // Decrement on 10w the power as said in interface
                        m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
                }
        }

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

```

```java
        @Override
        public void timer_inc(Microwave mw)
        {
                mw.setTime(mw.getTime() + 1);
                mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
        }

        @Override
        public void timer_dec(Microwave m)
        {
                if (m.getTime() > 0)
                {
                        m.setTime(m.getTime() - 1);
                        m.getDisplayComponent().setDisplay(Integer.toString(m.getTime()));
                }
        }

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

        @Override
        public void cooking_start(Microwave m)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You cant start cook with no item inside");
        }

        @Override
        public void cooking_stop(Microwave m)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("Microwave was already stopped");
        }

        @Override
        public void tick(Microwave m)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("Timer must not run when microwave not running");
        }


}
```

MW_OpenWithNoItem

116 lines (101 sloc) | 2.95 KB

Raw Blame

```java
package microwave;

/**This class represent a microwave that is opened, with no item inside of it
 * @author David RA
 * @version 1.1
 * */
public class MW_OpenWithNoItem implements Microwave_Interface
{
        /**The constructor must verify that the heating, turnable, Cooking and the door are off
         * Must have a item inside and lamp turned on
         *  */
        public MW_OpenWithNoItem(Microwave mw)
        {
                mw.setCooking(false); // Check cooking
                mw.setItem(false); // Check item
                mw.setOpen(true); // Check door

                mw.getLampComponent().lamp_on(); // Check lamp
                mw.getHeatComponent().heating_off(); // Check heat
                mw.getTurntComponent().turntable_stop(); // Check rotation
        }

        @Override
        public void door_opened(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You cant open a door in a opened microwave");
        }

        @Override
        public void door_closed(Microwave mw)
        {
                mw.setState(new MW_ClosedWithNoItem(mw));
        }

        @Override
        public void item_placed(Microwave mw)
        {
                mw.setState(new MW_OpenWithItem(mw));
        }

        @Override
        public void item_removed(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("There is no item to be removed in the microwave");
        }

        @Override
        public void power_inc(Microwave m)
        {
                m.setPower(m.getPower() + 10); // Increment on 10w the power as said in interface
                m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
        }

        @Override
        public void power_dec(Microwave mw)
        {
                if (mw.getPower() > 0)
                {
                        mw.setPower(mw.getPower() - 10); // Decrement on 10w the power as said in interface
                        mw.getDisplayComponent().setDisplay(Integer.toString(mw.getPower()));
                }
        }

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

```

```java
        @Override
        public void timer_inc(Microwave mw)
        {
                mw.setTime(mw.getTime() + 1);
                mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
        }

        @Override
        public void timer_dec(Microwave mw)
        {
                if (mw.getTime() > 0)
                {
                        mw.setTime(mw.getTime() - 1);
                        mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
                }
        }

        @Override
        public void timer_reset(Microwave mw)
        {
                mw.setTime(0);
        }

        @Override
        public void cooking_start(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You must no be able to start the microwave with the door oppened");
        }

        @Override
        public void cooking_stop(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("Microwave was stopped before");
        }

        @Override
        public void tick(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("Timer must not run when microwave not running");
        }
}
```

MW_OpenWithItem

115 lines (101 sloc) | 2.94 KB

Raw Blame

```java
package microwave;

/**This class represent a microwave that is opened, with a item inside of it
 * @author David RA
 * @version 1.1
 * */
public class MW_OpenWithItem implements Microwave_Interface
{
        /**The constructor must verify that the heating, turnable, Cooking and the door are off
         * Must have a item inside and lamp turned on
         *  */
        public MW_OpenWithItem(Microwave mw)
        {
                mw.setCooking(false); // Check cooking
                mw.setItem(true); // Check item
                mw.setOpen(true); // Check door
                mw.getLampComponent().lamp_on(); // Check lamp
                mw.getHeatComponent().heating_off(); // Check heat
                mw.getTurntComponent().turntable_stop(); // Check rotation
        }

        @Override
        public void door_opened(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You cant open a door in a opened microwave");
        }

        @Override
        public void door_closed(Microwave mw)
        {
                mw.setState(new MW_ClosedWithItem(mw));
        }

        @Override
        public void item_placed(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You cant place a item in a microwave that have some item inside");
        }

        @Override
        public void item_removed(Microwave mw)
        {
                mw.setState(new MW_OpenWithNoItem(mw));
        }

        @Override
        public void power_inc(Microwave m)
        {
                m.setPower(m.getPower() + 10); // Increment on 10w the power as said in interface
                m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
        }

        @Override
        public void power_dec(Microwave m)
        {
                if (m.getPower() > 0)
                {
                        m.setPower(m.getPower() - 10); // Decrement on 10w the power as said in interface
                        m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
                }
        }

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

```

```
72          @Override
73          public void timer_inc(Microwave mw)
74          {
75                  mw.setTime(mw.getTime() + 1);
76                  mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
77          }
78
79          @Override
80          public void timer_dec(Microwave m)
81          {
82                  if (m.getTime() > 0)
83                  {
84                          m.setTime(m.getTime() - 1);
85                          m.getDisplayComponent().setDisplay(Integer.toString(m.getTime()));
86                  }
87          }
88
89          @Override
90          public void timer_reset(Microwave mw)
91          {
92                  mw.setTime(0);
93          }
94
95          @Override
96          public void cooking_start(Microwave mw)
97          {
98                  // Exception would be manage - Message written in English don't show to the users
99                  throw new IllegalStateException("You must no be able to start the microwave with the door oppened");
100         }
101
102         @Override
103         public void cooking_stop(Microwave mw)
104         {
105                 // Exception would be manage - Message written in English don't show to the users
106                 throw new IllegalStateException("Microwave was stopped before");
107         }
108
109         @Override
110         public void tick(Microwave m)
111         {
112                 // Exception would be manage - Message written in English don't show to the users
113                 throw new IllegalStateException("Timer must not run when microwave not running");
114         }
115 }
```

MW_ClosedWithItem

139 lines (123 sloc) | 3.67 KB

Raw Blame

```java
package microwave;

/**This class represent a microwave that is closed, with a item inside of it
 * @author David RA
 * @version 1.1
 * */
public class MW_ClosedWithItem implements Microwave_Interface
{
        /**The constructor must verify that the lamp, heating, turnable, Cooking and the door are off
         * An must have a item inside
         *  */
        public MW_ClosedWithItem(Microwave mw)
        {
                mw.getLampComponent().lamp_off(); // Check lamp
                mw.getHeatComponent().heating_off(); // Check heat
                mw.getTurntComponent().turntable_stop(); // Check rotation
                mw.setCooking(false); // Check cooking
                mw.setOpen(false); // Check door
                mw.setItem(true); // Check item
        }

        @Override
        public void door_opened(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                mw.setState(new MW_OpenWithItem(mw));
        }

        @Override
        public void door_closed(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You cant close the door of a already closed microwave");
        }

        @Override
        public void item_placed(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You cant place a item in a closed microwave");
        }

        @Override
        public void item_removed(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You cant remove a item having the door closed");
        }

        @Override
        public void power_inc(Microwave mw)
        {
                mw.setPower(mw.getPower() + 10); // Increment on 10w the power as said in interface
                mw.getDisplayComponent().setDisplay(Integer.toString(mw.getPower()));
        }

        @Override
        public void power_dec(Microwave mw)
        {
                if (mw.getPower() > 0)
                {
                        mw.setPower(mw.getPower() - 10); // Decrement on 10w the power as said in interface
                        mw.getDisplayComponent().setDisplay(Integer.toString(mw.getPower()));
                }
        }

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

```

```java
        @Override
        public void timer_inc(Microwave mw)
        {
                mw.setTime(mw.getTime() + 1);
                mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
        }

        @Override
        public void timer_dec(Microwave mw)
        {
                if (mw.getTime() > 0)
                {
                        mw.setTime(mw.getTime() - 1);
                        mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
                }
        }

        @Override
        public void timer_reset(Microwave mw)
        {
                mw.setTime(0);
        }

        @Override
        public void cooking_start(Microwave mw)
        {
                if(!(mw.getTime() > 0) && !(mw.getPower() > 0))
                {
                        // Exception would be manage - Message written in English don't show to the users
                        throw new IllegalStateException("You cant start cook with no time neither power");
                }

                if(mw.getTime() > 0 && mw.getPower() > 0)
                {
                        mw.setState(new MW_Cooking(mw));
                }
                else
                {
                        if(mw.getTime() > 0)
                        {
                                // Exception would be manage - Message written in English don't show to the users
                                throw new IllegalStateException("You cant start cook with no time");
                        }
                        else
                        {
                                // Exception would be manage - Message written in English don't show to the users
                                throw new IllegalStateException("You cant start cook with no power");
                        }
                }
        }

        @Override
        public void cooking_stop(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("Microwave was already stopped");
        }

        @Override
        public void tick(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("Timer must not run when microwave not running");
        }

}
```

MW_Cooking

148 lines (131 sloc) | 3.52 KB

Raw Blame

```java
package microwave;

/**This class represent a microwave that is cooking (so have a item, ligth on, rotating ...)
 * @author David RA
 * @version 1.2
 * */
public class MW_Cooking implements Microwave_Interface
{
        /**The constructor must verify that the lamp, heating, turnable, Cooking are on
         * Must have a item inside and the door closed
         *  */
        public MW_Cooking(Microwave mw)
        {
                mw.setCooking(true); // Check cooking
                mw.setOpen(false);  // Check door
                mw.setItem(true); // Check item

                mw.getHeatComponent().setPower(mw.getPower()); // Set power value saved in the microwave variable

                mw.getLampComponent().lamp_on(); // Check lamp
                mw.getHeatComponent().heating_on(); // Check heat
                mw.getTurntComponent().turntable_start(); // Check rotation
        }

        @Override
        public void door_opened(Microwave mw)
        {
                mw.setState(new MW_OpenWithItem(mw));
        }

        @Override
        public void door_closed(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You cant close the door of a already closed microwave");
        }

        @Override
        public void item_placed(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You cant place a item in a closed microwave");
        }

        @Override
        public void item_removed(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You cant remove a item having the door closed");
        }

        @Override
        public void power_inc(Microwave m)
        {
                m.setPower(m.getPower() + 10);  // Increment on 10w the power as said in interface
                m.getDisplayComponent().setDisplay(Integer.toString(m.getPower()));
        }

        @Override
        public void power_dec(Microwave mw)
        {
                if (mw.getPower() > 0)
                {
                        mw.setPower(mw.getPower() - 10);  // Decrement on 10w the power as said in interface
                        mw.getDisplayComponent().setDisplay(Integer.toString(mw.getPower()));
                }
                if(mw.getPower() == 0)
                {
                        cooking_stop(mw);
                }
        }

```

```java
        @Override
        public void power_reset(Microwave mw)
        {
                mw.setState(new MW_ClosedWithItem(mw));
                mw.setPower(0);
                mw.getDisplayComponent().setDisplay(Integer.toString(mw.getPower()));
        }

        @Override
        public void timer_inc(Microwave mw)
        {
                mw.setTime(mw.getTime() + 1);
                mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
        }

        @Override
        public void timer_dec(Microwave mw)
        {
                if (mw.getTime() > 0)
                {
                        mw.setTime(mw.getTime() - 1);
                        mw.getDisplayComponent().setDisplay(Integer.toString(mw.getTime()));
                }
                if (mw.getTime() == 0)
                {
                        mw.getBeepComponent().beep(3);
                        mw.getDisplayComponent().setDisplay("Item ready");
                        cooking_stop(mw);
                }
        }

        @Override
        public void timer_reset(Microwave mw)
        {
                mw.setState(new MW_ClosedWithItem(mw));
                mw.setTime(0);
        }


        @Override
        public void cooking_start(Microwave mw)
        {
                // Exception would be manage - Message written in English don't show to the users
                throw new IllegalStateException("You cant start cooking if the microwave was already cooking ...");
        }

        @Override
        public void cooking_stop(Microwave mw)
        {
                mw.setState(new MW_ClosedWithItem(mw));
        }

        @Override
        public void tick(Microwave mw)
        {
                if (mw.getTime() > 1)
                {
                        mw.timer_dec();
                        try
                        {
                                Thread.sleep(1); // TODO Implement real tick in the GUI
                        }
                        catch (InterruptedException e)
                        {
                                e.printStackTrace();
                        }
                }
                else
                {
                        mw.timer_dec();
                        mw.getBeepComponent().beep(3);
                        mw.getDisplayComponent().setDisplay("Item ready");
                        cooking_stop(mw);
                }
        }
}
```

Apartado B

Definir pruebas unitarias con Junit para cada uno de los componentes que conforman el sistema;

Mi implementación de los tests con Junit está en la clase **MicrowaveTest**

360 lines (296 sloc) | 8.82 KB

Raw Blame

```java
package microwave_Junit_Test;

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

import microwave.*;

public class MicrowaveTest
{
        private Microwave mw = new Microwave();

        //COMPONENTS TESTS

        // Heating component test
        @Test
        public void MagnetronTest()
        {
                Heating h = new Heating();

                // Base
                assertEquals(0, h.getPower());
                assertEquals(false,h.isHeating());

                // On - Off
                h.heating_on();
                assertEquals(true,h.isHeating());
                h.heating_off();
                assertEquals(false,h.isHeating());

                // Increase Power
                h.setPower(888);
                assertEquals(h.getPower(), 888);
                h.setPower(0);
                assertEquals(h.getPower(), 0);

        }

        // Lamp component test
        @Test
        public void lumixTest()
        {
                Lamp l = new Lamp();

                // Base
                assertFalse(l.isLampOn());

                // On - Off
                assertFalse(l.isLampOn());
                l.lamp_on();
                assertTrue(l.isLampOn());
                l.lamp_off();
                assertFalse(l.isLampOn());
        }
```

```
54
55          // Turntable component test
56          @Test
57          public void rotationTest()
58          {
59                  Turntable t = new Turntable();
60
61                  // Base
62                  assertFalse(t.isMoving());
63
64                  // Start - Stop
65                  t.turntable_start();
66                  assertTrue(t.isMoving());
67                  t.turntable_stop();
68                  assertFalse(t.isMoving());
69          }
70
71          // Beeper component test
72          @Test
73          public void beeperTest()
74          {
75                  Beeper b = new Beeper();
76
77                  // Beep
78                  b.beep(5);
79                  assertTrue(BeeperCounter.isBeeped(5));
80
81                  // No Beep
82                  assertTrue(BeeperCounter.isBeeped(0));
83          }
84
85          // Display component test
86          @Test
87          public void displayTest()
88          {
89                  Display d = new Display();
90
91                  // Base
92                  assertNull(d.getDisplay());
93
94                  // Set - Clear
95                  d.setDisplay("Test");
96                  assertEquals("Test", d.getDisplay());
97                  d.clearDisplay();
98                  assertNull(d.getDisplay());
99          }
100
101         /** Metod to simulate the increase of the microwave power
102          * @param time - Amount of time to increase (Works fine with mult of 10)
103          * */
104         private void increment_power(int p)
105         {
106                 for (int i = 0; i < p; i = i + 10) // The increment is 10 by 10
107                 {
108                         mw.power_inc();
109                 }
110         }
111
```

```
112         /** Metod to simulate the decrease of the microwave power
113          *  @param time - Amount of time to decrease (Works fine with mult of 10)
114          * */
115         private void decrease_power(int p)
116         {
117                 for (int i = p; i > 0; i = i - 10) // The decrement is 10 by 10
118                 {
119                         mw.power_dec();
120                 }
121         }
122
123         /** Metod to simulate the action of increment the time remaining
124          * @param time - Amount of time to increase
125          * */
126         private void timer_inc(int time)
127         {
128                 for (int i = 0; i < time; i++)
129                 {
130                         mw.timer_inc();
131                 }
132         }
133
134         /** Metod to simulate the action of decrease the time remaining
135          * @param time - Amount of time to decrease
136          * */
137         private void timer_dec(int time)
138         {
139                 for (int i = 0; i < time; i++)
140                 {
141                         mw.timer_dec();
142                 }
143         }
144
145         /** Metod to simulate the action of time
146          * @param time - Amount of time passed
147          * */
148         private void timer_works(int time)
149         {
150                 for (int i = 0; i < time; i++)
151                 {
152                         mw.tick();
153                 }
154         }
155
```

```
156         // MICROWAVE TESTS
157
158         // Test for power
159         @Test
160         public void testPower()
161         {
162                 // Start
163                 mw.power_reset();
164                 assertEquals(0, mw.getPower());
165
166                 // Increase - Decrease
167                 increment_power(80);
168                 assertEquals(mw.getPower(), 80);
169                 assertEquals(mw.getDisplayComponent().getDisplay(),"80");
170                 decrease_power(100);
171                 assertEquals(mw.getPower(), 0);
172
173                 // End
174                 mw.power_reset();
175                 assertEquals(mw.getPower(), 0);
176                 assertEquals(mw.getDisplayComponent().getDisplay(), "0");
177         }
178
```

```java
179         // Test for timer
180         @Test
181         public void testTimer()
182         {
183                 // Start
184                 mw.timer_reset();
185                 assertEquals(0, mw.getTime());
186
187                 // Increase - Decrease
188                 timer_inc(80);
189                 assertEquals(80, mw.getTime());
190                 assertEquals("80", mw.getDisplayComponent().getDisplay());
191                 timer_dec(35);
192                 assertEquals(45, mw.getTime());
193                 assertEquals("45", mw.getDisplayComponent().getDisplay());
194
195                 // End
196                 mw.timer_reset();
197                 assertEquals(mw.getTime(), 0);
198                 assertEquals(mw.getDisplayComponent().getDisplay(), "0");
199         }
200
201         // State 1: ClosedWithNoItem
202         @Test
203         public void closedWithNoItemTest()
204         {
205                 mw.setState(new MW_ClosedWithNoItem(mw));
206                 // Exceptions check
207                 assertThrows(IllegalStateException.class, () -> mw.item_placed());
208                 assertThrows(IllegalStateException.class, () -> mw.item_removed());
209                 assertThrows(IllegalStateException.class, () -> mw.door_closed());
210                 assertThrows(IllegalStateException.class, () -> mw.cooking_start());
211                 assertThrows(IllegalStateException.class, () -> mw.cooking_stop());
212
213
214                 // Status check
215                 assertEquals(false,mw.isCooking());
216                 assertEquals(false,mw.isItem());
217                 assertEquals(false,mw.isOpen());
218                 assertEquals(false,mw.getHeatComponent().isHeating());
219                 assertEquals(false,mw.getLampComponent().isLampOn());
220                 assertEquals(false,mw.getTurntComponent().isMoving());
221                 assertEquals(true,mw.getState() instanceof MW_ClosedWithNoItem);
222         }
```

```java
223
224         // State 2: OpenWithNoItem
225         @Test
226         public void openWithNoItemTest()
227         {
228                 mw.setState(new MW_OpenWithNoItem(mw));
229
230                 // Exceptions check
231                 assertThrows(IllegalStateException.class, () -> mw.item_removed());
232                 assertThrows(IllegalStateException.class, () -> mw.cooking_start());
233                 assertThrows(IllegalStateException.class, () -> mw.door_opened());
234                 assertThrows(IllegalStateException.class, () -> mw.cooking_stop());
235
236
237                 // Status check
238                 assertEquals(false,mw.isCooking());
239                 assertEquals(false,mw.isItem());
240                 assertEquals(true,mw.isOpen());
241                 assertEquals(false,mw.getHeatComponent().isHeating());
242                 assertEquals(true,mw.getLampComponent().isLampOn());
243                 assertEquals(false,mw.getTurntComponent().isMoving());
244                 assertEquals(true,mw.getState() instanceof MW_OpenWithNoItem);
245         }
246
```

```java
247         // Phase 3: Test for an OpenWithItem situation
248         @Test
249         public void openWithItemTest()
250         {
251                 mw.setState(new MW_OpenWithItem(mw));
252
253                 // Exceptions check
254                 assertThrows(IllegalStateException.class, () -> mw.item_placed());
255                 assertThrows(IllegalStateException.class, () -> mw.cooking_start());
256                 assertThrows(IllegalStateException.class, () -> mw.door_opened());
257                 assertThrows(IllegalStateException.class, () -> mw.cooking_stop());
258
259
260                 // Status check
261                 assertEquals(false,mw.isCooking());
262                 assertEquals(true,mw.isItem());
263                 assertEquals(true,mw.isOpen());
264                 assertEquals(false,mw.getHeatComponent().isHeating());
265                 assertEquals(true,mw.getLampComponent().isLampOn());
266                 assertEquals(false,mw.getTurntComponent().isMoving());
267                 assertEquals(true,mw.getState() instanceof MW_OpenWithItem);
268
269                 // Removing item
270                 mw.item_removed();
271                 assertEquals(mw.getState().getClass(),MW_OpenWithNoItem.class);
272
273         }
274
275         // Phase 4: Test for a ClosedWithItem situation
276         @Test
277         public void closedWithItemTest()
278         {
279                 mw.setState(new MW_ClosedWithItem(mw));
280
281                 // Exceptions check
282                 assertThrows(IllegalStateException.class, () -> mw.item_placed());
283                 assertThrows(IllegalStateException.class, () -> mw.item_removed());
284                 assertThrows(IllegalStateException.class, () -> mw.door_closed());
285                 assertThrows(IllegalStateException.class, () -> mw.cooking_stop());
286
287
288                 // Status check
289                 assertEquals(false,mw.isCooking());
290                 assertEquals(true,mw.isItem());
291                 assertEquals(false,mw.isOpen());
292                 assertEquals(false,mw.getHeatComponent().isHeating());
293                 assertEquals(false,mw.getLampComponent().isLampOn());
294                 assertEquals(false,mw.getTurntComponent().isMoving());
295                 assertEquals(true,mw.getState() instanceof MW_ClosedWithItem);
296
297                 // Opening door
298                 mw.door_opened();
299                 assertEquals(mw.getState().getClass(),MW_OpenWithItem.class);
300         }
301
```

```java
        // Phase 5: Test for a Cooking situation
        @Test
        public void cookingTest()
        {
                mw.setState(new MW_Cooking(mw));

                // Cooking with wrong inputs

                // time == 0  &  power == 0
                mw.timer_reset();
                assertThrows(IllegalStateException.class, () -> mw.cooking_start());

                // time == 60  &  power == 0
                timer_inc(60);
                assertThrows(IllegalStateException.class, () -> mw.cooking_start());

                // time == 0  &  power == 800
                mw.timer_reset();
                increment_power(800);
                assertThrows(IllegalStateException.class, () -> mw.cooking_start());

                // Start cooking
                timer_inc(25);
                increment_power(100);
                mw.cooking_start();

                // Exceptions check
                assertThrows(IllegalStateException.class, () -> mw.cooking_start());
                assertThrows(IllegalStateException.class, () -> mw.door_closed());

                assertThrows(IllegalStateException.class, () -> mw.item_placed());
                assertThrows(IllegalStateException.class, () -> mw.item_removed());

                // Status check
                assertEquals(mw.isCooking(),true);
                assertEquals(mw.isItem(),true);
                assertEquals(mw.isOpen(),false);
                assertEquals(mw.getHeatComponent().isHeating(),true);
                assertEquals(mw.getLampComponent().isLampOn(),true);
                assertEquals(mw.getTurntComponent().isMoving(),true);
                assertEquals(mw.getState() instanceof MW_Cooking,true);

                // Open door
                mw.door_opened();
                assertTrue(mw.getState() instanceof MW_OpenWithItem);
                assertFalse(BeeperCounter.isBeeped(3));

                mw.door_closed();
                mw.cooking_start();

                assertEquals(25, mw.getTime());
                assertEquals(mw.getState().getClass(),MW_Cooking.class);

                // Time ends
                timer_works(25);
                assertEquals("Item ready", mw.getDisplayComponent().getDisplay());
                assertTrue(BeeperCounter.isBeeped(3));
        }
}
```

Apartado C

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

Yo he decidido crear una feature por cada estado del microondas, y así poder aislar las distintas etapas para poder realizadas las pruebas de forma más ordenada. Están en la ruta resources de la parte de test del src (**Enlace** a la carpeta que los contienen en GitHub)

MW_ClosedWithNoItem.feature

23 lines (19 sloc) | 476 Bytes

```
Feature: Testing first State

  Scenario Outline: Setting power
    Given Testing first State
    When Set power to <T>
    Then Screen value is "<D>"

    Examples:
      |  T  |  D  |
      |  10 | 100 |
      |   0 |   0 |
      |  -1 |   0 |

  Scenario Outline: Setting timer
    Given Testing first State
    When We settle timer at <T> s
    Then Screen value is "<D>"

    Examples:
      |  T  |  D  |
      |-888 |   0 |
      |   0 |   0 |
      |  88 |  88 |
```

MW_OpenWithNoItem.feature

23 lines (18 sloc) | 459 Bytes

```
Feature: Testing second State

  Scenario Outline: Setting power
    Given Testing second State
    When Set power to <a>
    Then Screen value is "<b>"

    Examples:
      | a   | b   |
      |   0 |   0 |
      |   8 |  80 |


  Scenario Outline: Setting timer
    Given Testing second State
                When We settle timer at <a> s
    Then Screen value is "<b>"

    Examples:
      | a   | b   |
      |  -1 |   0 |
      |   0 |   0 |
      |  60 |  60 |
```

MW_OpenWithItem.feature

22 lines (18 sloc) | 452 Bytes

Raw Blame

```
Feature: Testing third State

  Scenario Outline: Setting power
    Given Testing third State
    When Set power to <a>
    Then Screen value is "<b>"

    Examples:
      | a   | b   |
      |   1 |  10 |
      |   0 |   0 |

  Scenario Outline: Setting timer
    Given Testing third State
    When We settle timer at <a> s
    Then Screen value is "<b>"

    Examples:
      | a   | b   |
      |  -1 |   0 |
      |   0 |   0 |
      |  60 |  60 |
```

MW_ClosedWithItem.feature

30 lines (25 sloc) | 652 Bytes

Raw Blame

```
Feature: Testing fourth State

  Scenario: Cook
    Given Testing fourth State
    When Set power to 10
    And We settle timer at 90 s
    And Press start cooking
    Then We are in the fifth State

  Scenario Outline: Setting power
    Given Testing fourth State
    When Set power to <T>
    Then Screen value is "<D>"

    Examples:
      |  T  |  D  |
      |  10 | 100 |
      |   0 |   0 |
      |  -1 |   0 |

  Scenario Outline: Setting timer
    Given Testing fourth State
    When We settle timer at <S> s
    Then Screen value is "<D>"

    Examples:
      |  S  |  D  |
      |  88 |  88 |
      |   0 |   0 |
      |-888 |   0 |
```

MW_Cooking.feature

38 lines (32 sloc) | 1.05 KB

Raw Blame

```
Feature: Testing fifth State

  Scenario: Cook correctly
    Given Testing fourth State
    When Increase the power 10 times
    And We settle timer at 60 s
    And Press start cooking
    Then We are in the fifth State
    And Heating heats
    And Lamp turns on
    And Turntable turns

  Scenario: Testing change while runing 1
    Given  A running microwave with 2000 power and 60 timer
    When Increase the power 1 times
    Then Screen value is "2010"

  Scenario: Testing change while runing 2
    Given  A running microwave with 800 power and 1 timer
    When We settle timer at 2 s
    Then Screen value is "2"

  Scenario: Testing change while runing 3
    Given  A running microwave with 800 power and 2 timer
    When We settle timer at 1 s
    Then Screen value is "1"

  Scenario Outline: Testing Display
    Given Testing fourth State
    When We settle timer at <S> s
    And Increase the power <T> times
    Then Screen value is "<D>"

    Examples:
      |  S  | T  |  D  |
      | 15  | 63 | 630 |
      | 30  | 75 | 750 |
      | 100 | 89 | 890 |
```

StepDefinitions

Aquí he definido cada uno de los métodos que se invocarán con los escenarios (**Enlace** a GitHub del archivo)

120 lines (99 sloc) | 2.29 KB

Raw Blame

```java
package microwave_Cucumber_Test;

import io.cucumber.java.en.*;

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

import microwave.*;

/**Steps definition stores the mapping between each step of the scenario defined in the feature file with a code of function to be executed*/
public class StepDefinitions
{
        /**Our microwave instance that we are gona use for test*/
        private Microwave mw = new Microwave();

        // GIVEN

        @Given("Testing first State")
        public void first_State()
        {
                mw.setState(new MW_ClosedWithNoItem(mw));
        }

        @Given("Testing second State")
        public void second_State()
        {
                mw.setState(new MW_OpenWithNoItem(mw));
        }

        @Given("Testing third State")
        public void third_State()
        {
                mw.setState(new MW_OpenWithItem(mw));
        }

        @Given("Testing fourth State")
        public void fourth_State()
        {
                mw.setState(new MW_ClosedWithItem(mw));
        }

        @Given("A running microwave with {int} power and {int} timer")
        public void cooking(Integer power, Integer time)
        {
                mw.setState(new MW_ClosedWithItem(mw));
                mw.setTime(time);
                set_Power(power/10);
                mw.cooking_start();
        }

```

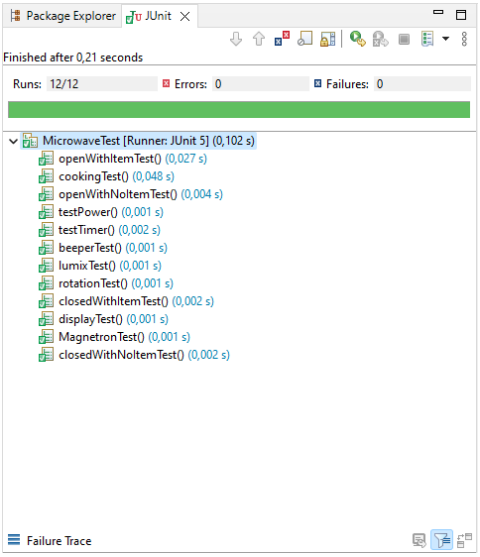
```java
        // WHEN

        @When("Set power to {int}")
        public void set_Power(Integer new_power)
        {
                mw.power_reset();
                increase_Power(new_power);
        }

        @When("Increase the power {int} times")
        public void increase_Power(Integer simulated_times)
        {
                for (int i = 0; i < simulated_times; i++)
                {
                        mw.power_inc();
                }
        }

        @When("We settle timer at {int} s")
        public void setTimer(Integer times)
        {
                mw.setTime(times);
                mw.timer_inc();
                mw.timer_dec();
        }

        @When("Press start cooking")
        public void try_to_cook()
        {
                try
                {
                        mw.cooking_start();
                }
                catch (IllegalStateException e)
                {
                        assertEquals(false, true);
                }
        }

        // THEN

        @Then("Heating heats")
        public void magnetronWorking()
        {
                assertEquals(mw.getHeatComponent().isHeating(), true);
        }

        @Then("Lamp turns on")
        public void lampIsOn()
        {
                assertEquals(mw.getLampComponent().isLampOn(), true);
        }

        @Then("Turntable turns")
        public void turntableIsTurnning()
        {
                assertEquals(mw.getTurntComponent().isMoving(), true);
        }

        @Then("Screen value is {string}")
        public void screenShows(String i)
        {
                assertEquals(mw.getDisplayComponent().getDisplay(), i);
        }

        @Then("We are in the fifth State")
        public void microwaveCooking()
        {
                assertEquals(mw.getState().getClass(), MW_Cooking.class);
        }
}
```

Apartado D

Definir e implementar tres interfaces de usuario que, a través de botones, permitan interactuar con el microondas de forma concurrente: uno con el panel de control, otro que simule la puerta y el hecho de meter y sacar un alimento del microondas, y un tercero que permita simular el tick de reloj.

Resultados

Test con Junit



Test con Cucumber

```
[INFO] Tests run: 30, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.763 s - in microwave_Cucumber_Test.RunCucumberTest
[INFO] Running microwave_Junit_Test.MicrowaveTest
[INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.153 s - in microwave_Junit_Test.MicrowaveTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 42, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  6.324 s
[INFO] Finished at: 2022-06-10T21:27:13+02:00
[INFO] ------------------------------------------------------------------------
```

Conclusión

El patrón de diseño Estado se adapta muy bien para este proyecto (ya que el comportamiento del microondas cambia dependiendo de su estado) y gracias a las pruebas realizadas con Gherkin-Cucumber en este proyecto me han quedado más claro en qué situaciones se pueden usar estas herramientas.