PRACTICA MICROONDAS

Cubillos Del Toro, David

Ingeniería del software avanzada

1. Implementación del sistema en java

Clase Microwave

```
package microwave;
public class Microwave {
    protected boolean doorOpen, cooking , withItem;
    protected int power, timer;
    protected Estado_Microondas estado;
    protected Heating calor;
    protected Lamp lampara;
    protected Turntable gira;
    protected Beeper alarma;
    protected Display pantalla;

    /*
      * Constructor
      * Crea un microondas cerrado sin ningun objeto dentro,
      * la base del sistema.
      */
    public Microwave () {

        this.estado= new Estado_ClosedWithNoItem();
        this.doorOpen= false ;
        this.cooking= false;
        this.withItem= false ;
        this.power= 0;
        this.timer= 0;
        this.alarma= new Beeper();
        this.pantalla= new Display();
        this.calor= new Heating();
        calor.heating_off();
        calor.SetPower(0);
        this.lampara=new Lamp();
        this.lampara.lamp_off();
        this.cooking=false;
        this.doorOpen=false;
        this.gira= new Turntable();
        gira.turntable_stop();
    }
    public boolean isDoorOpen() {
        return doorOpen;
    }
    public boolean isCooking() {
        return cooking;
    }
    public boolean isWithItem() {
        return withItem;
    }
    public int getPower() {
        return power;
    }
    public int getTimer() {
        return timer;
    }
    public Estado_Microondas getEstado() {
        return estado;
    }
    public Heating getCalor() {
        return calor;
    }
    public Lamp getLampara() {
        return lampara;
    }
    public Turntable getGira() {
        return gira;
    }
    public Beeper getAlarma() {
        return alarma;
    }
    public Display getPantalla() {
        return pantalla;
    }
    private void  Microwave(Microwave m){
        this.estado= m.estado;
        this.doorOpen= m.doorOpen;
        this.cooking= m.cooking;
        this.withItem= m.withItem;
        this.power= m.power;
        this.timer= m.timer;
        this.alarma= m.alarma;
        this.calor= m.calor;
        this.cooking=m.cooking;
        this.doorOpen=m.doorOpen;
        this.gira= m.gira;
        this.lampara=m.lampara;
        this.pantalla= m.pantalla;
    }
    public void door_closed() {
        Microwave(estado.door_closed(this));
    }
    public void door_opened() {
        this.Microwave(estado.door_opened(this));
    }
    public void item_placed() {
        this.Microwave(estado.item_placed(this));
    }
    public void item_removed() {
        this.Microwave(estado.item_removed(this));
    }
    public void power_inc () {
        if (this.power<5)this.power+=1;
    }
    public void power_dec() {
        if (this.power>0) {this.power-=1;}
    }
    public void power_reset() {
        if (this.power!=0) {this.power=0;}
    }
    public void timer_inc() {
        this.Microwave(estado.timer_inc(this));
    }
    public void timer_dec() {
        this.Microwave(estado.timer_dec(this));
    }
    public void timer_reset() {
        this.Microwave(estado.timer_reset(this));
    }
    public void cooking_start() {
        this.Microwave(estado.cooking_start(this));
    }
    public void cooking_stop() {
        this.Microwave(estado.cooking_stop(this));
    }
    public void tick() {
        this.Microwave(estado.tick(this));
    }
    public void setTimer(int timer) {
        if (timer>=0 && timer <301) {
            this.Microwave(estado.setTimer(this,timer));
        } else {
            this.pantalla.setDisplay("Solo valores entre 0 y 300");
        }

    }
}
```

La clase microondas guarda los atributos indicados en el UML para poder realizar las operaciones.

- **void Microwave(Microwave m)**:
  Se ha implementado un método llamado como la propia clase, que iguala las referencias del objeto con los del microondas que recibe como parámetro.
- Un **constructor** que inicia un microondas cerrado sin ítem con todos los valores al mínimo/apagado.
- Una serie de **getters** para verificar el valor de los atributos en los ficheros de pruebas.
- En el **resto de los métodos** se llama a su método homónimo del estado almacenado; es decir, tomando el ejemplo de door_closed: invoca al método llamado de la misma manera del estado (pasándole el objeto actual).Después llama a la función Microwave().

Clase Estado_Microondas

Para la implementación del patrón estado, se ha escogido una **clase abstracta** como super-clase de los escenarios. Se ha escogido una clase abstracta por la existencia de funciones que no realizan ninguna acción, como por ejemplo sacar el ítem cuando no esta abierta la puerta.
Imaginemos que el estado de un microondas es **cerrado con ítem** y se le pasa el método **ítem_removed()**, la cascada de ejecución continuará llamando a la misma función en el estado, pero pasándole el microondas como parámetro.
Sin embargo la clase cerrado con ítem (closedWithNoItem), como veremos a continuación, no guarda la implementación de ningún **ítem_removed()**, si no que utiliza la implementación vacua de la clase padre. Así solo deberemos rectificar con *@Override* el comportamiento que queramos que difiera de la clase padre.

```
package microwave;

public abstract class Estado_Microondas {

    protected enum state{OpenWithNoItem,ClosedWithNoItem,
        OpenWithItem,ClosedWithItem,Cooking}
    state fase;

    protected Microwave door_closed (Microwave m) {

        return m;
    }
    protected Microwave door_opened(Microwave m) {

        return m;
    }
    protected Microwave item_placed (Microwave m) {

        return m;
    }
    protected Microwave item_removed(Microwave m) {
        return m;
    }
    protected Microwave power_inc(Microwave m) {
        if (!(m.power>4)) {
            m.power+=1;
        }
        return m;
    }
    protected Microwave power_dec(Microwave m) {
        if (m.power>0) {
            m.power-=1;
        }
        return m;
    }
    protected Microwave power_reset(Microwave m) {
        m.power=0;
        return m;
    }
    protected Microwave timer_inc(Microwave m) {
        if(!(m.timer>299)) {
            m.timer+=1;
        }
        return m;
    }
    protected Microwave timer_dec(Microwave m) {
        if(m.timer>0) {
            m.timer+=1;
        }
        return m;
    }
    protected Microwave timer_reset(Microwave m) {
        m.timer=0;
        return m;
    }
    protected Microwave cooking_start(Microwave m) {
        return m;
    }
    protected Microwave cooking_stop(Microwave m) {
        return m;
    }
    protected Microwave tick(Microwave m) {
        return m;
    }
    protected Microwave setTimer(Microwave m,int i) {
        m.timer=i;
        return m;
    }
    public String fase() {

        return this.fase.toString();
    }
```

Principales características:

- Se ha declarado un **enum** para poder etiquetar fácilmente los estados y getter para obtener una versión de tipo string.
- Existen **métodos vacíos**, que devuelven el microondas que reciben intacto estos métodos vacíos son aquellos que no hacen cambiar el estado del sistema.
- Otros métodos como los **incrementadores** de potencia o de tiempo tienen una implementación que heredaran todas las subclases.

A continuación se mostraran todas las subclases derivadas de Estado_microondas:
Los constructores de todos los escenarios propuestos solo inician el enum correspondiente al estado.

Estado_ClosedWithNoItem

Este escenario tiene solo una función que lo haga cambiar de estado:

- **door_opened:** Realiza las operaciones correspondientes a abrir la puerta cambiando el estado, poniendo lampon= true y doorOpen= true

```
package microwave;

public class Estado_ClosedWithNoItem extends Estado_Microondas {
      // todas las operaciones basicas de la clase microondas
    public Estado_ClosedWithNoItem () {
        fase= state.ClosedWithNoItem;
        }
    @Override
    public Microwave door_opened (Microwave m) {
        // cambio de estado y cambio de micro
        m.estado= new Estado_OpenWithNoItem();
        m.lampara.lamp_on();
        m.doorOpen= true;
        return m;
    }
}
```

Estado_OpenWithNoItem

Cuando un microondas se encuentra en este estado, solo hay dos operaciones que le hacen cambiar de estado:

- **door_closed():** devuelve un microondas cerrado y sin ítem.
- **Ítem_placed():** simula el haber introducido un objeto en el microondas. Cambia el estado a abierto con ítem.

```
package microwave;

import microwave.Estado_Microondas.state;

public class Estado_OpenWithNoItem extends Estado_Microondas{
    protected Estado_OpenWithNoItem () {
        fase= state.OpenWithNoItem;
    }
    @Override
     protected Microwave door_closed (Microwave m) {


        return new Microwave();
    }
    @Override
    protected Microwave item_placed (Microwave m) {
        m.withItem=true;
        m.estado= new Estado_OpenWithItem();

        return m;
    }

}
```

Estado_OpenWithItem

En dicho estado, las operaciones que producen cambios son:

- **Ítem_removed():** vuelve al estado anterior y pone el boolean withItem a false.
- **Door_closed():** Cierra la puerta, apaga la lámpara y cambia de estado a cerrado con ítem.

```java
package microwave;

import microwave.Estado_Microondas.state;

public class Estado_OpenWithItem extends Estado_Microondas {
    protected Estado_OpenWithItem () {
        fase= state.OpenWithItem;
    }
    @Override
    protected Microwave item_removed(Microwave m) {
        m.estado= new Estado_OpenWithNoItem();
        m.withItem= false;
        return m;
    }
    @Override
    protected Microwave door_closed (Microwave m) {
        m.doorOpen= false;
        m.lampara.lamp_off();
        m.estado= new Estado_ClosedWithItem();
        return m;
    }


}
```

Estado_ClosedWithItem

Un microoandas cerrado con ítem tiene únicamente dos vías de cambio:

- **Cooking_start():** comprueba los requisitos para que se inicie el cocinado. Y asigna a las variables sus valores cuando están cocinando.
- **Door_opened():** produce un cambio de estado a abierto con ítem.

```java
package microwave;

public class Estado_ClosedWithItem extends Estado_Microondas {
    protected Estado_ClosedWithItem() {
        fase= state.ClosedWithItem;
    }
    @Override
    protected Microwave cooking_start(Microwave m) {
        if (m.timer>0 && m.power>0) {
            m.cooking= true;
            m.estado= new Estado_Cooking();
            m.calor.heating_on();
            m.gira.turntable_start();
            m.lampara.lamp_on();
            m.pantalla.setDisplay(String.valueOf(m.timer));
        }
        return m;
    }
    @Override
    protected Microwave door_opened(Microwave m) {
        m.doorOpen= true;
        m.estado= new Estado_OpenWithItem();
        m.lampara.lamp_on();
        return m;
    }


}
```

Estado_Cooking

```java
package microwave;

public class Estado_Cooking extends Estado_Microondas {

    public Estado_Cooking() {
        fase= state.Cooking;
    }
    @Override
    protected Microwave door_opened(Microwave m) {
        m.doorOpen=true;
        m=m.estado.cooking_stop(m);
        m.estado= new Estado_OpenWithItem();
        return m;
    }
    /*
     * en cooking stop paramos todo los procesos asociados a cocinar el alimento,
     * de forma que se pude reusar por otras funciones para parar de cocinar.
     */
    @Override
    protected Microwave cooking_stop(Microwave m ) {
        m.estado= new Estado_ClosedWithItem();
        m.calor.heating_off();
        m.gira.turntable_stop();
        m.lampara.lamp_off();
        m.pantalla.clearDisplay();
        m.cooking= false;

        return m;
    }
    @Override
    protected Microwave timer_dec(Microwave m) {
        if (m.timer>0) {
            if (m.timer==1) {
                m= m.estado.cooking_stop(m);
            }else {
                m.pantalla.setDisplay(String.valueOf(m.timer-1));

            }
            m.timer-=1;
        }
        return m;
    }
    @Override
    protected Microwave timer_reset(Microwave m) {
        m.timer=0;
        m.estado.cooking_stop(m);
        return m;
    }
@Override
protected Microwave tick(Microwave m) {
    m= timer_dec(m);
     if (m.estado instanceof Estado_ClosedWithItem ) {
         m.alarma.beep(2);
         m.pantalla.setDisplay("Tiempo finalizado \n");
     }else {
         m.pantalla.setDisplay(String.valueOf(m.timer));
     }
    return m;
}
@Override
protected Microwave power_dec(Microwave m) {
    if (m.power>0) {
        if (m.power==1) {
            //cambia el estado a no cocinando
            m.estado.cooking_stop(m);
        }
        m.power-=1;
    }
    return m;
}
@Override
protected Microwave power_reset(Microwave m) {
    m.power=0;
    m.estado.cooking_stop(m);
    return m;
}
@Override
protected Microwave setTimer(Microwave m,int i) {

    if (i==0) {
        m=this.timer_reset(m);
    }
    m.timer=i;
    m.estado.cooking_stop(m);
    return m;
}
```

Principales características:

- Es la clase que mas overrides tiene, ya que comportamientos como que se acabe el tiempo o que la potencia llegue a cero pararan el estado.
- La implementación de **cooking_strop()** ha resultado de utilidad para que sea llamada por todos aquellos métodos que interrumpen el cocinado.

2.Pruebas en JUnit

Se han implmentado los siguientes tests en JUnit, en el que se prueban los distintos escenarios con sus funciones.

```java
package microwave;

public class Estado_ClosedWithNoItem extends Estado_Microondas {
      // todas las operaciones basicas de la clase microondas
    public Estado_ClosedWithNoItem () {
        fase= state.ClosedWithNoItem;
        }
    @Override
    public Microwave door_opened (Microwave m) {
        // cambio de estado y cambio de micro
        m.estado= new Estado_OpenWithNoItem();
        m.lampara.lamp_on();
        m.doorOpen= true;
        return m;
    }
}
```

```java
package microwave;

import static org.junit.Assert.assertEquals;

class Estado_OpenWithNoItemTest {
     private Microwave m;

        @BeforeEach
        void inicialize_on() {

         m = new Microwave();
         m.door_opened();;
        }


    @Test
    void door_close() {
        m.door_closed();
        assert(!m.isDoorOpen());
        assertEquals ("ClosedWithNoItem", m.getEstado().fase.toString());

    }
    @Test
    void Itemplaced() {
        m.item_placed();
        assert(m.isWithItem());
        assertEquals ("OpenWithItem", m.getEstado().fase.toString());

    }

    @Test
    void OpenWithNoItemTest() {
        assertEquals("OpenWithNoItem",m.getEstado().fase());
        assert(m.isDoorOpen() );
        assert( !m.getCalor().isHeating() );
        assert( !m.isWithItem() );
        assert( !m.getGira().isMoving() );
        assert( m.getLampara().isLampOn());
        assert(!m.isCooking());
    }


}
```

```java
package microwave;

import static org.junit.Assert.assertEquals;

class Estado_OpenWithItemTest {
    private Microwave m;

    @BeforeEach
    void inicialize_oi() {

     m = new Microwave();
     m.door_opened();;
     m.item_placed();
    }

    @Test
    void item_removed() {
        m.item_removed();
        assert(!m.isWithItem());
        assertEquals("OpenWithNoItem",m.getEstado().fase());

    }
    @Test
    void door_close() {
        m.door_closed();
        assert(!m.isDoorOpen());
        assertEquals("ClosedWithItem",m.getEstado().fase());

    }

    @Test

    void Estado_OpenWithItemTest() {
        assertEquals("OpenWithItem",m.getEstado().fase());
        assert(m.isDoorOpen() );
        assert( !m.getCalor().isHeating() );
        assert( m.isWithItem() );
        assert(!m.getGira().isMoving() );
        assert( m.getLampara().isLampOn());
        assert(!m.isCooking());

    }
    @Test
    void OpenWithItemTest() {
        assertEquals("OpenWithItem",m.getEstado().fase());
        assert(m.isDoorOpen() );
        assert( !m.getCalor().isHeating() );
        assert( m.isWithItem() );
        assert( !m.getGira().isMoving() );
        assert( m.getLampara().isLampOn());
        assert(!m.isCooking());
    }

}
```

```java
package microwave;

import static org.junit.Assert.assertEquals;

class Estado_ClosedWithItemTest {
    private Microwave m;
    @BeforeEach
    void inicialize_ci() {
     m = new Microwave();
     m.door_opened();;
     m.item_placed();
     m.door_closed();
    }
    @Test
    void door_opened() {
        m.door_opened();
        assert(m.isDoorOpen());
        assertEquals ("OpenWithItem", m.getEstado().fase.toString());
    }
    @Test
    void cooking_start() {
        m.cooking_start();
        assert(!m.isCooking());
        m.setTimer(3);
        m.power_inc();
        m.power_inc();
        m.power_inc();
        assert(m.getTimer()>0);
        assert(m.getPower()>0);
        m.cooking_start();
        assert(m.isCooking());
        try {
                int a =Integer.parseInt(m.getPantalla().getDisplay());
        }
        catch (NumberFormatException nfe) {
                fail();// asi nos aseguramos de que esta imprimiendo el numero de segundos restantes en la pantalla
        }
        assertEquals ("Cooking", m.getEstado().fase.toString());
    }
    @Test
    void ClosedWithItemTest() {
        assertEquals("ClosedWithItem",m.getEstado().fase());
        assert(!m.isDoorOpen() );
        assert( !m.getCalor().isHeating() );
        assert( m.isWithItem() );
        assert( !m.getGira().isMoving() );
        assert( !m.getLampara().isLampOn());
        assert( !m.isCooking());
    }
}
```

```java
package microwave;

public class Estado_Cooking extends Estado_Microondas {

    public Estado_Cooking() {
        fase= state.Cooking;
    }
    @Override
    protected Microwave door_opened(Microwave m) {
        m.doorOpen=true;
        m=m.estado.cooking_stop(m);
        m.estado= new Estado_OpenWithItem();
        return m;
    }
    /*
     * en cooking stop paramos todo los procesos asociados a cocinar el alimento,
     * de forma que se pude reusar por otras funciones para parar de cocinar.
     */
    @Override
    protected Microwave cooking_stop(Microwave m ) {
        m.estado= new Estado_ClosedWithItem();
        m.calor.heating_off();
        m.gira.turntable_stop();
        m.lampara.lamp_off();
        m.pantalla.clearDisplay();
        m.cooking= false;

        return m;
    }
    @Override
    protected Microwave timer_dec(Microwave m) {
        if (m.timer>0) {
            if (m.timer==1) {
                m= m.estado.cooking_stop(m);
            }else {
                m.pantalla.setDisplay(String.valueOf(m.timer-1));

            }
            m.timer-=1;
        }
        return m;
    }
    @Override
    protected Microwave timer_reset(Microwave m) {
        m.timer=0;
        m.estado.cooking_stop(m);
        return m;
    }
    @Override
    protected Microwave tick(Microwave m) {
        m= timer_dec(m);
        if (m.estado instanceof Estado_ClosedWithItem ) {
            m.alarma.beep(2);
            m.pantalla.setDisplay("Tiempo finalizado \n");
        }else {
            m.pantalla.setDisplay(String.valueOf(m.timer));
        }
        return m;
    }
    @Override
    protected Microwave power_dec(Microwave m) {
        if (m.power>0) {
            if (m.power==1) {
                //cambia el estado a no cocinando
                m.estado.cooking_stop(m);
            }
            m.power-=1;
        }
        return m;
    }
    @Override
    protected Microwave power_reset(Microwave m) {
        m.power=0;
        m.estado.cooking_stop(m);
        return m;
    }
    @Override
    protected Microwave setTimer(Microwave m,int i) {

        if (i==0) {
            m=this.timer_reset(m);
        }
        m.timer=i;
        m.estado.cooking_stop(m);
        return m;
    }
```

```java
package microwave;

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

class NotImplementedTest {
    private Microwave m ;
    // en esta calse se probaran todos aquellos metodos que no se han probado en las demas clases,
    // aquellos metodos que se heredan de la calse padre Estado_Microondas.
    /*
     * iniciaremos un microonadas en el estado cerrrado con objeto
     */
    @BeforeEach
    void inicialize_ci() {
         m = new Microwave();
         m.door_opened();;
         m.item_placed();
         m.door_closed();
        }

    @Test
    void power_inc() {
        m.power_inc();
        assertEquals(1, m.getPower());
        m.power_inc();
        m.power_inc();
        m.power_inc();
        assertEquals(4, m.getPower());
        m.power_inc();
        assertEquals(5, m.getPower());// maximo valor posible
        m.power_inc();
        m.power_inc();
        m.power_inc();
        m.power_inc();
        m.power_inc();
        assertEquals(5, m.getPower());
    }

    @Test
    void power_dec() {
        m.power_inc();
        m.power_inc();
        m.power_inc();
        m.power_inc();
        m.power_inc();
        assertEquals(5, m.getPower());
        m.power_dec();
        assertEquals(4, m.getPower());
        m.power_dec();
        m.power_dec();
        m.power_dec();
        assertEquals(1, m.getPower());
        m.power_dec();
        assertEquals(0, m.getPower());
        m.power_dec();
        m.power_dec();
        assertEquals(0, m.getPower());
    }

    @Test
    void power_reset() {
        m.power_reset();
        assert(m.power==0);
    }

    @Test
    void timer_inc() {
        m.setTimer(298);
        m.timer_inc();
        m.timer_inc();
        assertEquals(300, m.getTimer());
        m.timer_inc();
        m.timer_inc();
        m.timer_inc();
        assertEquals(300, m.getTimer());
    }

    @Test
    void setTimer() {

    }
}
```

```java
package microwave;

import static org.junit.Assert.assertEquals;

class WorkflowTest {
private Microwave m ;
    // utilizaremos las funciones usadas en las demas clases de pruebas para
    // verificar que el estado interno del microondas es  correcto
    void ClosedWithNoItem() {


        assertEquals("ClosedWithNoItem",m.getEstado().fase());

        assert(!m.isDoorOpen() );
        assert( !m.getCalor().isHeating() );
        assert( !m.isWithItem() );
        assert( !m.getGira().isMoving() );
        assert( !m.getLampara().isLampOn());
        assert(!m.isCooking());
    }
    void OpenWithNoItemTest() {
        assertEquals("OpenWithNoItem",m.getEstado().fase());
        assert(m.isDoorOpen() );
        assert( !m.getCalor().isHeating() );
        assert( !m.isWithItem() );
        assert( !m.getGira().isMoving() );
        assert( m.getLampara().isLampOn());
        assert(!m.isCooking());
    }
    void OpenWithItemTest() {
        assertEquals("OpenWithItem",m.getEstado().fase());
        assert(m.isDoorOpen() );
        assert( !m.getCalor().isHeating() );
        assert( m.isWithItem() );
        assert( !m.getGira().isMoving() );
        assert( m.getLampara().isLampOn());
        assert(!m.isCooking());
    }
    void ClosedWithItemTest() {
        assertEquals("ClosedWithItem",m.getEstado().fase());
        assert(!m.isDoorOpen() );
        assert( !m.getCalor().isHeating() );
        assert( m.isWithItem() );
        assert( !m.getGira().isMoving() );
        assert( !m.getLampara().isLampOn());
        assert( !m.isCooking());
    }
    void Cooking() {
        assertEquals("Cooking",m.getEstado().fase());
        assert(!m.isDoorOpen() );
        assert( m.getCalor().isHeating() );
        assert( m.isWithItem() );
        assert( m.getGira().isMoving() );
        assert( m.getLampara().isLampOn());
        assert( m.isCooking());
    }


    /*
     * Simulacoin de un caso de uso comun.
     *  1. se inicializa el microondas
     *  2.se abre
     *  3.se introduce iten
     *  4.se estabelece el tiempo y la potencia
     *  5.Pulsa el boton de start
     */
    @Test
    void caso_de_uso_normal() {
        m = new Microwave();
         ClosedWithNoItem();
        m.door_opened();
        this.OpenWithNoItemTest();
        m.item_placed();
        this.OpenWithItemTest();
        m.door_closed();
        this.ClosedWithItemTest();
        m.power_inc();
        m.power_inc();
        m.power_inc();
        m.setTimer(8);
        m.cooking_start();
        this.Cooking();
        for (int i=0 ;i<30;i++) {
            m.tick();
        }
        m.power_reset();
        assertEquals(0,m.power);
        this.ClosedWithItemTest();




    }
}
```

Enlace al repositorio git con los archivos desarrollados:
https://github.com/davidcubillos0211/isa-practica-2-