Práctica 2 Patrones de Diseño

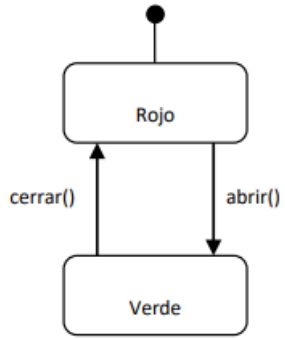
Ingeniería del Software Avanzada
Creado por: Francisco Rodríguez-Córdoba Lucena

Nombre del logotipo

Práctica 2

Apartado a

En este apartado hemos implementado un dispositivo software Biestable.



Según este diagrama de estados, el dispositivo comenzará en el estado *Rojo,* que reacciona al mensaje *abrir()* cambiando de estado al *Verde*, en cambio si recibe el mensaje *cerrar()* se mantiene en el mismo estado.
Si se encuentra en el estado *Verde* reacciona al mensaje *cerrar()* cambiando de estado al *Rojo*, en cambio si recibe el mensaje *abrir()* se mantiene en el mismo estado.
Además si el dispositivo recibe el mensaje *estado()* deberá devolver "cerrado" si está en el estado *Rojo* y "abierto" si está en el estado *Verde.*

Para esta práctica he utilizado el patrón **Estado** ya que permite a un objeto alterar su comportamiento cuando su estado interno cambia.

Para ello creamos la clase abstracta ***State*** que contiene los métodos que los diferentes estados deben implementar, este caso:

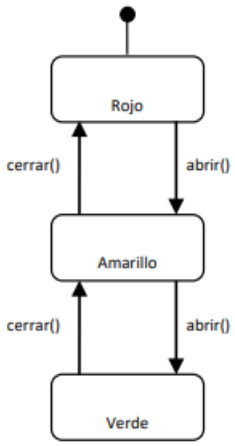
```
public abstract State abrir();
public abstract String estado();
public abstract State cerrar();
```

Así como una clase para cada estado : ***GreenState, RedState***.
En estas clases implementamos los métodos *abrir(),estado(),cerrar().*

También creamos la clase *Semáforo,* en la que en el constructor establecemos que siempre que se cree un semáforo empezará en el estado *Rojo.* Tambien llamamos a los métodos de la clase abstracta *State: abrir(),estado(),cerrar()*

Apartado b

En este apartado se debe implementar otro tipo de dispositivo, esta vez triestable, el cuál incorpora un nuevo estado ***YellowState*** que va a ser una nueva clase que extiende de ***State*** y en el caso de recibir el mensaje estado() debería devolver “precaución”.



Hemos seguido usando el patrón **Estado** y al ser dispositivos semejantes no se han producido muchos cambios.

Se ha añadido el enum ***Type*** para diferenciar el tipo de semáforo.

```
public enum Type {
    Biestable, Triestable

}
```

En la clase ***Semaforo*** se ha modificado solo el constructor, para que a la hora de crear el semáforo distinguir de que ***Type*** es. Y se ha añadido el método getType() que devuelve el tipo de semáforo para que los diferentes estados actuen dependiendo de dicho tipo.

En los diferentes estados solo se ha añadido un “if” que comprueba de que tipo es el semáforo para actuar de una manera u otra, en los diferentes métodos.

Apartado c

En este apartado añadimos un nuevo método cambio(), para intercambiar de un tipo de semáforo a otro.

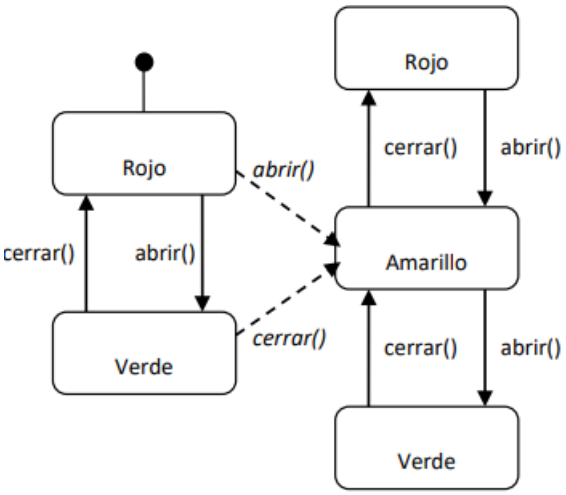


Figura (c). Transición de Biestable a Triestable

Para ello en la clase ***Semaforo*** hemos añadido los métodos setType(Type t) y cambio(). Además hemos creado otro estado ***TransicionState*** para que una vez que reciba el mensaje cambio(), se encuentre en el estado que sea, se pase a ***TransitionState***, y hasta que no reciba los mensajes de abrir() o cerrar() no cambiará finalmente al otro tipo de dispositivo (Biestable o Triestable).

Si el semaforo es biestable y recibe el mensaje cambio() pasará al estado ***YellowState*** si vuelve a recibir otro mensaje de abrir() o cerrar().
Si el semaforo es triestable da igual en el estado que se encuentre, después del mensaje cambio() si recibe el mensaje abrir() pasará al estado ***GreenState*** y si recibe cerrar() al estado ***RedState***.
La clase ***TransicionState*** se le ha añadido los métodos de los estados.

Hemos seguido con el patrón Estado , y en este apartado solo hemos añadido el estado nombrado anteriormente y alli hemos realizado los cambios.

A continuación las pruebas realizadas

```java
public class main {
    public static void main(String[] args) {
        Type biestable= Type.Biestable;
        Type triestable=Type.Triestable;
        // Creamos un semaforo biestable
        Semaforo s = new Semaforo(biestable);

        // Vemos si su estado inicial es "cerrado"
        System.out.println(s.estado());
        // Cambiamos a un dispositivo triestable
        s.cambio();
        //Observamos que aún no se haya hecho el cambio, ya que
        //tiene que recibir abrir() o cerrar()
        System.out.println(s.getType());
        // Ahora si le mandamos el mensaje abrir()
        s.abrir();
        //Debería devolver estado= precaucion y type=Triestable
        System.out.println(s.estado());
        System.out.println(s.getType());
        //Volvemos a hacer el cambio y no debería devolver aún el tipo= Biestable
        s.cambio();
        System.out.println(s.getType());
        // Tras abrilo debería devolver Biestable y abierto como estado
        s.abrir();
        System.out.println(s.estado());
        System.out.println(s.getType());
        //Cambiamos, mensaje abrir() para estado precaución y
        //abrir() otra vez para estado abierto
        s.cambio();
        s.abrir();
        s.abrir();
        System.out.println(s.estado());
        System.out.println(s.getType());
        /Cambiamos otra vez
        .cambio();
        ystem.out.println(s.estado());
        ystem.out.println(s.getType());
        /Al cerrar debería salir como resultado, cerrado y biestable
        .cerrar();
        ystem.out.println(s.estado());
        ystem.out.println(s.getType());
```

```
cerrado
Biestable
precaución
Triestable
Triestable
abierto
Biestable
abierto
Triestable
transicion
Triestable
cerrado
Biestable
```

URL github:

https://github.com/Franrodriguezcordoba/Triestables