Triestables-Práctica design patterns

El objetivo de esta práctica es implementar en Java biestables y triestables siguiendo un patrón de diseño que nos permita reutilizar el máximo código posible e implementar nuevas funcionalidades realizando los menores cambios posibles. Para el desarrollo de esta práctica se ha decidido seguir un patrón de diseño estado.

Se adjunta imágenes del código:

Apartado a) Biestable

```java
package Biestable;

public class Biestable {
    private static Estado est;

    public Biestable(Estado e) {
        est = e;
    }

    public void Abrir() {
        est.abrir();
    }

    public void Cerrar() {
        est.cerrar();
    }

    public String estado() {
        return est.estado();
    }

    public static void setEst(Estado est) {
        Biestable.est = est;
    }


}
```

```java
package Biestable;

public interface Estado {
    void abrir();
    void cerrar();
    String estado();
}
```

```java
package Biestable;

public class Rojo implements Estado {
    public Rojo() {
    }
    @Override
    public void abrir() {
        Biestable.setEst(new Verde());

    }

    @Override
    public void cerrar() {
    }

    @Override
    public String estado() {

        return "Cerrado";

    }


}
```

```java
package Biestable;

public class Verde implements Estado {

    @Override
    public void abrir() {
    }

    @Override
    public void cerrar() {
        Biestable.setEst(new Rojo());

    }

    @Override
    public String estado() {
        return "Abierto";

    }

}
```

Apartado b) Triestable

```java
package Triestable;

public class Triestable {
    private static Estado est;

    public Triestable(Estado e) {
        est = e;
    }

    public void Abrir() {
        est.abrir();
    }

    public void Cerrar() {
        est.cerrar();
    }

    public String estado() {
        return est.estado();
    }

    public static void setEst(Estado est) {
        Triestable.est = est;
    }
}
```

```java
package Triestable;

public interface Estado {
    void abrir();
    void cerrar();
    String estado();
}
```

```java
package Triestable;

public class Rojo implements Estado {
    public Rojo() {
    }
    @Override
    public void abrir() {
        Triestable.setEst(new Amarillo());

    }

    @Override
    public void cerrar() {
    }

    @Override
    public String estado() {

        return "Cerrado";

    }
}
```

```java
package Triestable;


public class Verde implements Estado {
    public Verde() {
    }
    @Override
    public void abrir() {


    }

    @Override
    public void cerrar() {
        Triestable.setEst(new Amarillo());
    }

    @Override
    public String estado() {
        return "Abierto";

    }
}
```

```java
package Triestable;

public class Amarillo implements Estado {
    public Amarillo() {

    }

    @Override
    public void abrir() {

        Triestable.setEst(new Verde());
    }

    @Override
    public void cerrar() {
        Triestable.setEst(new Rojo());

    }

    @Override
    public String estado() {

        return "Precaución";
    }

}
```

Apartado c) Posibilidad de alternar entre biestable y triestable

```java
package Alternador;


public class Alternador {
    private static Estado est;

    public Alternador(Estado e) {
        est = e;
    }

    public void Abrir() {
        est.abrir();
    }

    public void Cerrar() {
        est.cerrar();
    }

    public String estado() {
        return est.estado();
    }

    public static void setEst(Estado est) {
        Alternador.est = est;
    }
    public void cambio() {
        est.cambio();
    }

}
```

```java
package Alternador;

public interface Estado {
    void abrir();
    void cerrar();
    String estado();
    void cambio();

}
```

```java
public class Amarillo implements Estado {
    public Amarillo() {

    }

    @Override
    public void abrir() {

        Alternador.setEst(new VerdeTri());
    }

    @Override
    public void cerrar() {
        Alternador.setEst(new RojoTri());

    }

    @Override
    public String estado() {

        return "precaución";
    }

    @Override
    public void cambio() {
        // TODO Auto-generated method stub
        Alternador.setEst(new CambioBi());
    }

}
```

```java
package Alternador;

public class CambioBi implements Estado {

    @Override
    public void abrir() {
        // TODO Auto-generated method stub
        Alternador.setEst(new VerdeBi());

    }

    @Override
    public void cerrar() {
        // TODO Auto-generated method stub
        Alternador.setEst(new RojoBi());

    }

    @Override
    public String estado() {
        // TODO Auto-generated method stub
        return "precaución";
    }

    @Override
    public void cambio() {
        // TODO Auto-generated method stub
        Alternador.setEst(new Amarillo());

    }

}
```

```java
public class RojoBi implements Estado{
    public RojoBi() {

    }

    @Override
    public void abrir() {
        // TODO Auto-generated method stub
        Alternador.setEst(new VerdeBi());

    }

    @Override
    public void cerrar() {
        // TODO Auto-generated method stub

    }

    @Override
    public String estado() {
        // TODO Auto-generated method stub
        return "cerrado";
    }

    @Override
    public void cambio() {
        // TODO Auto-generated method stub
        Alternador.setEst(new RojoTri());

    }

}
```

```java
public class VerdeBi implements Estado {
        public VerdeBi(){

        }
        @Override
        public void abrir() {
            // TODO Auto-generated method stub


        }

        @Override
        public void cerrar() {
            // TODO Auto-generated method stub
            Alternador.setEst(new RojoBi());
        }

        @Override
        public String estado() {
            // TODO Auto-generated method stub
            return "abierto";

        }
        @Override
        public void cambio() {
            // TODO Auto-generated method stub
            Alternador.setEst(new VerdeTri());
        }

    }
```

```java
public class VerdeTri implements Estado {
    public VerdeTri() {

    }

    @Override
    public void abrir() {
        // TODO Auto-generated method stub


    }

    @Override
    public void cerrar() {
        // TODO Auto-generated method stub
        Alternador.setEst(new Amarillo());
    }

    @Override
    public String estado() {
        // TODO Auto-generated method stub
        return "abierto";
    }

    @Override
    public void cambio() {
        // TODO Auto-generated method stub
        Alternador.setEst(new VerdeBi());
    }
}
```

```java
public class RojoTri implements Estado{
    public RojoTri(){

    }

    @Override
    public void abrir() {
        // TODO Auto-generated method stub
        Alternador.setEst(new Amarillo());

    }

    @Override
    public void cerrar() {
        // TODO Auto-generated method stub

    }

    @Override
    public String estado() {
        // TODO Auto-generated method stub
        return "cerrado";
    }

    @Override
    public void cambio() {
        // TODO Auto-generated method stub
        Alternador.setEst(new RojoBi());

    }

}
```

Pruebas

```
class AlternadorTest {

    @Test
    void testAlternador() {
        assertEquals(new Alternador(new VerdeBi()).estado(), "abierto");
        assertEquals(new Alternador(new RojoBi()).estado(), "cerrado");
        assertEquals(new Alternador(new VerdeTri()).estado(), "abierto");
        assertEquals(new Alternador(new RojoTri()).estado(), "cerrado");
        assertEquals(new Alternador(new Amarillo()).estado(), "precaución");
        assertEquals(new Alternador(new CambioBi()).estado(), "precaución");
    }

    @Test
    void testAbrir() {
        Alternador RojoBi = new Alternador(new RojoBi());
        RojoBi.Abrir();
        assertEquals(RojoBi.estado(), "abierto");
        Alternador RojoTri = new Alternador(new RojoTri());
        RojoTri.Abrir();
        assertEquals(RojoTri.estado(), "precaución");
        Alternador Amarillo = new Alternador(new Amarillo());
        Amarillo.Abrir();
        assertEquals(Amarillo.estado(), "abierto");

    }
```

```
@Test
void testCerrar() {
    Alternador VerdeBi = new Alternador(new VerdeBi());
    VerdeBi.Cerrar();
    assertEquals(VerdeBi.estado(), "cerrado");
    Alternador VerdeTri = new Alternador(new VerdeTri());
    VerdeTri.Cerrar();
    assertEquals(VerdeTri.estado(), "precaución");
    Alternador Amarillo = new Alternador(new Amarillo());
    Amarillo.Cerrar();
    assertEquals(Amarillo.estado(), "cerrado");

}

@Test
void testCambio() {
    Alternador Semaforo = new Alternador(new VerdeBi());
    Semaforo.cambio();
    Semaforo.Cerrar();
    assertEquals(Semaforo.estado(), "precaución");
    Semaforo.cambio();
    assertEquals(Semaforo.estado(), "precaución");
    Semaforo.Cerrar();
    assertEquals(Semaforo.estado(), "cerrado");
    Semaforo.cambio();
    assertEquals(Semaforo.estado(), "cerrado");
    Semaforo.Abrir();
    assertEquals(Semaforo.estado(), "precaución");

}
```

Interpretación de código

Se ha elegido el patrón de diseño estado porque se ha interpretado que es el que mejor resultado aportaría al proyecto, a fin de cuentas, se aplican los mismos métodos peros estos cambian en función del estado en el que se encuentre.

Procedo a explicar el apartado c), pues considero que es el mas completo y que su explicación deja clara las otras dos implementaciones de las que ha evolucionado.

El objetivo era un sistema que permitiese alternar entre un semáforo biestable y un semáforo triestable, es por esto que se ha creado una interfaz “Estado” que podrá ser del tipo de los distintos colores del semáforo, de este modo cuando se llame a un método este evolucionara a su siguiente estado natural.

Se han diseñado 6 estados, pues al existir dos tipos de biestable existen dos evoluciones naturales a cada color del semáforo, en función del método” cambio()” un color verde actuara como un verde triestable o un verde biestable, así mismo sucede con los demás colores.

Finalmente se han implementado con Junit una serie de pruebas del comportamiento esperado de nuestro semáforo y todas han resultado satisfactorias.

Enlace GitHub: https://github.com/alexpascualm/Pr-SemaforoState/tree/main