Ejercicio Código 3

Autor: José Guillermo Sandoval Huerta

Fecha: 02 noviembre 2023

Implementación patrón Observer

El patrón Observer nos dice que un objeto conocido como "sujeto", notifica automáticamente a sus "observadores" cuando su estado cambia. En la vida real, un ejemplo de este tipo es un semáforo, cuando cambia de color, éste le puede avisar sea a un coche o a un peatón lo que puede realizar.

De acuerdo al diagrama de clases, este patrón debe de tener la siguiente estructura:

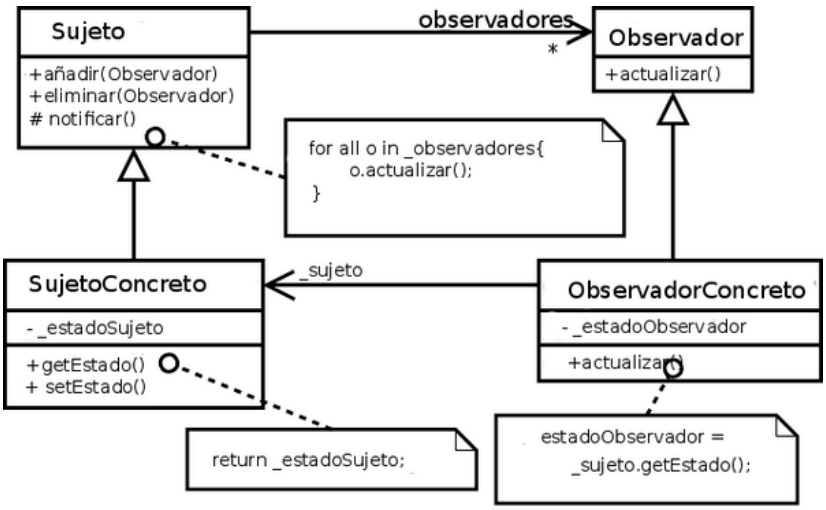

Imagen 1. Diagrama de clase Patrón Observer.

Donde se programó de la siguiente forma.

Código 1. Clase Observador.

```
…

public abstract class Observador {

        Sujeto sujeto;

        public Observador(Sujeto sujeto) {
                this.sujeto = sujeto;
                sujeto.agregarObservador(this);
        }

        abstract void verLuzVerde();  // Update 1

        abstract void verLuzAmarrilla();  // Update 2

        abstract void verLuzRoja();  // Update 3
}
```

Donde se tiene 3 métodos que actualizan el comportamiento de los observadores.

La clase **Sujeto**, la cual gestiona un arreglo de objetos Observadores y notifica a cada uno el comportamiento a realizar.

Código 2. Clase Sujeto.

```
…

public class Sujeto {

	List<Observador> listaObservadores = new ArrayList<>();

	void agregarObservador(Observador o) {
		listaObservadores.add(o);
	}

	void eliminarObservador(Observador o) {
		listaObservadores.remove(o);
	}

	void cambiarVerde() {  // Notificación 1
		System.out.println("**********Luz Verde**********");
		for (Observador o:listaObservadores)
			o.verLuzVerde();
	}

	void cambiarAmarillo() {  // Notificación 2
		System.out.println("**********Luz Amarilla**********");
		for (Observador o:listaObservadores)
			o.verLuzAmarrilla();
	}

	void cambiarRojo() {  // Notificación 3
		System.out.println("**********Luz Roja**********");
		for (Observador o:listaObservadores)
			o.verLuzRoja();
	}
}
```

La clase **Semaforo**, que extiende de sujeto, siendo el sujeto Concreto el cual realizara una serie de cambios.

Código 3. Clase Semaforo.

```
…

public class Semaforo extends Sujeto{

	void luzVerde(){  // Cambio 1
		cambiarVerde();
	}

	void luzAmarilla(){  // Cambio 2
		cambiarAmarillo();
```

```
        }

        void luzRoja(){  // Cambio 3
                cambiarRojo();
        }
}
```

La clase **Peaton**, que extiendo de observador y que responde a las notificaciones que la clase Observador notifique.

Código 4. Clase Peaton.

```
…

public class Peaton extends Observador{

        public Peaton(Sujeto sj) {
                super(sj);
        }

        void cruzar() {
                System.out.println("Peaton: cruzar calle");
        }

        void prepararse() {
                System.out.println("Peaton: prepararse a cruzar");
        }

        void esperar() {
                System.out.println("Peaton: esperar en la esquina");
        }

        @Override
        void verLuzVerde() {  // Respuesta 1
                esperar();
        }

        @Override
        void verLuzAmarrilla() {  // Respuesta 2
                prepararse();
        }

        @Override
        void verLuzRoja() {  // Respuesta 3
                cruzar();
        }
}
```

La clase **Coche**, que extiendo de observador y que responde a las notificaciones que la clase Observador notifique.

Código 5. Clase Coche.

```
…

public class Coche extends Observador{
	private String color;

	public Coche(String color, Sujeto sj) {
		super(sj);
		this.color = color;
	}

	void acelerar() {
		System.out.println("Coche "+color+": acelerar");
	}

	void frenar() {
		System.out.println("Coche "+color+": frenar");
	}

	void parar() {
		System.out.println("Coche "+color+": parar");
	}

	@Override
	void verLuzVerde() {  // Respuesta 1
		acelerar();
	}

	@Override
	void verLuzAmarrilla() {  // Respuesta 2
		frenar();
	}

	@Override
	void verLuzRoja() {  // Respuesta 3
		parar();
	}
}
```

La clase **Policia**, que extiendo de observador y que responde a las notificaciones que la clase Observador notifique.

Código 6. Clase Policia.

```
…

public class Policia extends Observador{

	private String tipo;

	public Policia(String tipo, Sujeto sj) {
		super(sj);
		this.tipo = tipo;
	}

	void dirigirPeaton() {
		System.out.println("Policia de "+ tipo +": dirigir peatón");
	}

	void pararCoches() {
		System.out.println("Policia de "+ tipo +": parar coches");
	}

	void dirigirCoches() {
		System.out.println("Policia de "+ tipo +": dirigir coches");
	}

	@Override
	void verLuzVerde() {  // Respuesta 1
		dirigirCoches();
	}

	@Override
	void verLuzAmarrilla() {  // Respuesta 2
		pararCoches();
	}

	@Override
	void verLuzRoja() {  // Respuesta 3
		dirigirPeaton();
	}
}
```

Y por último la clase **Principal**, donde se instancian los objetos y él se declaran los diferentes comportamientos del sujeto concreto que es el semáforo.

Código 7. Clase Principal.

```
…

public class Principal {

	public static void main(String[] args) {
		Semaforo semaforo = new Semaforo();
```

```
                Peaton peaton = new Peaton(semaforo);
                Coche coche1 = new Coche("azul", semaforo);
                Coche coche2 = new Coche("blanco", semaforo);

                Policia policiaTransito = new Policia("transito", semaforo);

                semaforo.cambiarVerde();
                System.out.println("");

                semaforo.cambiarAmarillo();
                System.out.println("");

                semaforo.cambiarRojo();
                System.out.println("");

        }

}
```