PATRÓN OBSERVER

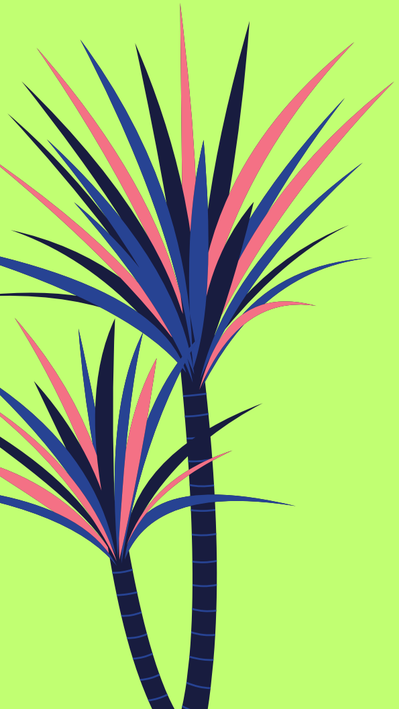
Danna Valeria Morales Aguilar

¿QUÉ ES EL PATRÓN OBSERVER?

Es un patrón de diseño de comportamiento para establecer una relación de dependencia uno a muchos entre objetos.
Un objeto llamado sujeto mantiene una lista de observadores y notifica automaticamente a los observadores cada vez que cambia su estado.
Basado en el diseño SOLID, sugiere que estas clases deben estar abiertas para la extensión pero cerradas para la modficiación.

ESCENARIO

Mostrando un ejemplo sobre lo que es el Patrón Observer, decidi basarme en una cafeteria como lo es Starbucks, esta actuará como el sujeto observado y los clientes serán los observadores interesados en recibir notificaciones cuando se prepare su café.



```java
public interface ObservadorCafe {
    void recibirCafePreparado(String tipoCafe);
}
```

Es una interfaz que define el método recibirCafePreparado(). Todos los clientes deben implementar esta interfaz para recibir las notificaciones de nuevos cafés preparados.

```java
public class Cliente implements ObservadorCafe {
    private String nombre;

    public Cliente(String nombre) {
        this.nombre = nombre;
    }

    @Override
    public void recibirCafePreparado(String tipoCafe) {
        System.out.println(nombre + " recibió un café de tipo: " +
    }
}
```

Representa a un obervador interesado en recibir notificaciones sobre los cafes preparados. Implementa la interfaz ObservadorCafe y da la implementación para el método recibirCafePreparado().

La clase Starbucks es el sujeto observable. Tiene una lista de observadores que son los clientes. Tiene el metodo notificar y retirar para agregar y eliminar clientes de la lista de observadores.
El método prepararCafe() simula la accion de preparar un café y notifica sobre el tipo de café que se esta preparando.

```java
public class Starbucks {
    private List<ObservadorCafe> observadores = new ArrayList<>();

    public void notificar(ObservadorCafe observador) {
        observadores.add(observador);
    }

    public void retirar(ObservadorCafe observador) {
        observadores.remove(observador);
    }

    public void prepararCafe(String tipoCafe) {
        System.out.println("Preparando café: " + tipoCafe);
        notificarObservadores(tipoCafe);
    }

    private void notificarObservadores(String tipoCafe) {
        for (ObservadorCafe observador : observadores) {
            observador.recibirCafePreparado(tipoCafe);
        }
    }
}
```

```
Preparando café: Capuccino
Cliente1 recibió un café de tipo: Capuccino
Cliente2 recibió un café de tipo: Capuccino
Preparando café: Chocolate
Cliente1 recibió un café de tipo: Chocolate
```

```java
public class Main {

    public static void main(String[] args) {
        Starbucks starbucks = new Starbucks();

        Cliente cliente1 = new Cliente("Cliente1");
        Cliente cliente2 = new Cliente("Cliente2");

        starbucks.notificar(cliente1);
        starbucks.notificar(cliente2);

        starbucks.prepararCafe("Capuccino");

        starbucks.retirar(cliente2);

        starbucks.prepararCafe("Chocolate");
    }
}
```

Creamos una instancia de Starbucks, dos clientes y luego ambos clientes mediante el método notificar.
Se prepara un café capuccino y se notificara a ambos clientes, al retirar al cliente2 , se prepara un chocolate y ya no se le notifica a este.

CONCLUSIÓN

Como se puede observar, los clientes reciben notificaciones cada vez que se prepara un nuevo café en la cafetería. Al retirar a un cliente de la lista de observadores, este ya no recibe las notificaciones.
Este es un ejemplo del patrón Observer, donde varios objetos (clientes) están interesados en recibir notificaciones de un objeto observable (cafetería) cuando ocurre un evento (se prepara un nuevo café).