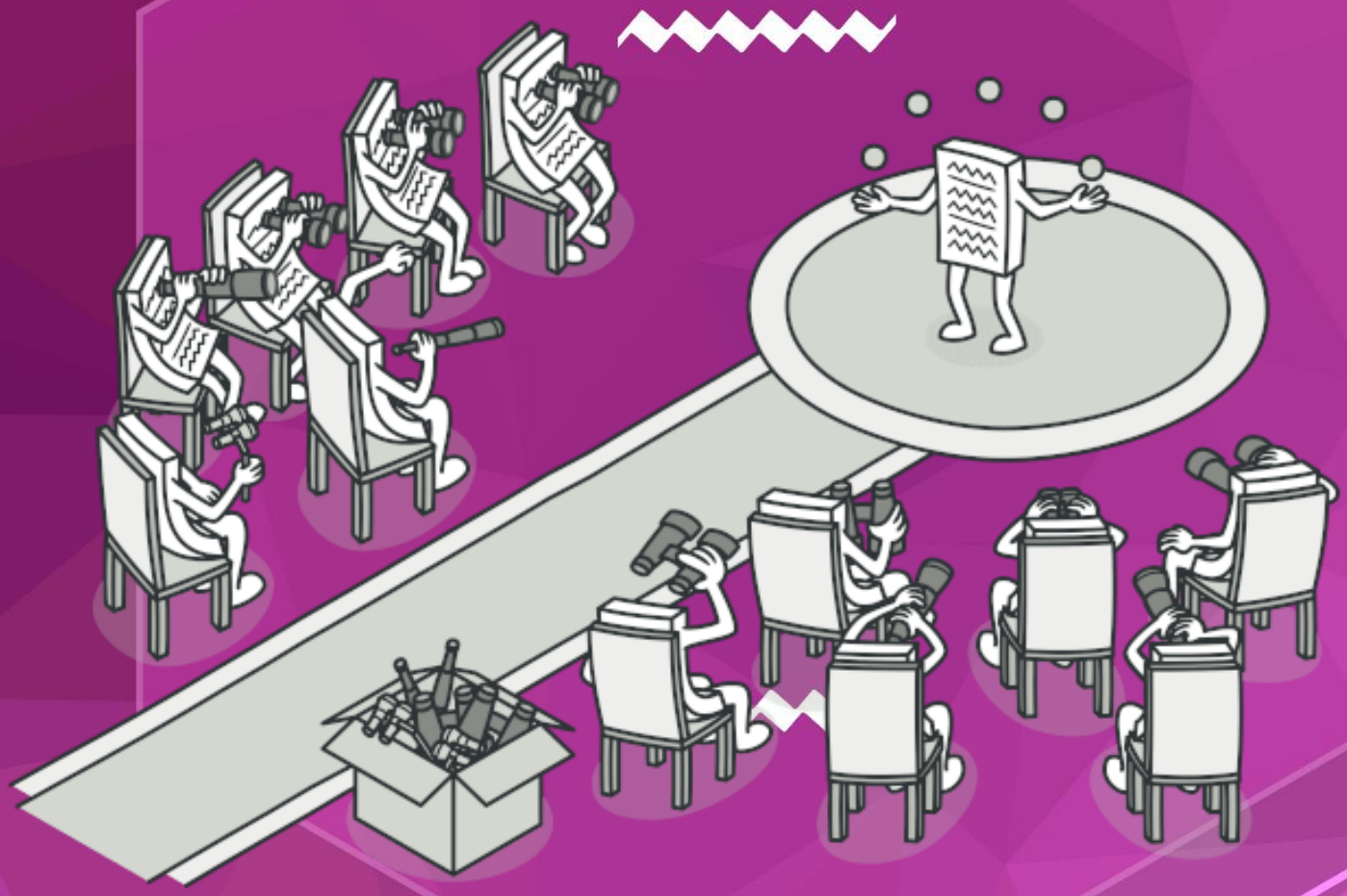
Patrones de diseño:

Patrones de comportamiento.

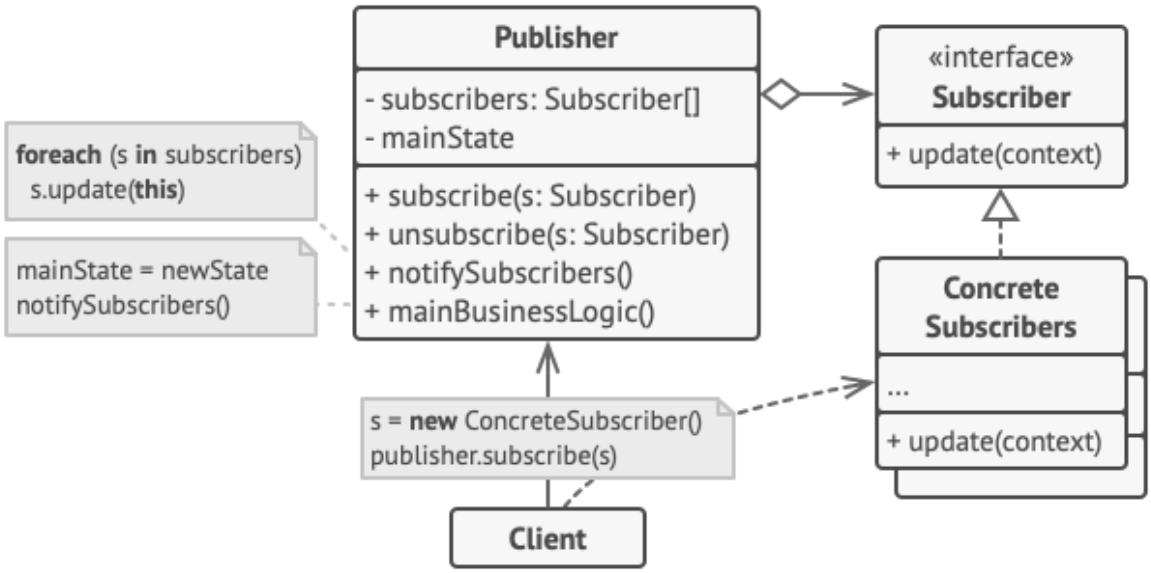
Observer



Lucero Guadalupe Domínguez Pérez

Observer es un patrón de diseño de comportamiento que te permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que están observando.

Estructura:



Ejemplo en código Java:

1. Creamos la interfaz **Observer**

```java
// Creamos la interfaz Observer
    interface Observer {

        // Creamos un metodo void, que rcibe por parametro un String y un double
        void actualizar(String productName, double price);

    }
```

2. Creamos la clase **Subject** que interpreta el objeto observado

```java
//Creamos la clase Compra (Subject) que interpreta como el objeto observado
class Compra {

    //Creamos un arrayList, para ir creando la lista de observadores
    private List<Observer> observers = new ArrayList<>();

    // Creamos el metodo para agregar observadores
    public void agregarObservador(Observer observer) { observers.add(observer);  }

    // Creamos el metodo para eliminar observadores
    public void eliminarObservador(Observer observer) { System.out.println("Se desactivaron la
notificaciones: ");
        observers.remove(observer);
    }

    // Creamos el metodo para notificar las compras a los observadores
    public void norificarObservador(String productoNombre, double precio) {
        for (Observer observer : observers) {
            observer.actualizar(productoNombre, precio);
        }
    }

    // Creamos un metodo, para simular una compra realizada
    public void compraRealizada(String productoNombre, double precio) {
        // En este metodo, se hace la  notificacion con la logica que se implementa
        norificarObservador(productoNombre, precio);
    }
}
```

3. Creamos la clase que represente a los observadores

```java
//Clase Cliente, hereda de la interfaz de Observer
class Cliente implements Observer {

    //Creamos una variable de referencia de tipo String
    private String nombre;

    //Creamos el constructor Cliente, que recibe por parametro un String
    public Cliente(String nombre) {
        this.nombre = nombre;
    }

    //Sobreescribimos el metodo actualizar, con su comportamiento
    @Override
    public void actualizar(String productoNombre, double precio) {
        System.out.println(nombre + ", realizaste compra de: " + productoNombre + " por $" + precio);
    }
}
```

4. Creamos el Principal para ejecutar el patrón **Observer**

```java
public class Notificacion {
        public static void main(String[] args) {

            //Creamos la variable de referencia
            Compra Compra = new Compra();

            // Creamos observadores (clientes)
            Observer cliente1 = new Cliente("Angela Aguilar");
            Observer cliente2 = new Cliente("Pepe Aguilar");

            // Los clientes se suscriben para recibir las notificaciones de sus compras realizadas
            Compra.agregarObservador(cliente1);
            // Simulamos una compra realizada
            Compra.compraRealizada("Vestido", 350.56);
            // Simulamos otra compra realizada
            Compra.compraRealizada("Zapatos", 400);

            System.out.println("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*");

            Compra.agregarObservador(cliente2);
            Compra.compraRealizada("Pantalon", 350.56);
            // Simulamos otra compra realizada
            Compra.compraRealizada("Zapatos", 400);

            // Los clientes desactivan las notificaciones de sus compras realizadas
            Compra.eliminarObservador(cliente1);

        }
    }
```