observer

Wendy Julieth guio quintero

Patrón de Diseño Observer - Explicación General

¿Qué es el Patrón Observer?

El patrón Observer es un patrón de comportamiento que define una dependencia uno-a-muchos entre objetos, de manera que cuando un objeto cambia de estado, todos sus dependientes (observadores) son notificados y actualizados automáticamente.

Este patrón es muy útil cuando varios objetos necesitan estar informados de los cambios en otro objeto sin que haya un acoplamiento fuerte entre ellos.

Características Principales

Bajo acoplamiento entre el sujeto (objeto observado) y los observadores.

Reactividad automática: los observadores se actualizan automáticamente ante cambios del sujeto.

Escalabilidad: puedes agregar nuevos observadores sin modificar el sujeto.

Flexibilidad: el sujeto no necesita saber detalles concretos de sus observadores.

🧱 Estructura del Patrón Observer en Java

Interfaz Observer

Define el método update(), que debe implementar cualquier clase que desee recibir notificaciones.

java

Copiar

Editar

public interface Observer {

void update(String mensaje);

}

Interfaz Subject (Observable)

Define métodos para registrar, eliminar y notificar observadores.

java

Copiar

Editar

public interface Subject {

void agregarObservador(Observer o);

void eliminarObservador(Observer o);

void notificarObservadores(String mensaje);

}

Clase Concreta que implementa Subject

Gestiona la lista de observadores y contiene la lógica para notificarles.

java

Copiar

Editar

public class CanalYouTube implements Subject {

private List<Observer> observadores = new ArrayList<>();

public void agregarObservador(Observer o) {

observadores.add(o);

}

public void eliminarObservador(Observer o) {

observadores.remove(o);

}

public void notificarObservadores(String mensaje) {

for (Observer o : observadores) {

o.update(mensaje);

}

}

public void subirNuevoVideo(String titulo) {

System.out.println("Nuevo video: " + titulo);

notificarObservadores(titulo);

}

}

Clases Concretas que implementan Observer

Implementan la lógica de actualización al recibir una notificación.

java

Copiar

Editar

public class Usuario implements Observer {

private String nombre;

public Usuario(String nombre) {

this.nombre = nombre;

}

public void update(String mensaje) {

System.out.println(nombre + " recibió la notificación: " + mensaje);

}

}

Clase Main

Demuestra el uso del patrón.

java

Copiar

Editar

public class Main {

public static void main(String[] args) {

CanalYouTube canal = new CanalYouTube();

Usuario user1 = new Usuario("Carlos");

Usuario user2 = new Usuario("Ana");

canal.agregarObservador(user1);

canal.agregarObservador(user2);

canal.subirNuevoVideo("Tutorial Observer en Java");

}

}

✅ Conclusión

El patrón Observer permite una comunicación eficiente entre objetos que dependen unos de otros sin generar un acoplamiento fuerte. Es ideal para sistemas donde los cambios en un componente deben reflejarse automáticamente en otros, como en interfaces gráficas, sistemas de eventos, redes sociales, y más.