Mediator

Introduccion

En ingeniería de software, los patrones de diseño conductual se ocupan de la asignación de responsabilidades entre objetos y el comportamiento de encapsulación en un objeto para delegar solicitudes. Y la motivación detrás del Patrón de diseño del mediador es proporcionar una comunicación adecuada entre los componentes al permitir que los componentes estén al tanto (o no, también, dependiendo del caso de uso) de la presencia o ausencia del otro en el sistema .

¿Que es mediator? 🤨

> Facilita la comunicación entre objetos

El mediador implementa una funcionalidad que dicta `cómo interactúa un conjunto de objetos entre sí`. También promueve el acoplamiento flojo al evitar que los objetos se refieran explícitamente entre sí. Y le permite variar su interacción de forma independiente.

Uso del Patron Iterator

- El ejemplo clásico y más adecuado del Patrón de diseño de mediador sería una sala de chat tipo Google Meet donde sus componentes (personas más probables) pueden entrar y salir del sistema en cualquier momento.
- Por lo tanto, no tiene sentido que los diferentes participantes tengan referencias directas entre sí porque esas referencias pueden desaparecer en cualquier momento.
- Entonces, la solución aquí es hacer que todos los componentes se refieran a algún tipo de componente central que facilite la comunicación y que ese componente sea el mediador.

Codigo en C++

```
#include <string>
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
class ChatRoom {
    public:
    // Transmitir
    virtual void broadcast(string from, string msg)= 0;
};

class Person : public ChatRoom{
        public:
        string              name;
        ChatRoom*           room{nullptr};

        void broadcast(string , string ){};
        Person(string _name) { this->name = _name;}
        void say(string msg) { room->broadcast(name, msg);}
        void receive(string from, string msg) {
            string s{from + ": \"" + msg + "\""};
            cout << "[Chat de " << name << "]" << s << "\n";
        }
    };

class GoogleMeet : public ChatRoom
{
    public:
    vector<Person*>     people;
    // Transmitir
    void broadcast(string from, string msg) {
        for (auto p : people)
            if (p->name != from)
                p->receive(from, msg);
    }
    // Unen al chat
    void join(Person *p) {
        string join_msg = p->name + " se unio al chat";
        broadcast("room", join_msg);
        p->room = this;
        people.push_back(p);
    }
};

int main() {
    GoogleMeet room;

    Person john("John");
    Person jane("Jane");

    room.join(&john);
    room.join(&jane);
```

```
    john.say("Jane hiciste la tarea?");
    jane.say("oh, hey john");

    Person simon("Simon");
    room.join(&simon);
    simon.say("Hola a todos!");

    return 0;
}
```

Beneficios del patron de diseño Meadiator

- Puede reemplazar cualquier componente en el sistema sin afectar a otros componentes y sistema.
- El patrón de diseño de mediador reduce la complejidad de la comunicación entre los diferentes componentes de un sistema. Promoviendo así el acoplamiento flojo y menos número de subclases .
- Para superar la limitación del Patrón de diseño del observador que funciona en una relación de uno a muchos, el Patrón de diseño de mediador puede emplearse para una relación de muchos a muchos.

Mediator vs Facade

El patrón de Mediator puede verse como un patrón de Facade multiplexada. En el Mediator, en lugar de trabajar con una interfaz de un solo objeto, está creando una interfaz multiplexada entre varios objetos para proporcionar transiciones suaves.

Mediator vs Observer

- Observer = relacion de uno a muchos
- Mediator = relacion muchos a muchos

Debido al control centralizado de la comunicación, el mantenimiento del sistema diseñado con el patrón de diseño de Mediator es fácil.