



PATRÓN MEDIATOR

Angélica Torres Velderrain 359628





Es un patrón de diseño de comportamiento que te permite reducir las dependencias caóticas entre objetos. Este patrón restringe las comunicaciones directas entre los objetos, forzándolos a colaborar únicamente a través de un objeto mediador.

Su objetivo principal es promover la comunicación indirecta entre los componentes, lo que mejora el modularidad y facilita la mantenibilidad del código.





En un sistema que utiliza el patrón mediator, un elemento central, conocido como el "mediador", actúa como un intermediario que coordina la comunicación entre los diferentes componentes o subsistemas. En lugar de que los componentes se comuniquen directamente entre sí, se comunican a través del mediador. Esto reduce el acoplamiento entre los componentes y hace que sea más fácil agregar, eliminar o modificar componentes sin afectar significativamente al resto del sistema.

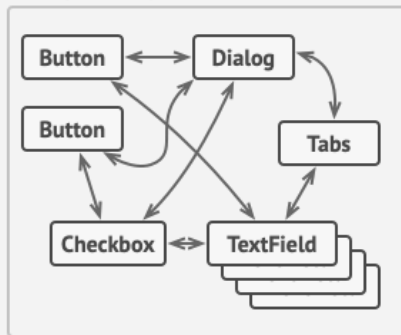


Problema

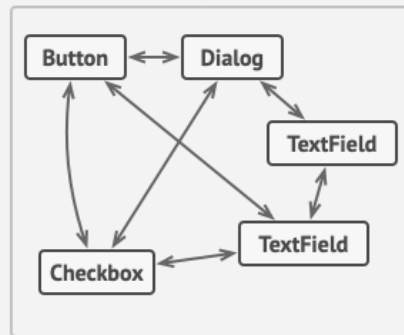
Digamos que tienes un diálogo para crear y editar perfiles de cliente el cual consiste en varios controles de formulario, como campos de texto, casillas, botones, etc.



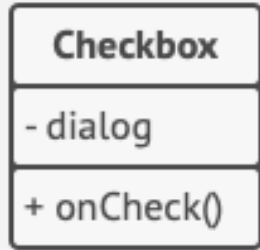
Diálogo de Perfil



Diálogo de Inicio de Sesión



Algunos de los elementos del formulario pueden interactuar con otros. Por ejemplo, al seleccionar la casilla “tengo un perro” puede aparecer un campo de texto oculto para introducir el nombre del perro.



```
if (dialog.name == "profile_form")  
    // ...  
if (dialog.name == "login_form")  
    // ...
```

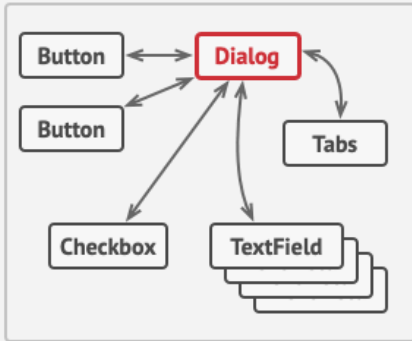
Al implementar esta lógica directamente dentro del código, haces que las clases de estos elementos sean mucho más difíciles de reutilizar en otros formularios de la aplicación. Por ejemplo, no podrás utilizar la clase de la casilla dentro de otro formulario porque está acoplada al campo de texto del perro.

Solución

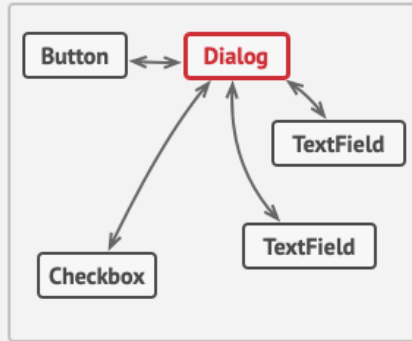
El patrón mediator sugiere que detengas toda comunicación directa entre los componentes que quieres hacer independientes entre sí. En lugar de ello, estos componentes deberán colaborar indirectamente, invocando un objeto mediador especial que redireccione las llamadas a los componentes adecuados. Como resultado, los componentes dependen únicamente de una sola clase mediadora, en lugar de estar acoplados a decenas de sus colegas.

En el ejemplo, la clase de diálogo puede actuar como mediadora. Lo más probable es que la clase de diálogo conozca ya todas sus subelementos, por lo que ni siquiera será necesario que introduzcas nuevas dependencias en esta clase.

Diálogo de Perfil

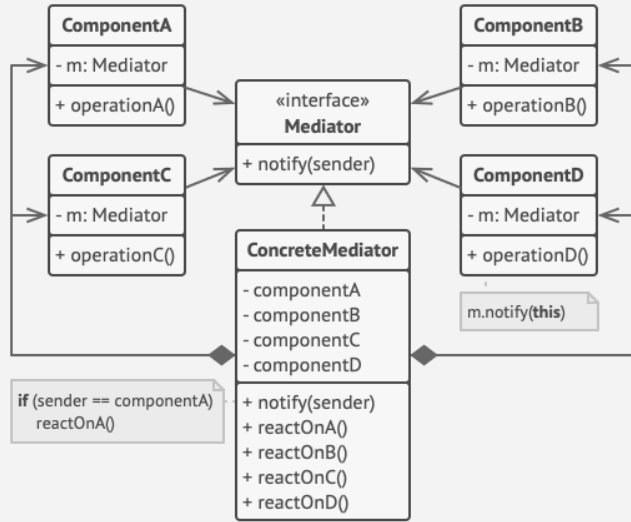


Diálogo de Inicio de Sesión



De este modo, el patrón mediator te permite encapsular una compleja red de relaciones entre varios objetos dentro de un único objeto mediador. Cuantas menos dependencias tenga una clase, más fácil es modificar, extender o reutilizar esa clase.

Estructura




1. Los Componentes son varias clases que contienen cierta lógica de negocio. Cada componente tiene una referencia a una interfaz mediadora, declarada con su tipo.
2. La interfaz Mediadora declara métodos de comunicación con los componentes. Los componentes pueden pasar cualquier contexto como argumentos de este método, pero sólo de tal forma que no haya acoplamiento entre un componente receptor y la clase del emisor.
3. Los Mediadores Concretos encapsulan las relaciones entre varios componentes. Estos a menudo mantienen referencias a todos los componentes que gestionan y en ocasiones gestionan incluso su ciclo de vida.
4. Los componentes no deben conocer otros componentes. Si le sucede algo importante a un componente, sólo debe notificar a la interfaz mediadora. Cuando la mediadora recibe la notificación, puede identificar fácilmente al emisor, lo cual puede ser suficiente para decidir qué componente debe activarse en respuesta.



pros y contras

Pros	Contras
<ul style="list-style-type: none">• Principio de responsabilidad única: puedes extraer las comunicaciones entre varios componentes en un solo lugar, lo que facilita su comprensión y mantenimiento.• Principio de abierto/cerrado: puedes introducir nuevos mediadores sin tener que cambiar los componentes reales.• Puedes reducir el acoplamiento entre varios componentes de un programa.• Puedes reutilizar componentes individuales de forma fácil.	<ul style="list-style-type: none">• Con el tiempo, un mediador puede evolucionar hasta convertirse en un objeto de dios.



```
import java.util.ArrayList;
import java.util.List;

class User{
    private String name;

    public User(String name){
        this.name = name;
    }

    public void sendMessage(String message){
        ChatRoom.showMessage(this, message);
    }

    public String getName(){
        return name;
    }
}

class ChatRoom{
    private static List<User> users = new ArrayList<>();

    public static void addUser(User user){
        users.add(user);
    }

    public static void showMessage(User user, String message){
        System.out.println(user.getName() + " says: " + message);
    }
}

public class Main{
    public static void main(String[] args){
        User alice = new User("Alice");
        User bob = new User("Bob");

        ChatRoom.addUser(alice);
        ChatRoom.addUser(bob);

        alice.sendMessage("Hi, Bob!");
        bob.sendMessage("Hello, Alice!");
    }
}
```

Ejercicio

Implemente el patrón mediator en el siguiente código:

