**Propósito**

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

**Problema**

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

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. Otro ejemplo es el botón de envío que tiene que validar los valores de todos los campos antes de guardar la información.

Al implementar esta lógica directamente dentro del código de los elementos del formulario, 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. O bien podrás utilizar todas las clases implicadas en representar el formulario de perfil, o no podrás usar ninguna en absoluto.

**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 nuestro ejemplo del formulario de edición de perfiles, la propia clase de diálogo puede actuar como mediadora. Lo más probable es que la clase de diálogo conozca ya todos sus subelementos, por lo que ni siquiera será necesario que introduzcas nuevas dependencias en esta clase.

El cambio más significativo lo sufren los propios elementos del formulario. Pensemos en el botón de envío. Antes, cada vez que un usuario hacía clic en el botón, tenía que validar los valores de todos los elementos individuales del formulario. Ahora su único trabajo consiste en notificar al diálogo acerca del clic. Al recibir esta notificación, el propio diálogo realiza las validaciones o pasa la tarea a los elementos individuales. De este modo, en lugar de estar ligado a una docena de elementos del formulario, el botón solo es dependiente de la clase diálogo.

Puedes ir más lejos y reducir en mayor medida la dependencia extrayendo la interfaz común para todos los tipos de diálogo. La interfaz declarará el método de notificación que pueden utilizar todos los elementos del formulario para notificar al diálogo sobre los eventos que le suceden a estos elementos. Por lo tanto, ahora nuestro botón de envío debería poder funcionar con cualquier diálogo que implemente esa interfaz.

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.

**Aplicabilidad**

Utiliza el patrón Mediator cuando resulte difícil cambiar algunas de las clases porque están fuertemente acopladas a un puñado de otras clases.

 El patrón te permite extraer todas las relaciones entre clases dentro de una clase separada, aislando cualquier cambio en un componente específico, del resto de los componentes.

 Utiliza el patrón cuando no puedas reutilizar un componente en un programa diferente porque sea demasiado dependiente de otros componentes.

 Una vez aplicado el patrón Mediator, los componentes individuales no conocen los otros componentes. Todavía pueden comunicarse entre sí, aunque indirectamente, a través del objeto mediador. Para reutilizar un componente en una aplicación diferente, debes darle una nueva clase mediadora.

 Utiliza el patrón Mediator cuando te encuentres creando cientos de subclases de componente sólo para reutilizar un comportamiento básico en varios contextos.

 Debido a que todas las relaciones entre componentes están contenidas dentro del mediador, resulta fácil definir formas totalmente nuevas de colaboración entre estos componentes introduciendo nuevas clases mediadoras, sin tener que cambiar los propios componentes.

**Pros**

* Principio de responsabilidad única. Puedes extraer las comunicaciones entre varios componentes dentro de un único sitio, haciéndolo más fácil de comprender y mantener.
* Principio de abierto/cerrado. Puedes introducir nuevos mediadores sin tener que cambiar los propios componentes.
* Puedes reducir el acoplamiento entre varios componentes de un programa.
* Puedes reutilizar componentes individuales con mayor facilidad.

## Contras

* Con el tiempo, un mediador puede evolucionar a un [**objeto todopoderoso**](https://refactoring.guru/es/antipatterns/god-object).