refactoring.guru

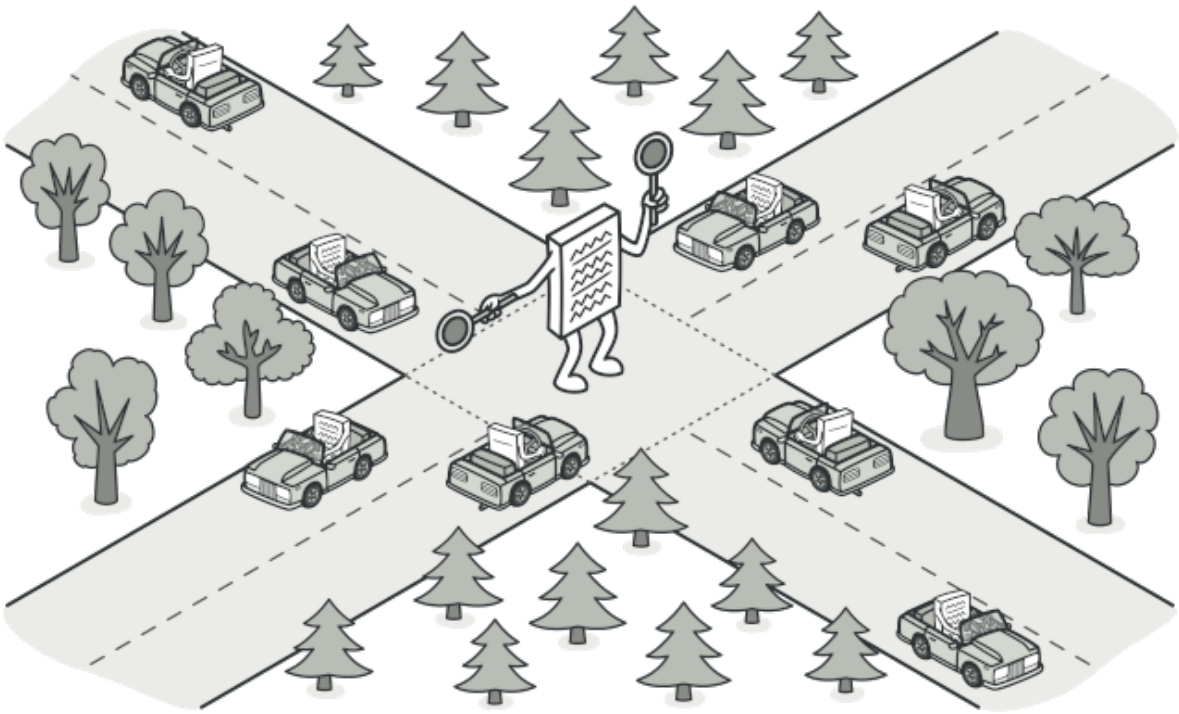
Mediator

15–21 minutes

Também conhecido como:
Mediador, Intermediário, Intermediary, Controlador, Controller

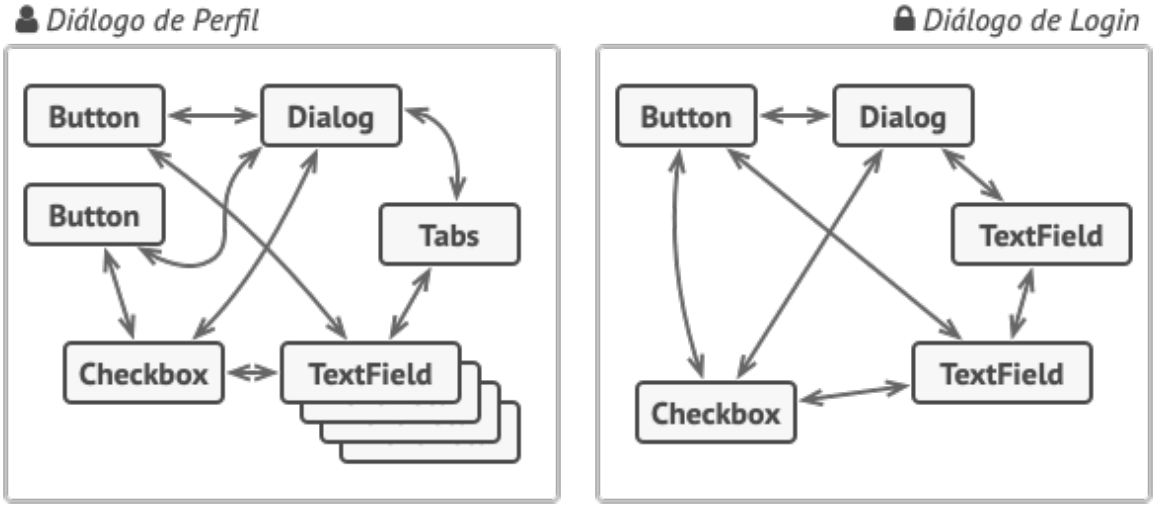
Propósito

O **Mediator** é um padrão de projeto comportamental que permite que você reduza as dependências caóticas entre objetos. O padrão restringe comunicações diretas entre objetos e os força a colaborar apenas através do objeto mediador.



Problema

Digamos que você tem uma caixa de diálogo para criar e editar perfis de clientes. Ela consiste em vários controles de formulário tais como campos de texto, caixas de seleção, botões, etc.



As relações entre os elementos da interface de usuário podem se tornar caóticas a medida que a aplicação evolui.

Alguns dos elementos do formulário podem interagir com outros. Por exemplo, selecionando a caixa de “Eu tenho um cão” pode revelar uma caixa de texto escondida para inserir o nome do cão. Outro exemplo é o botão enviar que tem que validar todos os campos antes de salvar os dados.

Os elementos podem ter várias relações com outros elementos. Portanto, mudanças a alguns elementos podem afetar os outros.

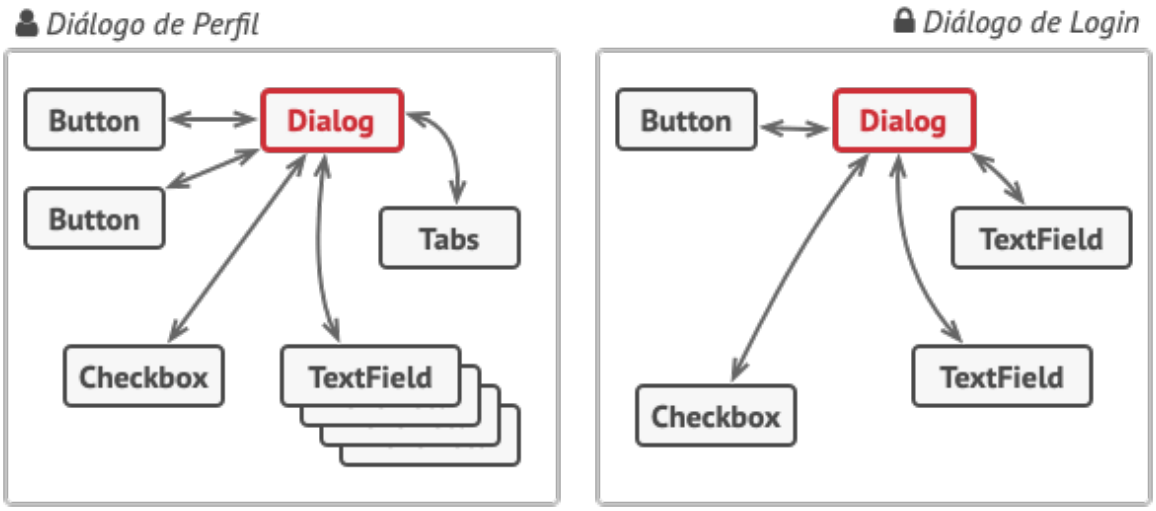
Ao ter essa lógica implementada diretamente dentro do código dos elementos de formulários você torna as classes dos elementos muito difíceis de se reutilizar em outros formulários da aplicação. Por exemplo, você não será capaz de usar aquela

classe de caixa de seleção dentro de outro formulário porque ela está acoplado com o campo de texto do nome do cão. Você pode ter ou todas as classes envolvidas na renderização do formulário de perfil, ou nenhuma.

Solução

O padrão Mediator sugere que você deveria cessar toda comunicação direta entre componentes que você quer tornar independentes um do outro. Ao invés disso, esses componentes devem colaborar indiretamente, chamando um objeto mediador especial que redireciona as chamadas para os componentes apropriados. Como resultado, os componentes dependem apenas de uma única classe mediadora ao invés de serem acoplados a dúzias de outros colegas.

No nosso exemplo com o formulário de edição de perfil, a classe diálogo em si pode agir como mediadora. O mais provável é que a classe diálogo já esteja ciente de todos seus sub-elementos, então você não precisa introduzir novas dependências nessa classe.



Os elementos UI devem se comunicar indiretamente, através do

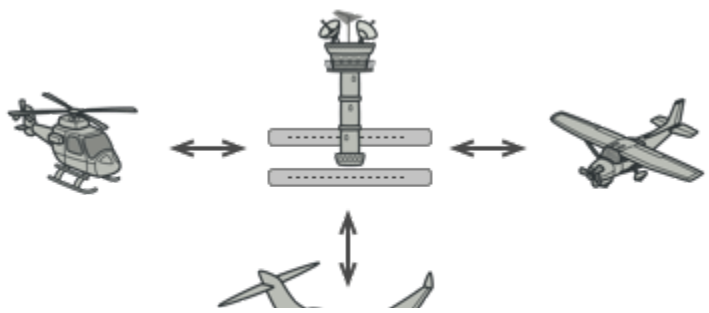
objeto mediador.

A mudança mais significativa acontece com os próprios elementos do formulário. Vamos considerar o botão de enviar. Antes, cada vez que um usuário clicava no botão, ele teria que validar os valores de todos os elementos de formulário. Agora seu único trabalho é notificar a caixa de diálogo sobre o clique. Ao receber essa notificação, a própria caixa de diálogo realiza as validações ou passa a tarefa para os elementos individuais. Portanto, ao invés de estar amarrado a uma dúzia de elementos de formulário, o botão está dependente apenas da classe diálogo.

Você pode ir além e fazer a dependência ainda mais frouxa extraindo a interface comum de todos os tipos de caixas de diálogo. A interface deve declarar o método de notificação que todos os elementos do formulário podem usar para notificar a caixa de diálogo sobre eventos acontecendo a aqueles elementos. Portanto, nosso botão enviar deve agora ser capaz de trabalhar com qualquer caixa de diálogo que implemente aquela interface.

Dessa forma, o padrão Mediator permite que você encapsule uma complexa rede de relações entre vários objetos em apenas um objeto mediador. Quanto menos dependências uma classe tenha, mais fácil essa classe se torna para se modificar, estender, ou reutilizar.

Analogia com o mundo real



Pilotos de aeronaves não falam entre si diretamente na hora de decidir quem é o próximo a aterrissar seu avião. Toda a comunicação passa pela torre de controle.

Os pilotos de aeronaves que se aproximam ou partem da área de controle do aeroporto não se comunicam diretamente entre si. Ao invés disso falam com um controlador de tráfego aéreo, que está sentando em uma torre alta perto da pista de aterrissagem. Sem o controlador do tráfego aéreo os pilotos precisariam estar cientes de cada avião nas redondezas do aeroporto, discutindo as prioridades de aterrissagem com um comitê de dúzias de outros pilotos. Isso provavelmente aumentaria em muito as estatísticas de acidentes aéreos.

A torre não precisa fazer o controle de todo o voo. Ela existe apenas para garantir o condicionamento da área do terminal devido ao número de pessoas envolvidas ali, o que poderia ser demais para um piloto.

Estrutura

1. Os **Componentes** são várias classes que contém alguma lógica de negócio. Cada componente tem uma referência a um mediador, declarada com o tipo de interface do mediador. O componente não está ciente da classe atual do mediador, então você pode reutilizar o componente em outros programas ao ligá-lo com um mediador diferente.
2. A interface do **Mediador** declara métodos de comunicação com os componentes, os quais geralmente incluem apenas um método

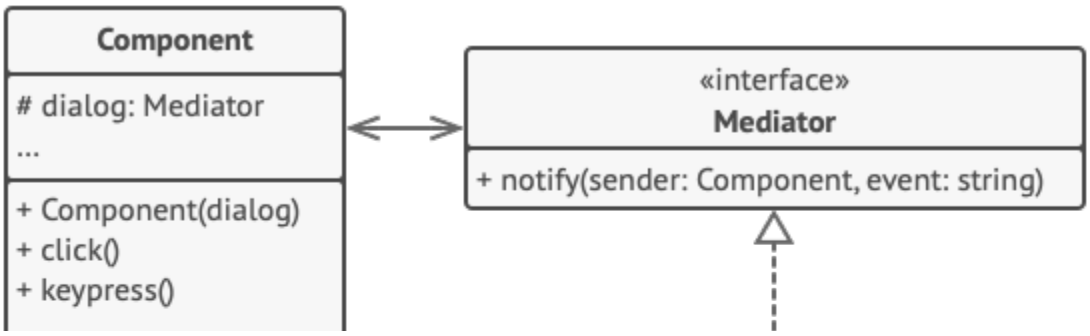
de notificação. Os componentes podem passar qualquer contexto como argumentos desse método, incluindo seus próprios objetos, mas apenas de tal forma que nenhum acoplamento ocorra entre um componente destinatário e a classe remetente.

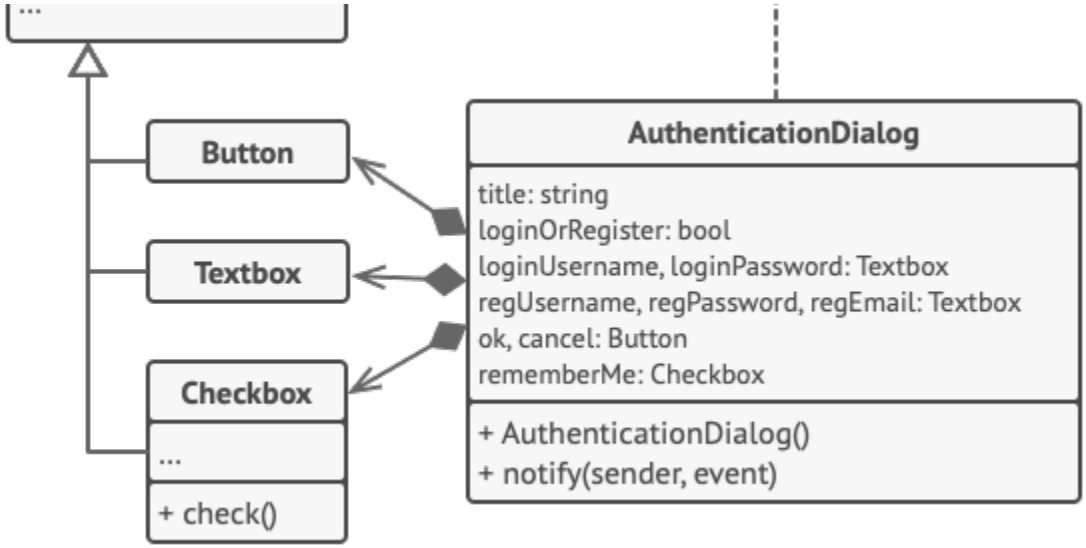
3. Os **Mediadores Concretos** encapsulam as relações entre vários componentes. Os mediadores concretos quase sempre mantém referências de todos os componentes os quais gerenciam e, algumas vezes, até gerenciam o ciclo de vida deles.
4. Componentes não devem estar cientes de outros componentes. Se algo importante acontece dentro ou para um componente, ele deve apenas notificar o mediador. Quando o mediador recebe a notificação, ele pode facilmente identificar o remetente, o que é suficiente para decidir que componente deve ser acionado em retorno.

   Da perspectiva de um componente, tudo parece como uma caixa preta. O remetente não sabe quem vai acabar lidando com o seu pedido, e o destinatário não sabe quem enviou o pedido em primeiro lugar.

Pseudocódigo

Neste exemplo, o padrão **Mediator** ajuda você a eliminar dependências mútuas entre várias classes UI: botões, caixas de seleção, e textos de rótulos.





Estrutura das classes UI caixa de diálogo.

Um elemento, acionado por um usuário, não se comunica com outros elementos diretamente, mesmo que pareça que ele deva fazer isso. Ao invés disso, o elemento apenas precisa fazer o mediador saber do evento, passando qualquer informação de contexto junto com a notificação.

Neste exemplo, todas caixas de diálogo de autenticação agem como o mediador. Elas sabem quais os elementos concretos devem colaborar e facilita sua comunicação indireta. Ao receber a notificação de um evento, a caixa de diálogo decide que elemento deve lidar com o evento e redireciona a chamada de acordo.

```
// A interface mediadora declara um método usado pelos
// componentes para notificar o mediador sobre vários eventos. O
// mediador pode reagir a esses eventos e passar a execução
para
// outros componentes.
interface Mediator is
    method notify(sender: Component, event: string)
```

```
// A classe mediadora concreta. A rede entrelaçada de conexões
// entre componentes individuais foi desentrelaçada e movida
// para dentro do mediador.
class AuthenticationDialog implements Mediator is
    private field title: string
    private field loginOrRegisterChkBx: Checkbox
    private field loginUsername, loginPassword: Textbox
    private field registrationUsername, registrationPassword,
                  registrationEmail: Textbox
    private field okBtn, cancelBtn: Button

    constructor AuthenticationDialog() is
        // Cria todos os objetos componentes e passa o atual
        // mediador em seus construtores para estabelecer links.

    // Quando algo acontece com um componente, ele notifica o
    // mediador. Ao receber a notificação, o mediador pode fazer
    // alguma coisa por conta própria ou passar o pedido para
    // outro componente.
    method notify(sender, event) is
        if (sender == loginOrRegisterChkBx and event == "check")
            if (loginOrRegisterChkBx.checked)
                title = "Log in"
                // 1. Mostra componentes de formulário de login.
                // 2. Esconde componentes de formulário de
                // registro.
            else
                title = "Register"
                // 1. Mostra componentes de formulário de
```

```
            // registro.
            // 2. Esconde componentes de formulário de
            // login.
        if (sender == okBtn && event == "click")
            if (loginOrRegister.checked)
                // Tenta encontrar um usuário usando as
                // credenciais de login.
                if (!found)
                    // Mostra uma mensagem de erro acima do
                    // campo login.
            else
                // 1. Cria uma conta de usuário usando dados dos
                // campos de registro.
                // 2. Loga aquele usuário.


// Os componentes se comunicam com o mediador usando a
interface
// do mediador. Graças a isso, você pode usar os mesmos
// componentes em outros contextos ao ligá-los com diferentes
// objetos mediadores.
class Component is
    field dialog: Mediator

    constructor Component(dialog) is
        this.dialog = dialog

    method click() is
        dialog.notify(this, "click")
```

```
    method keypress() is
        dialog.notify(this, "keypress")

// Componentes concretos não falam entre si. Eles têm apenas um
// canal de comunicação, que é enviar notificações para o
// mediador.
class Button extends Component is


class Textbox extends Component is


class Checkbox extends Component is
    method check() is
        dialog.notify(this, "check")
```

Aplicabilidade

Utilize o padrão Mediator quando é difícil mudar algumas das classes porque elas estão firmemente acopladas a várias outras classes.

O padrão lhe permite extrair todas as relações entre classes para uma classe separada, isolando quaisquer mudanças para um componente específico do resto dos componentes.

Utilize o padrão quando você não pode reutilizar um componente em um programa diferente porque ele é muito dependente de outros componentes.

Após você aplicar o Mediator, componentes individuais se tornam

alheios aos outros componentes. Eles ainda podem se comunicar entre si, mas de forma indireta, através do objeto mediador. Para reutilizar um componente em uma aplicação diferente, você precisa fornecer a ele uma nova classe mediadora.

Utilize o Mediator quando você se encontrar criando um monte de subclasses para componentes apenas para reutilizar algum comportamento básico em vários contextos.

Como todas as relações entre componentes estão contidas dentro do mediador, é fácil definir novas maneiras para esses componentes colaborarem introduzindo novas classes mediadoras, sem ter que mudar os próprios componentes.

Como implementar

1. Identifique um grupo de classes firmemente acopladas que se beneficiariam de estar mais independentes (por exemplo, para uma manutenção ou reutilização mais fácil dessas classes).
2. Declare a interface do mediador e descreva o protocolo de comunicação desejado entre os mediadores e os diversos componentes. Na maioria dos casos, um único método para receber notificações de componentes é suficiente.

   Essa interface é crucial quando você quer reutilizar classes componente em diferentes contextos. Desde que o componente trabalhe com seu mediador através da interface genérica, você pode ligar o componente com diferentes implementações do mediador.
3. Implemente a classe concreta do mediador. Essa classe se beneficia por armazenar referências a todos os componentes que gerencia.

4. Você pode ainda ir além e fazer que o mediador fique responsável pela criação e destruição de objetos componente. Após isso, o mediador pode montar uma fábrica ou uma fachada.
5. Componentes devem armazenar uma referência ao objeto do mediador. A conexão é geralmente estabelecida no construtor do componente, onde o objeto mediador é passado como um argumento.
6. Mude o código dos componentes para que eles chamem o método de notificação do mediador ao invés de métodos de outros componentes. Extraia o código que envolve chamar os outros componentes para a classe do mediador. Execute esse código sempre que o mediador receba notificações daquele componente.

Prós e contras

- *Princípio de responsabilidade única*. Você pode extrair as comunicações entre vários componentes para um único lugar, tornando as de mais fácil entendimento e manutenção.
- *Princípio aberto/fechado*. Você pode introduzir novos mediadores sem ter que mudar os próprios componentes.
- Você pode reduzir o acoplamento entre os vários componentes de um programa.
- Você pode reutilizar componentes individuais mais facilmente.
- Com o tempo um mediador pode evoluir para um Objeto Deus.

Relações com outros padrões

- O Chain of Responsibility, Command, Mediator e Observer abrangem várias maneiras de se conectar remetentes e

destinatários de pedidos:

- O *Chain of Responsibility* passa um pedido sequencialmente ao longo de um corrente dinâmica de potenciais destinatários até que um deles atua no pedido.
- O *Command* estabelece conexões unidirecionais entre remetentes e destinatários.
- O *Mediator* elimina as conexões diretas entre remetentes e destinatários, forçando-os a se comunicar indiretamente através de um objeto mediador.
- O *Observer* permite que destinatários inscrevam-se ou cancelem sua inscrição dinamicamente para receber pedidos.
- O Facade e o Mediator têm trabalhos parecidos: eles tentam organizar uma colaboração entre classes firmemente acopladas.
- O *Facade* define uma interface simplificada para um subsistema de objetos, mas ele não introduz qualquer nova funcionalidade. O próprio subsistema não está ciente da fachada. Objetos dentro do subsistema podem se comunicar diretamente.
- O *Mediator* centraliza a comunicação entre componentes do sistema. Os componentes só sabem do objeto mediador e não se comunicam diretamente.
- A diferença entre o Mediator e o Observer é bem obscura. Na maioria dos casos, você pode implementar qualquer um desses padrões; mas às vezes você pode aplicar ambos simultaneamente. Vamos ver como podemos fazer isso.

  O objetivo primário do *Mediator* é eliminar dependências múltiplas entre um conjunto de componentes do sistema. Ao invés disso, esses componentes se tornam dependentes de um único objeto

mediador. O objetivo do *Observer* é estabelecer comunicações de uma via dinâmicas entre objetos, onde alguns deles agem como subordinados de outros.

Existe uma implementação popular do padrão Mediator que depende do *Observer*. O objeto mediador faz o papel de um publicador, e os componentes agem como assinantes que inscrevem-se ou removem a inscrição aos eventos do mediador. Quando o *Mediator* é implementado dessa forma, ele pode parecer muito similar ao *Observer*.

Quando você está confuso, lembre-se que você pode implementar o padrão Mediator de outras maneiras. Por exemplo, você pode ligar permanentemente todos os componentes ao mesmo objeto mediador. Essa implementação não se parece com o *Observer* mas ainda irá ser uma instância do padrão Mediator.

Agora imagine um programa onde todos os componentes se tornaram publicadores permitindo conexões dinâmicas entre si. Não haverá um objeto mediador centralizado, somente um conjunto distribuído de observadores.