Mediator

Padrões Comportamentais

O padrão *Mediator* é um padrão de projeto que serve para encapsular a maneira que um conjunto de objetos interage, ou seja, a comunicação entre os objetos é estabelecida através de um objeto mediador (*Mediator*). Este padrão de projeto promove o fraco acoplamento ao evitar que objetos se refiram uns aos outros de forma explícita e permite variar suas intenções independentemente.

Motivação (Por que utilizar?)

O paradigma de desenvolvimento de *software* orientado a objetos incentiva a distribuição do comportamento entre objetos. Essa distribuição pode resultar em uma estrutura onde objetos possuem muitas conexões entre si, na pior das hipóteses, todo objeto acaba conhecendo um ao outro.

O particionamento de um sistema em muitos objetos visa melhorar a capacidade de reutilização de código, porém, a proliferação de interconexões entre estes objetos tende a voltar a reduzir o reúso de código. Muitas interconexões tornam menos provável que um objeto possa funcionar corretamente sem o apoio de outros, desta forma o sistema passa a agir como se fosse monolítico.

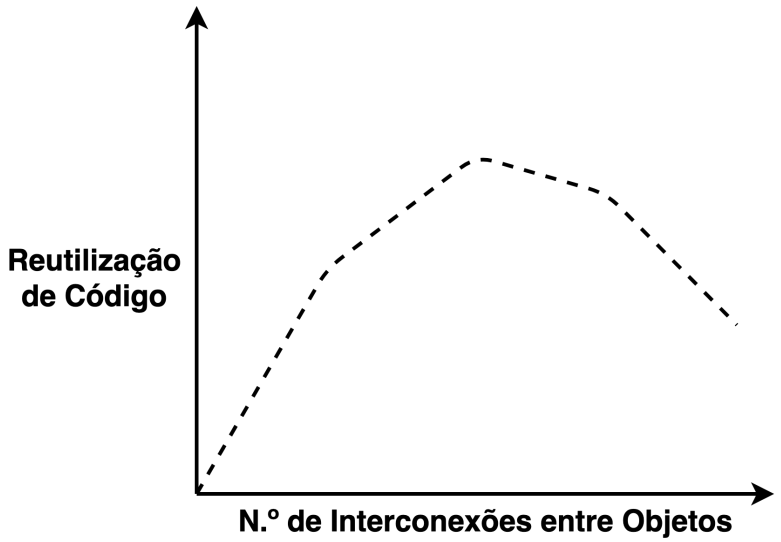
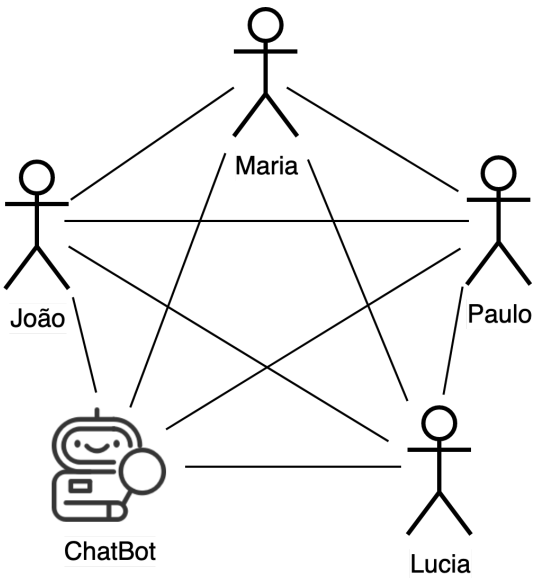


Ilustração da queda da reutilização de código conforme o número de interconexões cresce.

O padrão *Mediator* tem como objetivo inibir a necessidade de tantas interconexões entre objetos. Vamos utilizar um exemplo para ilustrar uma utilização do *Mediator.* Considere que precisamos criar um chat de discussão onde:

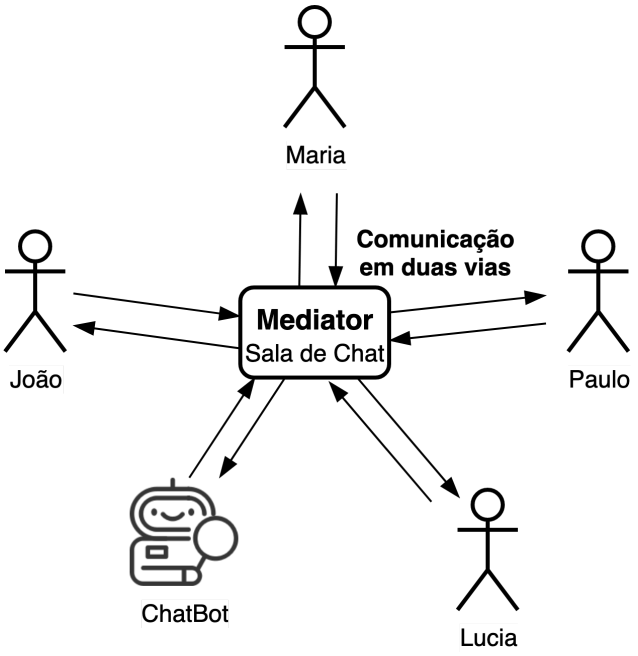
- Podem existir várias salas de chat.
- Uma sala de chat poderá ter vários usuários e eles poderão ser adicionados e removidos dela dinamicamente.
- Um usuário poderá fazer parte de apenas uma sala de chat por vez.
- Sempre que um usuário enviar uma mensagem, os demais usuários da sala de chat devem recebê-la.
- Sempre existirá um *ChatBot* que será o moderador de uma sala de chat. Cada sala terá seu próprio *ChatBot* e não poderá existir sem ele.
- Por ser um chat de discussão um usuário não poderá enviar duas mensagens consecutivas, se fizer isso, as mensagens consecutivas serão interceptadas pelo chatbot e serão negadas.

Para enviar uma mensagem para todos os usuários em uma sala de chat, um determinado usuário teria que conhecer todos os demais e ainda conhecer o *ChatBot* moderador da sala.



Relacionamento muitos-para-muitos direto nos usuários

Quanto mais usuários estiverem no chat, mais ligações serão necessárias entre eles e maior será o acoplamento. Para contornar isso, podemos utilizar um mediador (*Mediator*). Deste modo os usuários não irão mais se comunicar de forma direta, na verdade nem saberão que os outros usuários existem, pois só conhecerão e se comunicarão com o mediador.



Relacionamento um-para-muitos entre Usuários e *Mediator*

Nota: A essa altura talvez você tenha percebido que este parece ser um problema clássico para a aplicação do padrão ***Observer***. E você está certo(a), *Mediador* e *Observer* são padrões concorrentes. A diferença entre eles é que o *Observer* distribui a comunicação introduzindo objetos "*observer*" e "*subject*", enquanto o padrão *Mediador* encapsula a comunicação entre outros objetos. O que determina qual é a melhor escolha é o contexto do problema que precisa ser resolvido.

Vejamos algumas características interessantes do nosso problema, como elas são representadas no diagrama de classes e no código a seguir:

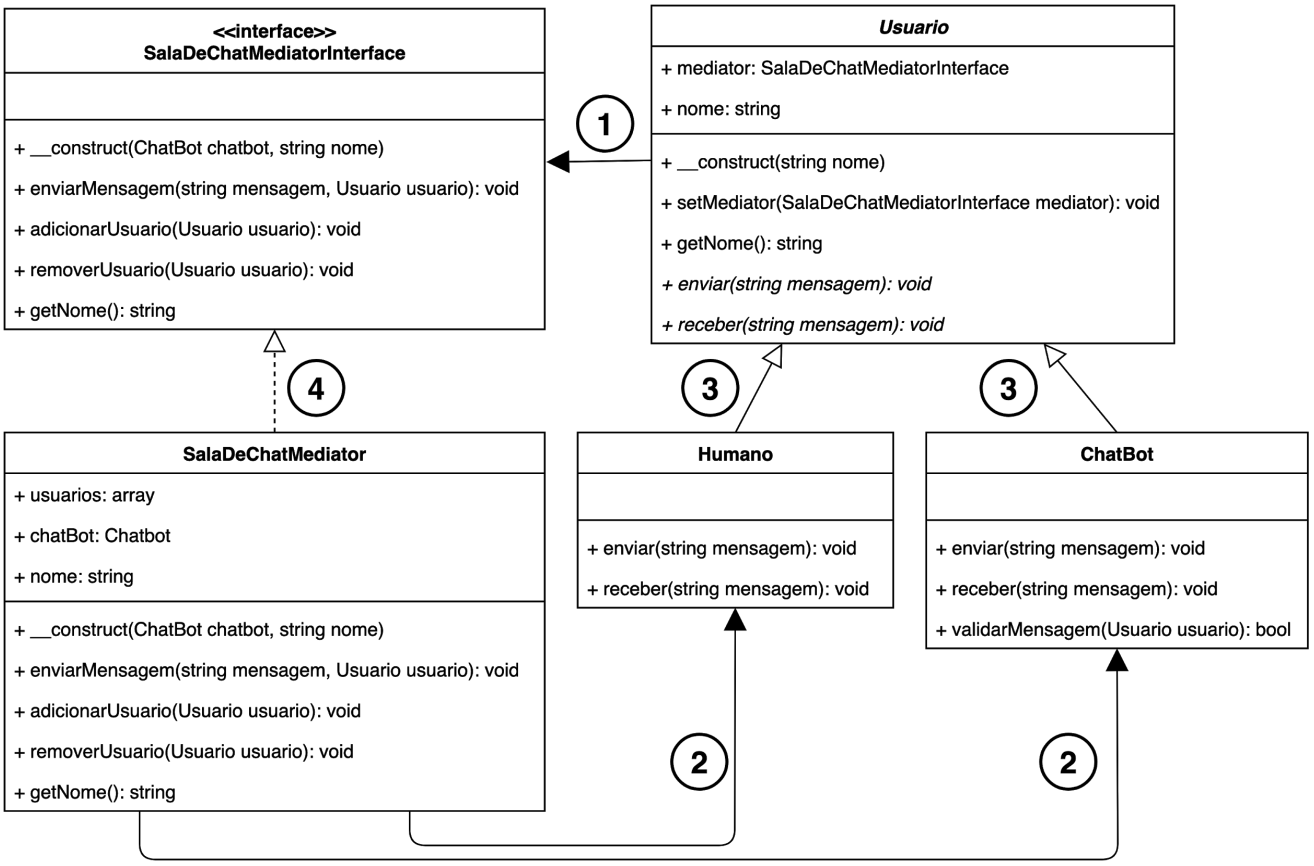


Diagrama de classes do sistema de chat

1. Um usuário deve saber em qual sala de chat ele está, precisa manter uma referência a ela. **Uma sala de chat é um *Mediator*;**

2. Para ser capaz de enviar mensagens a todos os usuários, o *Mediator* precisa conhecer e manter referência a todos eles.

3. Para aceitar um novo usuário, o *Mediator* precisa saber se ele é capaz de enviar e receber mensagens. existem dois tipos de usuários: `Humano` e `ChatBot`, então, para que o *Mediator* possa aceitar ambos, eles precisam ter um supertipo em comum. Vamos chamar este supertipo de `Usuario`.

4. Todo *Mediator* (sala de chat) precisa ser capaz de:

    - Aceitar (adicionar) novos usuários.
    - Remover usuários.
    - Enviar a mensagem de um usuário aos demais usuários, sejam eles do tipo `Humano` ou `ChatBot`.

    Para que um `Usuario` aceite um *Mediator* ele precisa ter a garantia que as condições acima serão supridas por ele. Precisamos então de uma interface.

Vamos começar a implementação pela interface do *Mediator*, ela será chamada de **SalaDeChatMediatorInterface.** Essa interface garante aos objetos do tipo **Usuario** as funcionalidade esperadas em uma sala de chat, ou seja, um *Mediator.*

```
interface SalaDeChatMediatorInterface
{
    /*Toda sala de chat precisa de um ChatBot Moderador (ChatBot será definida a seguir)
      Para uma boa experiência do usuário vamos adicionar também um nome a sala de chat*/
    public function __construct(ChatBot $chatBot, string $nome);

    /*Recebe a mensagem e o Usuario que a enviou (Usuario será definida a seguir)
      Depois a repassa a todos os demais usuários da sala de chat*/
    public function enviarMensagem(string $mensagem, Usuario $usuario);

    //Adiciona um usuário a sala se chat (ao Mediator)
    public function adicionarUsuario(Usuario $usuario): void;

    //Remove um usuário da sala se chat (do Mediator)
    public function removerUsuario(Usuario $usuario): void;

    //Retorna o nome da sala de chat
    public function getNome(): string;
}
```

Os usuários da sala de chat terão coisas em comum, deste modo, para reaproveitar o código vamos criar uma classe abstrata **Usuario** ao invés de uma interface.

```php
abstract class Usuario
{
    /*Um usuário precisa conhecer o mediador que permitirá
      que se comunique com outros usuários. É inicializado como null*/
    protected ?SalaDeChatMediatorInterface $mediator = null;

    //Nome do usuário
    protected string $nome;

    //O nome do usuário é definido no momento de sua criação.
    public function __construct(string $nome)
    {
        $this->nome = $nome;
    }

    /*Define o mediador de chat que será utilizado
      para se comunicar com os demais usuários.*/
    public function setMediator(SalaDeChatMediatorInterface $mediator): void
    {
        /*Um usuário pode estar em apenas um mediator (sala de chat) por vez
          então, se seu mediator for diferente de null*/
        if (!is_null($this->mediator)) {
            /*O usuário pede para que o mediator o remova de sua lista de usuários
              (O usuário pede para ser removido da sala de chat que estava)*/
            $this->mediator->removerUsuario($this);
        }
        //Define seu Mediator (sala de chat)
        $this->mediator = $mediator;
        //Pede para que o mediador o adicione a sua lista de usuários (entrar na sala)
        $this->mediator->adicionarUsuario($this);
    }

    //Retorna o nome do usuário.
    public function getNome(): string
    {
        return $this->nome;
    }

    /*Método abstrato, será implementado por cada um dos usuários concretos
      conforme suas necessidades particulares.*/
    public abstract function enviar(string $mensagem): void;

    /*Método abstrato, será implementado por cada um dos usuários concretos
      conforme suas necessidades particulares.*/
    public abstract function receber(string $mensagem): void;
}
```

A classe **Humano** herdará as características da classe **Usuario** e terá apenas que implementar os métodos **enviar()** e **receber()**.

```
class Humano extends Usuario
{
    //Envia uma mensagem.
    public function enviar(string $mensagem): void
    {
        //Imprime em tela o nome da sala de chat em que se encontra
        echo '<br>----------- ' . $this->mediator->getNome() . ' -----------<br>';
        //Imprime que enviou a mensagem e o texto da mensagem
        echo $this->getNome() . ' <strong>enviou: </strong>: ' . $mensagem . '<br>';
        echo '----<br>';
        //Solicita ao Mediator (Sala de chat) que envie a mensagem aos demais usuários.
        $this->mediator->enviarMensagem($mensagem, $this);
    }

    //Recebe uma mensagem.
    public function receber(string $mensagem): void
    {
        //Imprime em tela que recebeu uma mensagem na sala de chat.
        echo $this->getNome() . ' <strong>recebeu: </strong>: ' . $mensagem . '<br>';
    }
}
```

Além de **enviar()** e **receber()** mensagens o **ChatBot** possui mais uma função, verificar se um usuário enviou mais de uma mensagem consecutiva. Se isso acontecer, ele precisa interceptar a mensagem e informar a todos os usuários que ela foi negada. Para que tal funcionalidade seja possível, a classe **ChatBot** guarda uma referência do último **Usuario** que enviou uma mensagem. Além disso existe também o método **validarMensagem()** que verifica se a mensagem recebida foi enviada pelo mesmo usuário que enviou a mensagem anterior.

```php
class ChatBot extends Usuario
{
    //Guarda referência ao último Usuario que enviou uma mensagem.
    private Usuario $ultimoUsuario;

    /*Sobrescreve o construtor de Usuario por dois motivos:
      - O nome do ChatBot sempre será CHATBOT;
      - Inicializa o $ultimoUsuario como sendo ele mesmo
        no momento que o ChatBot é criado nenhuma mensagem foi enviada na sala.*/
    public function __construct()
    {
        parent::__construct('CHATBOT');
        $this->ultimoUsuario = $this;
    }

    //Envia uma mensagem da mesma forma que Humano, apenas a formatação é diferente.
    public function enviar(string $mensagem): void
    {
        echo '## ' . $this->getNome() . ' <strong>enviou: </strong>: ' . $mensagem . ' ##<br>';
        echo '----<br>';
        $this->mediator->enviarMensagem($mensagem, $this);
    }

    //Recebe uma mensagem da mesma forma que Humano.
    public function receber(string $mensagem): void
    {
        echo $this->getNome() . ' <strong>recebeu: </strong>: ' . $mensagem . '<br>';
    }

    //Verifica se o usuário que enviou a mensagem atual é o mesmo que enviou a anterior.
    public function validarMensagem(Usuario $usuario): bool
    {
        //Se o usuário atual for igual ao anterior.
        if ($this->ultimoUsuario === $usuario) {
            //Imprime que a mensagem foi negada e retorna false;
            $this->enviar('Uma mensagem de ' . $usuario->getNome() . " foi negada!");
            return false;
        }
        //Senão, atualiza o último Usuario que enviou uma mensagem e retorna true
        $this->ultimoUsuario = $usuario;
        return true;
    }
}
```

Por fim vamos implementar o *Mediator* concreto, que será a classe SalaDeChatMediator:

```php
class SalaDeChatMediator implements SalaDeChatMediatorInterface
{
    //Mantém uma lista de usuários que precisam receber as mensagens da sala.
    private array $usuarios;
    //Mantém a referência ao ChatBot Moderador
    private ChatBot $chatBot;
    //Nome da sala de chat
    private string $nome;

    //Recebe o ChatBot moderador e o nome da sala em seu construtor.
    public function __construct(ChatBot $chatBot, string $nome)
    {
        //Inicializa a lista de usuários como vazia.
        $this->usuarios = [];
        //Adiciona a referência ao ChatBot moderador.
        $this->chatBot = $chatBot;
        //Informa ao ChatBot que ele deve moderar esta sala de chat.
        $this->chatBot->setMediator($this);
        //Define o nome da sala de chat
        $this->nome = $nome;
    }

    //Repassa uma mensagem recebida pelo mediator a todos os usuários.
    public function enviarMensagem(string $mensagem, Usuario $usuario): void
    {
        /*Solicita ao ChatBot a validação da mensagem. Trata-se da verificação
          de duas mensagens consecutivas sendo enviada por um mesmo Usuario.*/
        if ($this->chatBot->validarMensagem($usuario)) {
            foreach ($this->usuarios as $u) { //Para cada usuário na sala de chat.
                //A mensagem não deve ser recebida pelo usuário que a enviou
                if ($u != $usuario) {
                    $u->receber($mensagem); //Faz os usuários da lista receberem a mensagem.
                }
            }
            echo "---------------------------------<br>"; //Indica o fim dos envios
        }
    }

    //Adiciona um Usuario a lista de usuários que devem receber as mensagens
    public function adicionarUsuario(Usuario $usuario): void
    {
        $this->usuarios[] = $usuario;
    }

    //Remove um Usuario a lista de usuários que devem receber as mensagens
    public function removerUsuario(Usuario $usuario): void
    {
        foreach ($this->usuarios as $key => $u) {
            if ($u === $usuario) {
                unset($this->usuarios[$key]);
            }
        }
    }

    //Retorna o nome da sala de chat
    public function getNome(): string
    {
        return $this->nome;
    }
}
```

Vamos ao teste:

```
///========= Vamos criar a sala de chat 1 Vamos criar a sala de chat 1 ==========
//Criacão do chatbot mediador.
$chatBot1 = new ChatBot();
//Criacão da sala com atribuição do mediador e nome da sala.
$salaDeChat1 = new SalaDeChatMediator($chatBot1, 'Sala de chat 1');

// Mesmo processo para a sala de chat 2
$chatBot2 = new ChatBot();
$salaDeChat2 = new SalaDeChatMediator($chatBot2, 'Sala de chat 2');

//========= Vamos criar os usuários Humanos ==========
$joao  = new Humano("João");
$maria = new Humano("Maria");
$paulo = new Humano("Paulo");
$lucia = new Humano("Lucia");
$pedro = new Humano("Pedro");

//========= Vamos informar aos usuários qual é a sua sala de chat ==========
//Sala 1
$joao->setMediator($salaDeChat1);
$maria->setMediator($salaDeChat1);
$paulo->setMediator($salaDeChat1);

//Sala 2
$lucia->setMediator($salaDeChat2);
$pedro->setMediator($salaDeChat2);

//========= Envio de mensagens ==========
echo '<br>[Sala 1] João envia sua primeira mensagem'; //Instrução
$joao->enviar("Olá pessoal da sala 1, tudo bem?");

echo '<br>[Sala 2] Lucia envia sua primeira mensagem'; //Instrução
$lucia->enviar("Olá Pedro, tudo bem?");

echo '<br>[Sala 1] Paulo responde a João'; //Instrução
$paulo->enviar("Tudo bem comigo!");

echo '<br>[Sala 1] Paulo tenta enviar duas mensagens consecutivas'; //Instrução
$paulo->enviar("E com vc?");

echo '<br>[Sala 2] Pedro não responde, Lucia tentar enviar outra mensagem'; //Instrução
$lucia->enviar("Pedro, vc ta ai?");

echo '<br>Lucia troca de sala de chat';//Instrução
$lucia->setMediator($salaDeChat1);

echo '<br>[Sala 2] Lucia envia sua primeira mensagem na sala 2'; //Instrução
$lucia->enviar("Olá pessoa da sala 1, sou a Lucia. Tudo bem?");
```

Saída:

```
[Sala 1] João envia sua primeira mensagem
----------- Sala de chat 1 -----------
João enviou: : Olá pessoal da sala 1, tudo bem?
----
CHATBOT recebeu: : Olá pessoal da sala 1, tudo bem?
Maria recebeu: : Olá pessoal da sala 1, tudo bem?
Paulo recebeu: : Olá pessoal da sala 1, tudo bem?
---------------------------------

[Sala 2] Lucia envia sua primeira mensagem
----------- Sala de chat 2 -----------
Lucia enviou: : Olá Pedro, tudo bem?
----
CHATBOT recebeu: : Olá Pedro, tudo bem?
Pedro recebeu: : Olá Pedro, tudo bem?
---------------------------------

[Sala 1] Paulo responde a João
----------- Sala de chat 1 -----------
Paulo enviou: : Tudo bem comigo!
----
CHATBOT recebeu: : Tudo bem comigo!
João recebeu: : Tudo bem comigo!
Maria recebeu: : Tudo bem comigo!
---------------------------------

[Sala 1] Paulo tenta enviar duas mensagens consecutivas
----------- Sala de chat 1 -----------
Paulo enviou: : E com vc?
----
## CHATBOT enviou: : Uma mensagem de Paulo foi negada! ##
----
João recebeu: : Uma mensagem de Paulo foi negada!
Maria recebeu: : Uma mensagem de Paulo foi negada!
Paulo recebeu: : Uma mensagem de Paulo foi negada!
---------------------------------

[Sala 2] Pedro não responde, Lucia tentar enviar outra mensagem
----------- Sala de chat 2 -----------
Lucia enviou: : Pedro, vc ta ai?
----
## CHATBOT enviou: : Uma mensagem de Lucia foi negada! ##
----
Lucia recebeu: : Uma mensagem de Lucia foi negada!
Pedro recebeu: : Uma mensagem de Lucia foi negada!
---------------------------------

Lucia troca de sala de chat
[Sala 2] Lucia envia sua primeira mensagem na sala 2
----------- Sala de chat 1 -----------
Lucia enviou: : Olá pessoa da sala 1, sou a Lucia. Tudo bem?
----
CHATBOT recebeu: : Olá pessoa da sala 1, sou a Lucia. Tudo bem?
João recebeu: : Olá pessoa da sala 1, sou a Lucia. Tudo bem?
Maria recebeu: : Olá pessoa da sala 1, sou a Lucia. Tudo bem?
Paulo recebeu: : Olá pessoa da sala 1, sou a Lucia. Tudo bem?
---------------------------------
```

Aplicabilidade (Quando utilizar?)

- Quando um conjunto de objetos se comunica de maneiras bem definidas, porém complexas. As interdependências resultantes são desestruturadas e difíceis de entender.

- Quando reutilizar um objeto é difícil por estar fazendo referência e se comunicando com muitos outros objetos.

- Quando um comportamento distribuído entre várias classes deve ser personalizável sem que seja necessário criar muitas subclasses.

Componentes

- **Mediator:** Define uma interface para se comunicar com objetos do tipo *Colega*.

- **MediadorConcreto:**

    - Implementa o comportamento cooperativo coordenando os objetos do tipo Colega.

    - Conhece e mantém seus colegas.

- **Classes Colega:**

    - Cada classe do tipo Colega conhece o objeto Mediador que a coordena.

    - Sempre que um objeto do tipo Colega precisar se comunicar com outro objeto do tipo Colega, ele fará isso por meio de seu mediador (*Mediator*).

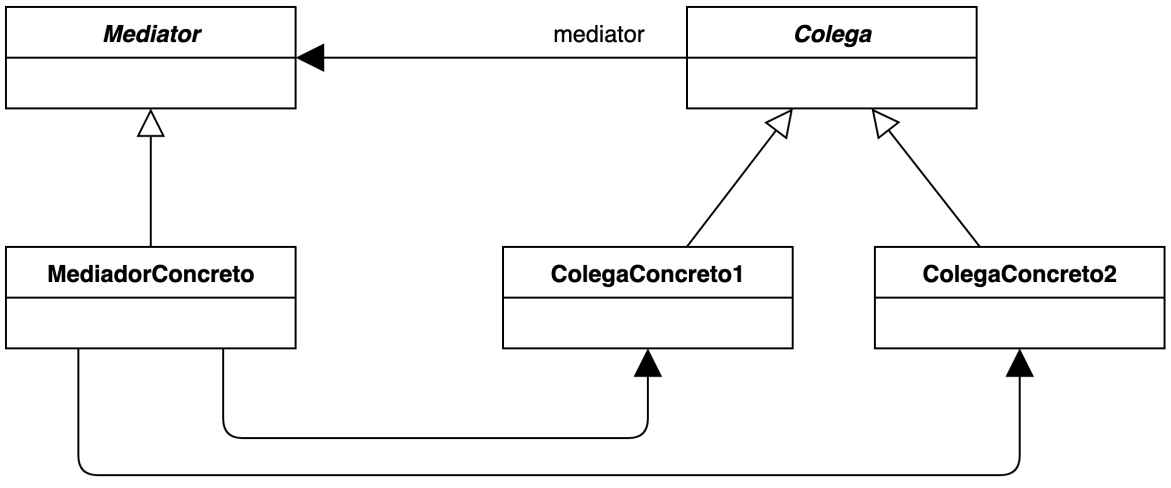


Diagrama de Classes

Consequências

- Um mediador centraliza o comportamento que seria distribuído entre vários colegas. Para alterar esse comportamento é preciso alterar apenas o mediador. As classes colegas podem ser reutilizadas como estão.

- Segue o princípio Aberto/Fechado. Promove um acoplamento flexível entre colegas. É possível introduzir, variar e reutilizar as classes Colega e *Mediador* de forma independente.

- Segue o princípio da responsabilidade única. O *Mediator* extrai as comunicações entre vários colegas. Isso simplifica os protocolos entre objetos, pois, o *Mediator* substitui uma comunicação do tipo muitos-para-muitos por uma comunicação um-para-muitos, entre ele e os objetos do tipo Colega. Os relacionamentos um-para-muitos são mais fáceis de entender e manter.

- Abstrai a forma como os objetos cooperam. Tornar a mediação um conceito independente e encapsula-la em um objeto permite o foco em como os objetos interagem independentemente de seu comportamento individual. Deste modo fica mais fácil entender como os objetos interagem em um sistema.

- O padrão Mediador troca complexidade de interação por complexidade no mediador. Como um mediador encapsula protocolos, ele pode se tornar mais complexo do que qualquer colega individual. Isso pode tornar o mediador um monolito difícil de manter.