Proxy

Padrões Estruturais

O padrão *proxy* fornece um substituto ou representante de outro objeto para gerenciar o acesso a ele.

Motivação (Por que utilizar?)

O padrão *Proxy* deve ser utilizado para criar um objeto representante que controla o acesso a outro objeto, por sua vez pode ser um objeto remoto, um recurso custoso ou algo que exija algum nível de segurança.

Existem vários tipos de *proxy*, suas aplicações variam conforme a finalidade exigida pelo contexto em que se encontra. Algumas das maneiras com que os proxies controlam o acesso aos objetos são:

- **Proxy remoto:** controla o acesso a um objeto remoto.

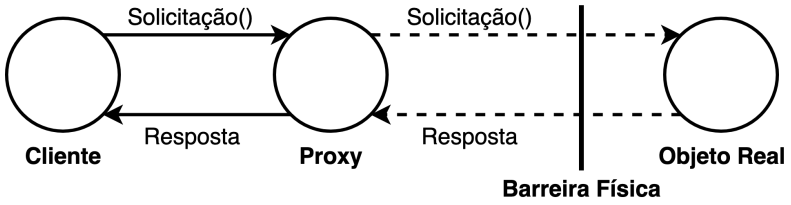


Diagrama do proxy remoto

Neste modelo de *proxy* o Cliente acredita que está se comunicando diretamente com o Objeto Real mas está se comunicando com um representante local (*proxy*) do objeto remoto, isso é possível devido ao fato de que *Proxy* e o Objeto Real implementam a mesma interface, ou seja, possuem o mesmo supertipo.

O *Proxy* finge ser o objeto remoto. Ele recebe a solicitação do Cliente a delega, via rede, ao Objeto Real.

O Objeto Real é quem realiza o trabalho, é nele onde estão implementados os métodos responsáveis por atender às solicitações do Cliente.

O *Proxy* remoto é muito utilizado em sistemas distribuídos, onde existe comunicação entre objetos que não estão no mesmo espaço de endereçamento, ou até mesmo distantes fisicamente (em computadores diferentes).

- **Proxy virtual**: controla o acesso a um objeto cuja criação é custosa.

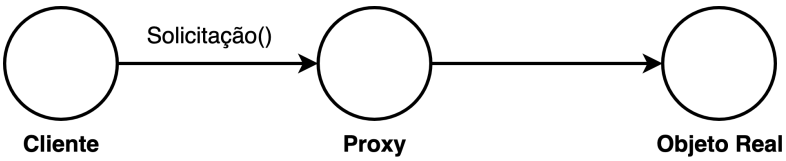


Diagrama do proxy virtual

Assim como no *proxy* remoto, neste modelo o Cliente também acredita estar se comunicando diretamente com o Objeto Real, porém o papel do *proxy* é outro.

O objeto *Proxy* geralmente retarda a criação do Objeto Real até que ele seja realmente necessário. Ele lida com as solicitações do Cliente, cria o Objeto Real quando necessário e delega as tais solicitações a ele. A classe *Proxy* cria e mantém referência a uma instância do Objeto Real.

Quando o Cliente envia uma solicitação ao *Proxy*, o objeto real pode ainda nem existir. Desta forma adotamos a política inicialização preguiçosa (*lazy initialization*), onde o Objeto Real será criado somente se for realmente necessário.

- **Proxy de proteção**: controla o acesso a um recurso com base em direitos de acesso.

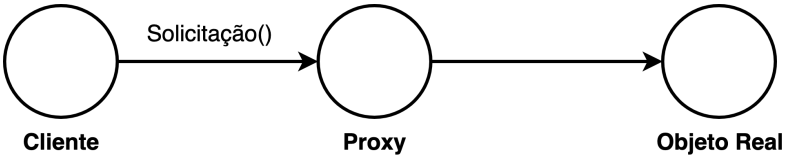
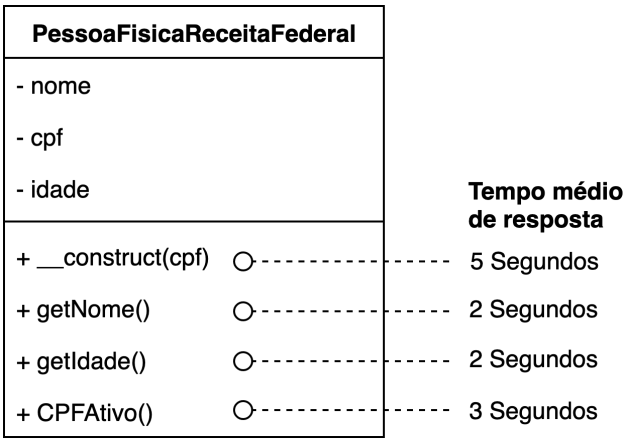


Diagrama do proxy de proteção

O *proxy* de proteção é muito semelhante ao *proxy* virtual, diferindo apenas em sua finalidade. Enquanto o proxy virtual foca na otimização do gerenciamento de recursos custoso, o *proxy* de proteção foca no controle de acesso ao Objeto Real. A classe *Proxy* é responsável por verificar se o cliente pode ou não acessar e/ou editar determinados recursos do objeto real.

Considere que estamos trabalhando em um projeto que possui a seguinte classe:



Classe de consulta de dados de pessoa física na receita federal

Suponhamos que a classe acima consome uma API da receita federal para consultar os dados de uma pessoa física a partir de seu CPF. Repare que o tempo de instanciação da classe é alto, em torno de 5 segundos, pois no construtor é feita a conexão e autenticação na API. Os métodos desta classe também possuem tempos de resposta consideravelmente altos para os padrões atuais.

A classe `PessoaFisicaReceitaFederal` é necessária no projeto para:

- Recuperar o nome real do dono do CPF;
- Recuperar a idade real do dono do CPF;
- Verificar se um CPF se encontra ativo na receita federal.

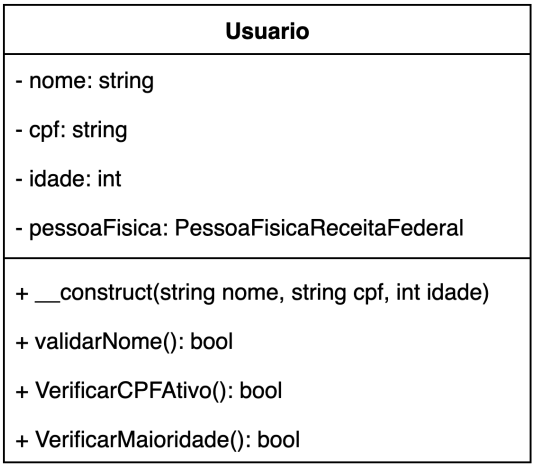
Para validar as informações esses dados serão comparados com os dados informados pelo usuário no momento do cadastro.

O problema é que o supervisor do projeto nos disse que os clientes estão reclamando devido a demora em todas as funcionalidades relacionadas a um usuário. Sabemos que a origem do problema é a classe `PessoaFisicaReceitaFederal`, porém, não é possível otimizar esta classe, já que a demora se dá pelo tempo de resposta da API.

A implementação abaixo é da classe **PessoaFisicaReceitaFederal.**

```php
class PessoaFisicaReceitaFederal
{
    private string $nome;
    private string $cpf;
    private int $idade;
    private bool $cpfAtivo;

    public function __construct(string $cpf)
    {
        $this->cpf = $cpf;
        $this->nome = 'João da Silva'; //Simulação de dado encontrado com base no CPF
        $this->idade = 25; //Simulação de dado encontrado com base no CPF
        $this->cpfAtivo = true; //Simulação de dado encontrado com base no CPF

        sleep(5); //Simulação do tempo de resposta da requisição à API.
        echo 'PessoaFisicaReceitaFederal criada com sucesso <br>';
    }

    public function getNome(): string
    {
        sleep(2); //Simulação do tempo de resposta da requisição à API.
        return $this->nome;
    }

    public function getIdade(): int
    {
        sleep(2); //Simulação do tempo de resposta da requisição à API.
        return $this->idade;
    }

    public function CPFAtivo(): bool
    {
        sleep(3); //Simulação do tempo de resposta da requisição à API.
        return $this->cpfAtivo;
    }
}
```

A classe **Usuario** utiliza a classe **PessoaFisicaReceitaFederal** e por este motivo também é afetada pelo seu longo tempo de resposta.



Classe de Usuario (Getters foram ocultados)

A classe **Usuario** se encontra implementada da seguinte forma:

```
class Usuario
{
    private string $nome;
    private string $cpf;
    private int $idade;
    private PessoaFisicaReceitaFederal $pessoaFisica;

    public function __construct(string $nome, string $cpf, int $idade)
    {
        $this->nome = $nome;
        $this->cpf = $cpf;
        $this->idade = $idade;

        //Aqui é inicializada uma instância da classe PessoaFisicaReceitaFederal.
        $this->pessoaFisica = new PessoaFisicaReceitaFederal($cpf);
    }

    //Compara o nome informado pelo usuário e o nome na receita federal (RF).
    public function validarNome(): bool
    {
        return $this->nome === $this->pessoaFisica->getNome();
    }

    //Verifica se o CPF se encontra ativo na RF.
    public function VerificaCPFAtivo(): bool
    {
        return $this->pessoaFisica->CPFAtivo();
    }

    //Verifica se a idade é maior ou igual a 18 anos e a compara com a idade na RF.
    public function verificarMaioridade(): bool
    {
        return $this->idade >= 18 && $this->idade === $this->pessoaFisica->getIdade();
    }

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

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

    //Retorna a idade do usuário.
    public function getIdade(): int
    {
        return $this->idade;
    }
}
```

Repare que toda vez que a classe **Usuario** é instanciada também é criada uma instância da classe **PessoaFisicaReceitaFederal** em seu construtor. A instância da classe da receita federal é utilizada pelos métodos **validarNome()**, **VerificaCPFAtivo()** e **verificarMaioridade()**, desse modo somente quando esses métodos são chamados que tal instância se faz necessária.

Considere que um usuário seja instanciado e apenas seu nome é recuperado.

```
$usuario = new Usuario('João da Silva', '11122233344', 25);
echo $usuario->getNome();
```

Resposta:

```
João da Silva
```

Em momento algum do código acima a classe **PessoaFisicaReceitaFederal** foi realmente necessária, porém ela ainda foi instanciada no construtor da classe **Usuario** adicionando 5 segundos no tempo de execução.

Se a criação de **PessoaFisicaReceitaFederal** fosse removida do construtor e colocada dentro de cada método que a utiliza a instanciação da classe **Usuario** seria mais rápida, porém todas as vezes que os métodos que utilizam a classe **PessoaFisicaReceitaFederal** fossem chamados seria necessário esperar os 5 segundos da criação da classe **PessoaFisicaReceitaFederal,** ou seja, resolveria um problema e criaria outro.

Outra abordagem seria a criação de um proxy virtual, lembrando que ele tem como objetivo controlar o acesso a um objeto cuja criação é custosa, e este é exatamente nosso problema.

No cenário em que nos encontramos o Cliente, ou seja, quem consome a classe **PessoaFisicaReceitaFederal** é a classe **Usuario,** portanto nosso Objeto Real é a classe **PessoaFisicaReceitaFederal.** Sabemos que o Cliente deve acreditar que o *proxy* é o próprio Objeto Real, então eles precisam ter o mesmo supertipo. Para satisfazer essa condição vamos criar uma interface com base nos métodos da classe **PessoaFisicaReceitaFederal.**

```
interface ReceitaFederalInterface
{
    public function __construct(string $cpf);

    public function getNome(): string;

    public function getIdade(): int;

    public function CPFAtivo(): bool;
}
```

Agora a classe **PessoaFisicaReceitaFederal** passa a implementar a interface **ReceitaFederealInterface.**

```
class PessoaFisicaReceitaFederal implements ReceitaFederalInterface
{
    //Nada muda no restante da classe.
}
```

Vamos agora à implementação proxy.

```php
class PessoaFisicaReceitaFederalProxy implements ReceitaFederalInterface
{
    private string $cpf;
    //A referência a classe PessoaFisicaReceitaFederal é inicializada como null.
    private ?ReceitaFederalInterface $pessoaFisicaRF = null;

    public function __construct(string $cpf)
    {
        //No construtor da Proxy apenas é guardado o valor do CPF.
        $this->cpf = $cpf;

        //Aqui não existe mais a instanciação da classe PessoaFisicaReceitaFederal.
    }

    /*Quando a classe PessoaFisicaReceitaFederal for realmente necessária
      Este método será o responsável pela sua criação.*/
    private function criarPessoaFisicaReceitaFederal(): void
    {
        /*Primeiro é testado se a classe já não foi instanciada.
          Como sua criação é custosa, e ela não irá mudar dentro deste contexto
          é melhor que ela seja instanciada apenas uma vez.*/
        if (is_null($this->pessoaFisicaRF)) {
            $this->pessoaFisicaRF = new PessoaFisicaReceitaFederal($this->cpf);
        }
    }

    /*Solicita a instanciação da classe PessoaFisicaReceitaFederal e delega a ela
      a responsabilidade de recuperar o nome do usuário.*/
    public function getNome(): string
    {
        $this->criarPessoaFisicaReceitaFederal();
        return $this->pessoaFisicaRF->getNome();
    }

    /*Solicita a instanciação da classe PessoaFisicaReceitaFederal e delega a ela
      a responsabilidade de recuperar a idade do usuário.*/
    public function getIdade(): int
    {
        $this->criarPessoaFisicaReceitaFederal();
        return $this->pessoaFisicaRF->getIdade();
    }

    /*Solicita a instanciação da classe PessoaFisicaReceitaFederal e delega a ela
      a responsabilidade de verificar se o CPF usuário está ativo.*/
    public function CPFAtivo(): bool
    {
        $this->criarPessoaFisicaReceitaFederal();
        return $this->pessoaFisicaRF->CPFAtivo();
    }
}
```

O próximo passo é fazer a Classe **Usuario** utilizar a **PessoaFisicaReceitaFederalProxy** ao invés da classe **PessoaFisicaReceitaFederal**.

```
class Usuario
{
    private string $nome;
    private string $cpf;
    private int $idade;
    //Agora ao invés da classe concreta PessoaFisicaReceitaFederal a interface é esperada.
    private ReceitaFederalInterface $pessoaFisica;

    public function __construct(string $nome, string $cpf, int $idade)
    {
        $this->nome = $nome;
        $this->cpf = $cpf;
        $this->idade = $idade;
        /*A classe PessoaFisicaReceitaFederalProxy é instanciada no lugar da classe
          PessoaFisicaReceitaFederal.
          A classe PessoaFisicaReceitaFederalProxy não instancia a classe
          PessoaFisicaReceitaFederal em seu construtor, portanto, a instanciação da classe
          PessoaFisicaReceitaFederal é evitada até que ela seja realmente necessária.*/
        $this->pessoaFisica = new PessoaFisicaReceitaFederalProxy($cpf);
    }

    //Nada muda no restante da implementação.
}
```

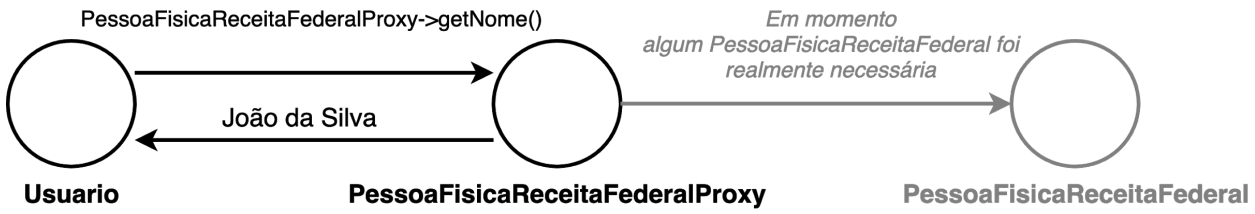
Feitas as modificações acima na classe **Usuario** as linhas abaixo teria sua execução feita instantaneamente.

```
$usuario = new Usuario('João da Silva', '11122233344', 25);
echo $usuario->getNome();
```

Resposta:

```
João da Silva
```

Isso se dá devido ao fato da classe **PessoaFisicaReceitaFederal** não ser mais instanciada no construtor da classe **Usuario**, portanto, serão economizados os 5 segundos que a classe **PessoaFisicaReceitaFederal** leva para ser instanciada. Isso é possível graças a *proxy* que está intermediando essa comunicação.



Fluxo de informação intermediado pela classe PessoaFisicaReceitaFederalProxy

Os métodos **validarNome()**, **VerificaCPFAtivo()** e **verificarMaioridade()** da classe Usuario, precisam de uma instância da classe **PessoaFisicaReceitaFederal**. Somente quando eles são chamados essa classe será instanciada.

```php
$usuario = new Usuario('João da Silva', '11122233344', 25);
if ($usuario->verificarMaioridade()) {
    echo "O usuário é maior de idade";
} else {
    echo "O usuário não é maior de idade";
}
```

Resposta:

```
O usuário é maior de idade
```

No código acima o seguinte fluxo é iniciado.

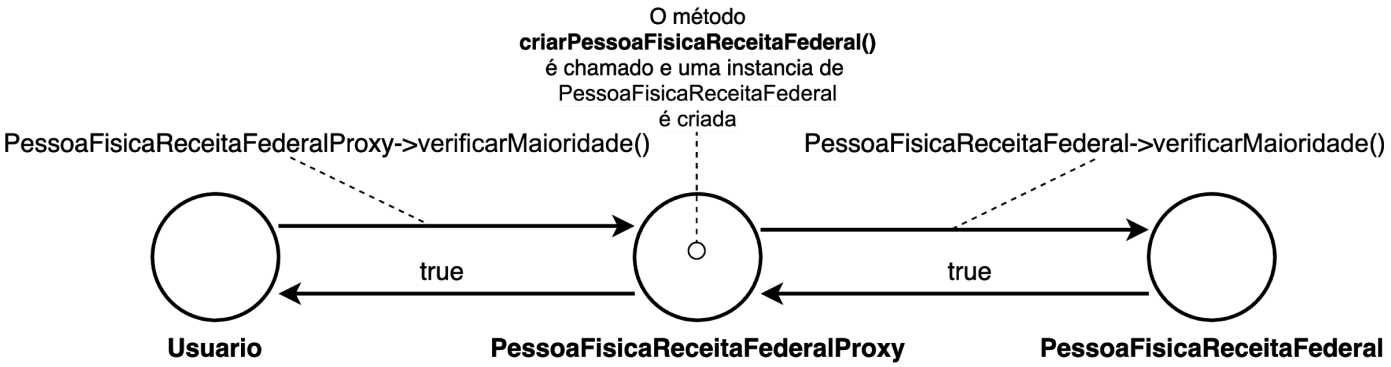
1. O método **verificarMaioridade()** da classe **Usuario** chama o método **getIdade()** da classe **PessoaFisicaReceitaFederalProxy**.

2. O método **getIdade()** da classe **PessoaFisicaReceitaFederalProxy** chama o método **criarPessoaFisicaReceitaFederal()** desta mesma classe.

3. O método **criarPessoaFisicaReceitaFederal()** cria uma instância da classe **PessoaFisicaReceitaFederal** e mantém uma referência a ela.

4. O **getIdade()** da classe **PessoaFisicaReceitaFederalProxy** agora chama o método **getIdade()** da classe **PessoaFisicaReceitaFederal** que foi instanciada no passo 3.

5. A reposta é retornada de **PessoaFisicaReceitaFederal** para **PessoaFisicaReceitaFederalProxy** depois para **Usuario** até chegar no **if** do código acima.



Fluxo de informação intermediado pela classe PessoaFisicaReceitaFederalProxy

Depois dos passos de 1 até 5 serem executados, caso os métodos `validarNome()` ou `VerificaCPFAtivo()` fossem chamados, a criação da instância de `PessoaFisicaReceitaFederal` (Passo 3) não aconteceria mais, pois já teria sido criada anteriormente. Isso evita que o processamento custoso seja feito de forma desnecessária.

Os métodos da classe `PessoaFisicaReceitaFederal` também possuem tempos de resposta consideravelmente altos. É possível fazer cache da primeira resposta destes métodos na classe `PessoaFisicaReceitaFederalProxy`. Deste modo, a classe `PessoaFisicaReceitaFederalProxy`, caso seja necessário, irá delegar o processamento de um método a classe `PessoaFisicaReceitaFederal` apenas na primeira vez que ele for chamado, e armazenará a resposta. Da segunda vez em diante a classe `PessoaFisicaReceitaFederalProxy` já saberá o que responder sem ter que recorrer a classe `PessoaFisicaReceitaFederal`.

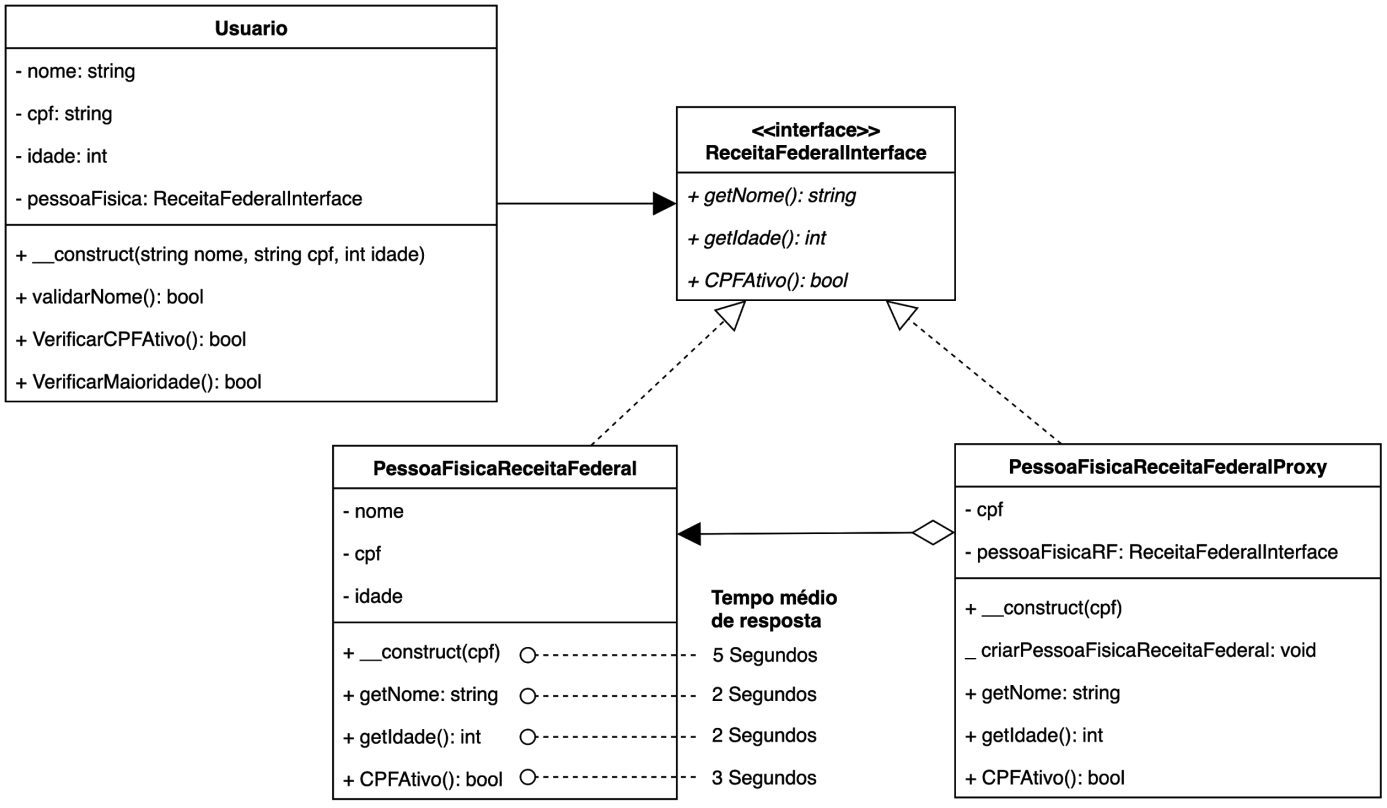


Diagrama de classes completo do exemplo

Aplicabilidade (Quando utilizar?)

O padrão *Proxy* é aplicável sempre que for necessária uma referência a um objeto que seja mais versátil e sofisticada do que um simples ponteiro. Essas são algumas situações comuns onde o padrão *Proxy* é aplicável:

1. Quando se precisa de um representante local para um objeto em um espaço de endereço diferente um **proxy remoto** pode ser aplicado.

2. Quando se deseja criar objetos custosos apenas sob demanda pode se utilizar um **proxy virtual.**

3. Quando objetos devem ter direitos de acesso diferentes, um **proxy de proteção** pode ser utilizado para controlar o acesso ao objeto original.

4. Uma **referência inteligente *(smart reference)*** é uma substituição de um ponteiro simples que executa ações adicionais quando um objeto é acessado. Os usos típicos incluem:

    - Contagem do número de referências ao objeto real para que ele possa ser liberado automaticamente da memória quando não houver mais referências a ele.

    - Carregamento persistente de um objeto na memória quando é mencionado pela primeira vez.

    - Verificação de bloqueio no Objeto Real antes de ser acessado para garantir que nenhum outro objeto possa alterá-lo de forma indevida.

Componentes

- **Cliente:** Objeto que consome a *Proxy*.

- **Objeto:** Define a interface que é implementada tanto pelo ObjetoReal quanto pelo Proxy, permitindo que o Cliente trate o Proxy como sendo o ObjetoReal.

- **ObjetoReal:** É o objeto que executa as solicitações feitas pelo Cliente. Nele estão as implementações capazes de responder tais solicitações.

- **Proxy:** Mantém uma referência ao ObjetoReal para poder encaminhar solicitações a ele sempre que necessário. Prove para o Cliente a mesma interface do ObjetoReal para que possa ser utilizada pelo Cliente de forma transparente. Além de controlar o acesso ao ObjetoReal também pode ser responsável por criá-lo e excluí-lo.

Algumas outras responsabilidades variam de acordo com o tipo de *proxy*.

- Os **proxies remotos** são responsáveis por codificar uma solicitação e seus argumentos e por enviar tal solicitação codificada para o ObjetoReal em um espaço de endereço diferente.

- Já os **proxies virtuais** podem fazer cache de informações adicionais sobre o ObjetoReal, para seja possível adiar o acesso direto a ele.

- Por fim, os **proxies de proteção** verificam se o Cliente possui as permissões de acesso necessárias para executar uma solicitação.

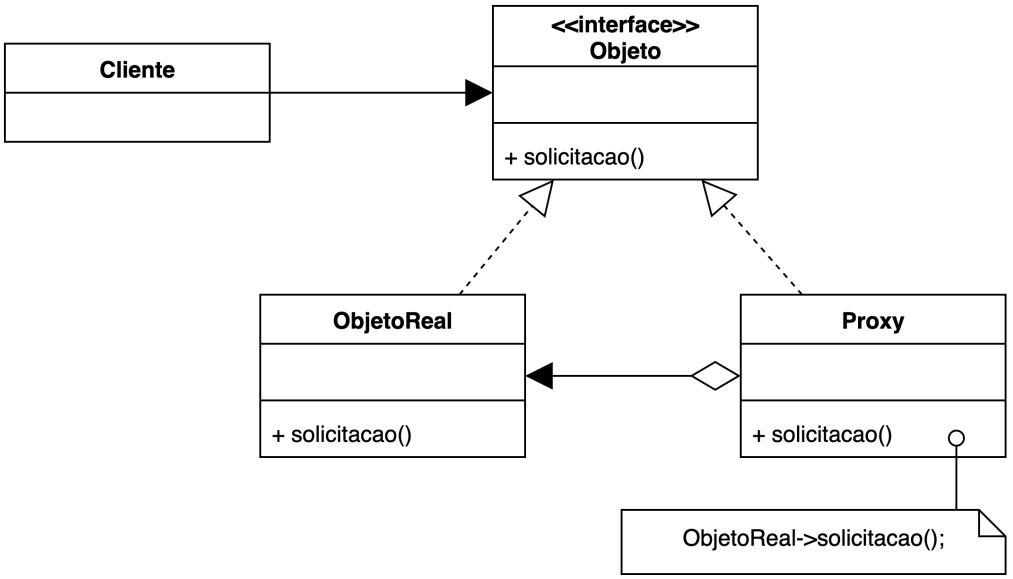


Diagrama de Classes

Consequências

- Um proxy remoto pode ocultar o fato de um objeto estar alocado em um espaço de endereçamento diferente.

- Um proxy virtual pode executar otimizações, como criar um objeto sob demanda.

- Os proxies de proteção e as referências inteligentes permitem tarefas adicionais de manutenção quando um objeto é acessado.

- Princípio Aberto/Fechado é respeitado. É possível introduzir novos proxies sem alterar o serviço ou os clientes.

- O código pode se tornar mais complicado, pois diversas novas classes podem ser introduzidas no código.

- A latência (tempo de resposta) dos serviços pode aumentar.