# Padrões de Projeto da Semana 3

Clovis Fernandes

# Padrões de Projeto do GoF

| Criação | Estrutural | Comportamental |
|---|---|---|
| Factory Method | Bridge Pattern | Strategy Pattern |
| Singleton Pattern | Composite Pattern | Template Method Pattern |
| Builder Pattern | Proxy Pattern | State pattern |
| Abstract Factory | Decorator Pattern | Observer Pattern |
| Prototype Pattern | Adapter Pattern | Chain of Responsibility Pattern |
| | Flyweight Pattern | Command Pattern |
| | Façade Pattern | Iterator Pattern |
| | | Mediator Pattern |
| | | Visitor Pattern |
| | | Interpreter Pattern |
| | | Memento Pattern |

# Padrão de Projeto Composite

O padrão de projeto Composite é um padrão do tipo Padrão Estrutural, de acordo com o GoF.

Objetivo

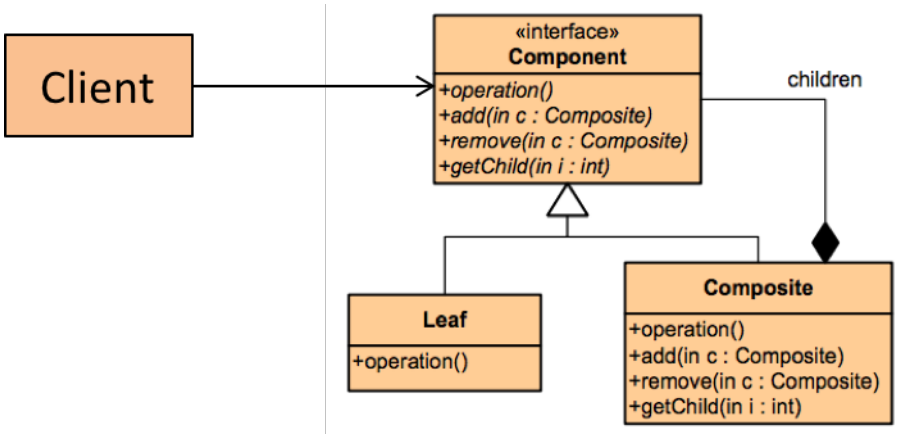
Compor (recursivamente) objetos em estruturas de árvore para representar hierarquias do tipo parte-todo (composição de classes). Permite aos clientes tratar objetos individuais e composições de objetos de maneira uniforme.

O padrão Composite é resultado da aplicação da composição recursiva, que permite associar a classe composta (classe Composite na figura abaixo) a uma abstração comum (interface Component na mesma figura abaixo).

Como exemplo, temos um Diretório de Arquivos, que pode conter Arquivos ou Diretórios; se for Diretório que ele contenha, esse Diretório pode conter Arquivos ou Diretórios e assim por diante. Claramente, essa é uma estrutura em árvore gerada pela composição recursiva!

Além disso, pode-se ter mais de uma subclasse Leaf: subclasses Leaf01, Leaf02 etc.; quantas subclasses que servem para parar a composição recursiva forem necessárias para um dado contexto!

Diagrama de Classes Representativo




# Padrão de Projeto Chain of Responsibility

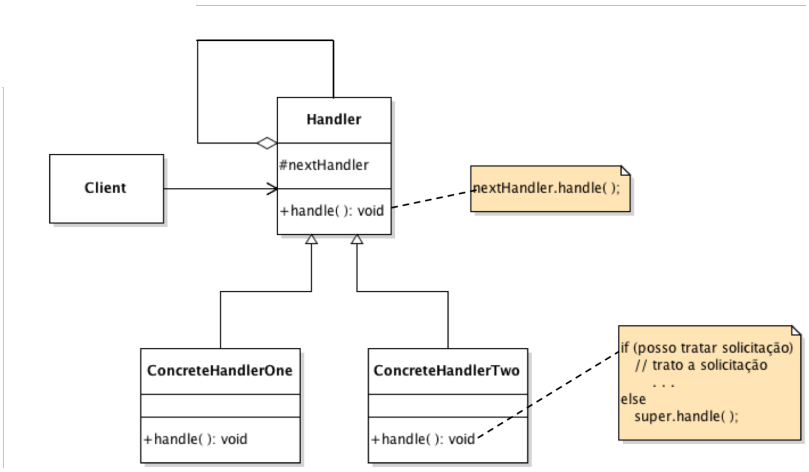
O padrão de projeto Chain of Responsibility é um padrão do tipo Padrão Comportamental, de acordo com o GoF.

Objetivo

Evita acoplar o remetente de uma solicitação ao seu receptor ao permitir que mais de um objeto tenha a chance de tratar a solicitação também.

Os objetos receptores tornam-se partes de uma cadeia ou "pipeline"de objetos receptores e a solicitação é enviada de um objeto receptor a outro ao longo da cadeia, até que um (ou mais, talvez todos) dos objetos receptores manipule a solicitação.

Diagrama de Classes Representativo




# Padrão de Projeto Proxy

O padrão de projeto Proxy é um padrão do tipo Padrão Estrutural, de acordo com o GoF.

Objetivo

Proxy é um procurador em português, alguém que substitui ou faz o trabalho de outra pessoa, a pedido dessa pessoa.

Dessa forma, o objetivo do padrão Proxy é fornecer um objeto substituto ou espaço reservado para outro objeto, que vai agir como se fosse o outro objeto em algumas situações preestabelecidas, de forma a controlar o acesso de objetos clientes.

Usa-se para isso um nível extra de indireção para, entre outras possibilidades, apoiar o seguinte:

- Acesso virtual controlado – Prover espaço reservado para controlar quando um objeto complexo ou que ocupa muitos recursos do sistema operacional, por exemplo, uma imagem muito grande, precisa ser instanciado e inicializado → Proxy Virtual

- Acesso protegido – Prover diferentes direitos de acesso a um objeto com dados sensíveis, de modo que nem sempre um cliente terá acesso ao mesmo → Proxy Protetor

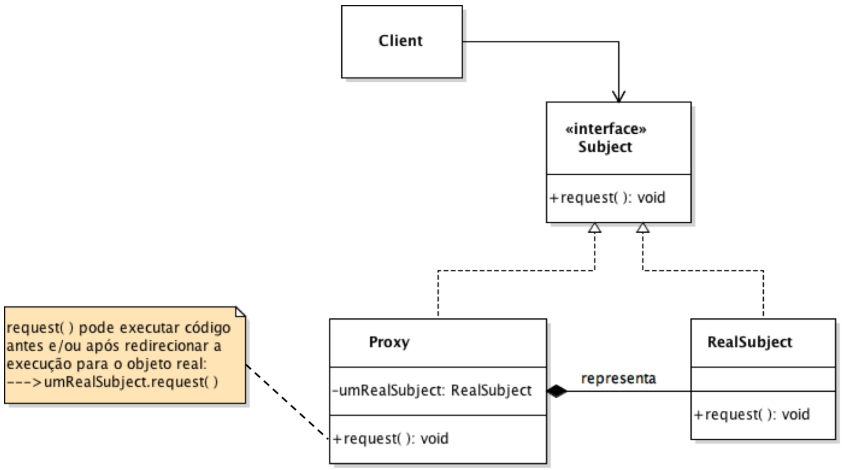
- Acesso distribuído – Fornecer meios para acessar e referenciar objetos que executam em outros processos ou em outras máquinas → Proxy Remoto

O importante é que, no diagrama abaixo, um objeto Client nunca vai enviar mensagens para um objeto RealSubject; um objeto Client só vai acessar o objeto

Proxy associado! Por exemplo, no caso de Proxy Remoto, o objeto Proxy iria encapsular todo código necessário para permitir o acesso remoto do objeto RealSubject; de maneira geral, não seria necessário para o objeto Client saber se o acesso do objeto RealSubject é por meio de sockets, RMI ou CORBA, por exemplo!

Finalmente, um Proxy fornece geralmente a mesma interface do RealSubject, com as mesmas responsabilidades.

Diagrama de Classes Representativo




# Padrão de Projeto Decorator

O padrão de projeto Decorator é um padrão do tipo Padrão Estrutural, de acordo com o GoF.

Objetivo

Anexar responsabilidades adicionais a um objeto de forma dinâmica (em tempo de execução). Fornecer uma alternativa flexível à subclasse para ampliar a sua funcionalidade, uma vez que com a herança pura isso não é viável, porque a herança é estática e se aplica a uma classe.

Uma responsabilidade adicional pode ser fruto da adição de novo estado à subclasse do Decorator, como se pode ver no diagrama representativo do Padrão Decorator abaixo.

Graças à composição recursiva, pode-se dizer que é um empacotamento recursivo que permite adicionar, de forma incremental, novas responsabilidades ao componente a ser decorado, conforme especificado pelo cliente. Ou seja, é o Cliente que tem a responsabilidade de compor as configurações desejadas, envolvendo o objeto central recursivamente.

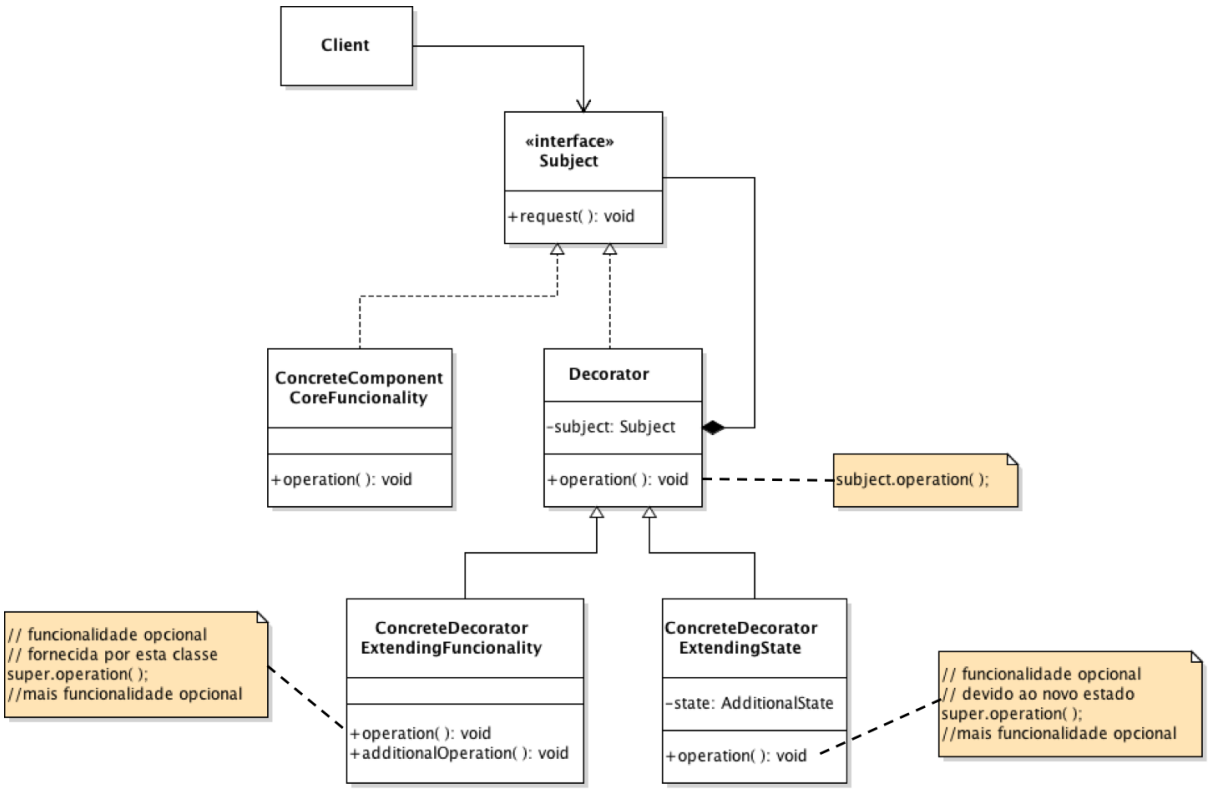
De forma simbólica, imagine que um cliente embrulha um presente, coloca-o em uma caixa e embrulha a caixa. Suponha que eu queira, como as bonequinhas russas, colocar essa caixa embrulhada em uma nova caixa e embrulhar essa nova caixa. Tudo isso é possível.

Pense que embrulhar e colocar em caixa sejam peles novas do meu objeto presente. O Padrão Decorator permite que você altere a pele de um objeto.

Isto está em oposição ao Padrão Strategy, que por sua vez permite que você altere as entranhas de um objeto, variando suas responsabilidades, no caso entendidas como algoritmos diferentes, enquanto a pele do objeto não se altera.

Enquanto um objeto Proxy fornece geralmente a mesma interface do RealSubject, com as mesmas responsabilidades, um objeto Decorator fornece uma interface enriquecida, com novas responsabilidades adicionadas em tempo de execução.

Diagrama de Classes Representativo




# Padrão de Projeto Adapter

O padrão de projeto Adapter é um padrão do tipo Padrão Estrutural, de acordo com o GoF.

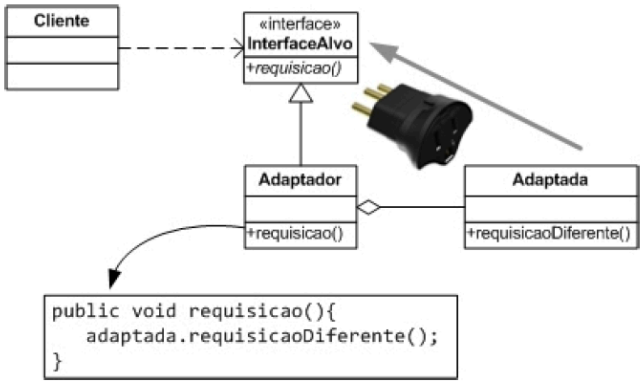
Objetivo

Converter a interface de uma classe em outra interface que os clientes esperam. Permitir que as classes funcionem juntas, o que, caso contrário, não poderia ocorrer devido à incompatibilidade das interfaces envolvidas.

Geralmente corresponde ao reúso de um componente legado em um novo sistema, resolvendo o problema das diferentes interfaces, o que significaria empacotar/encapsular essa classe (por meio de uma wrapper class) com uma nova interface compatível com o novo ambiente de aplicação.

O Padrão Adapter fornece uma interface diferente para a classe sendo adaptada; o Padrão Proxy fornece a mesma interface da classe que ele substitui; e Padrão Decorator fornece uma interface enriquecida para a sua classe alvo.

Diagrama de Classes Representativo




# Padrão de Projeto Singleton

O padrão de projeto Singleton é um padrão do tipo Padrão de Criação, de acordo com o GoF.

Objetivo

Certificar-se de que uma classe tenha uma e apenas uma instância e fornecer um ponto de acesso global a essa instância.

Às vezes, é importante ter apenas uma instância para uma classe. Por exemplo, em um sistema, deve haver apenas um gerenciador de janelas ou apenas um sistema de arquivos ou apenas um spooler de impressão. Normalmente, o Padrão Singleton é usado para gerenciar recursos internos ou externos de forma centralizada e fornecem um ponto de acesso global para sua instância única.

O Padrão Singleton é um dos padrões de design mais simples:

- Envolve apenas uma classe, que é responsável por instanciar-se, para garantir que ela não crie mais do que uma instância
- Ao mesmo tempo, fornece um ponto de acesso global a essa instância.
- Neste caso, a mesma instância pode ser usada em todos os lugares, sendo impossível invocar diretamente o construtor do dado objeto Singleton

Pode-se garantir uma única instância com a maioria das aplicações corriqueiras de Singleton. Contudo, a literatura recomenda usar algoritmos mais complicados e sofisticados nas seguintes 3 situações, para se garantir instância única do objeto Singleton:

1. Em aplicativos multi-threaded
2. Em casos em que a classe Singleton implementa a interface java.io.Serializable
3. Em casos em que as classes carregadas por diferentes carregadores de classe (ClassLoaders) acessarem um objeto Singleton

Contudo, uma implementação simples de Singleton usando ***enum*** do Java é suficiente para garantir sempre instância única, mesmo nas 3 situações acima!

Diagrama de Classes Representativo