Design Patterns

> Design Patterns são soluções já consolidadas para problemas comuns em software design, especificamente design de código.

Design Patterns

- São esqueletos de ideias que já foram utilizadas diversas vezes em outros projetos.
- Confundidos com algoritmos prontos que adicionamos a nosso código.
- Ferramentas incríveis que todo desenvolvedor deve revisitar de tempos em tempos na sua carreiras.

Diagrama UML

> UML é uma estrutura padrão para desenvolvimento de software, bastante utilizada na diagramação de estruturas do código, bancos de dados e fluxo de código.

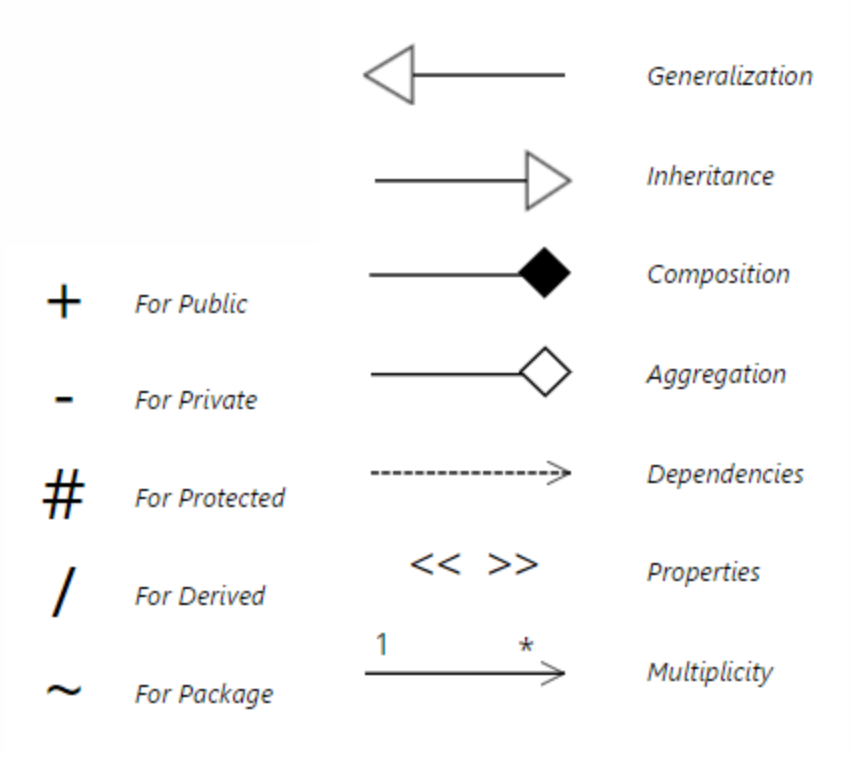
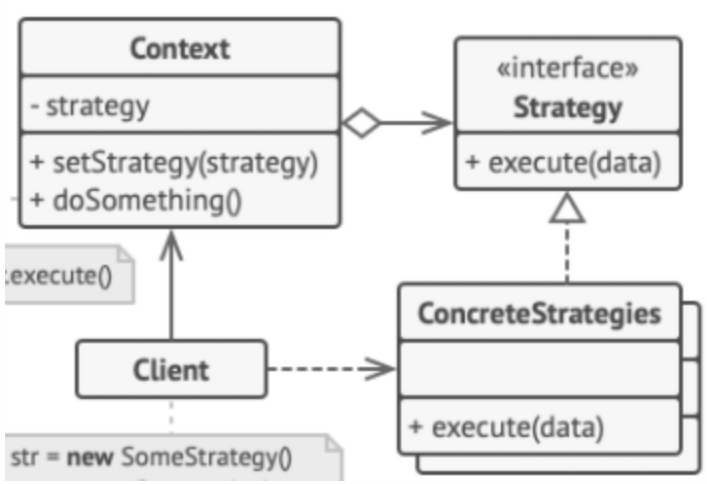


Diagrama UML

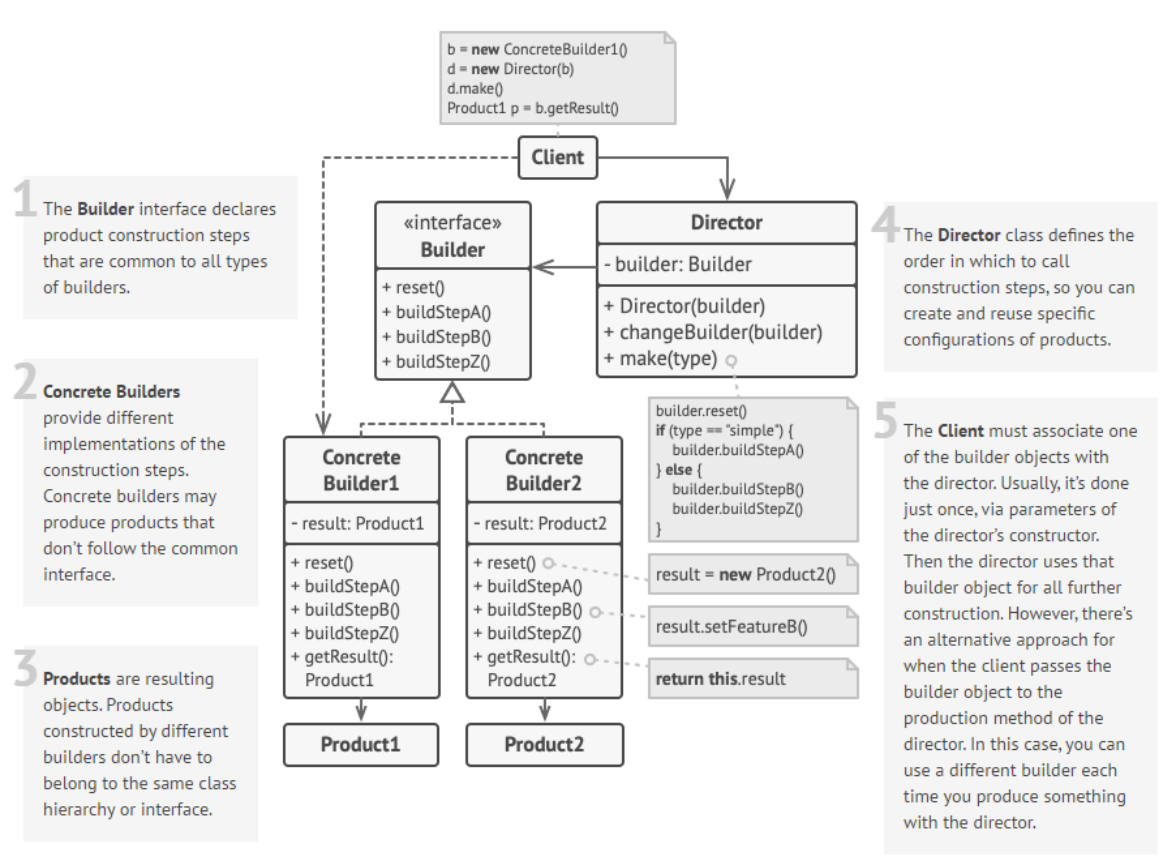


Classificação dos Design Patterns

- Creational Patterns
  - Provê mecanismos de criação de objetos que aumentam a flexibilidade e o reuso de código existem.
- Structural Patterns
  - Provê mecanismos para montar classes e objetos em grandes estruturas, enquanto ainda mantem flexibilidade e eficiência.
- Behavioral Patterns
  - Define uma comunicação efetiva e atribui responsabilidades entre os objetos e classes.

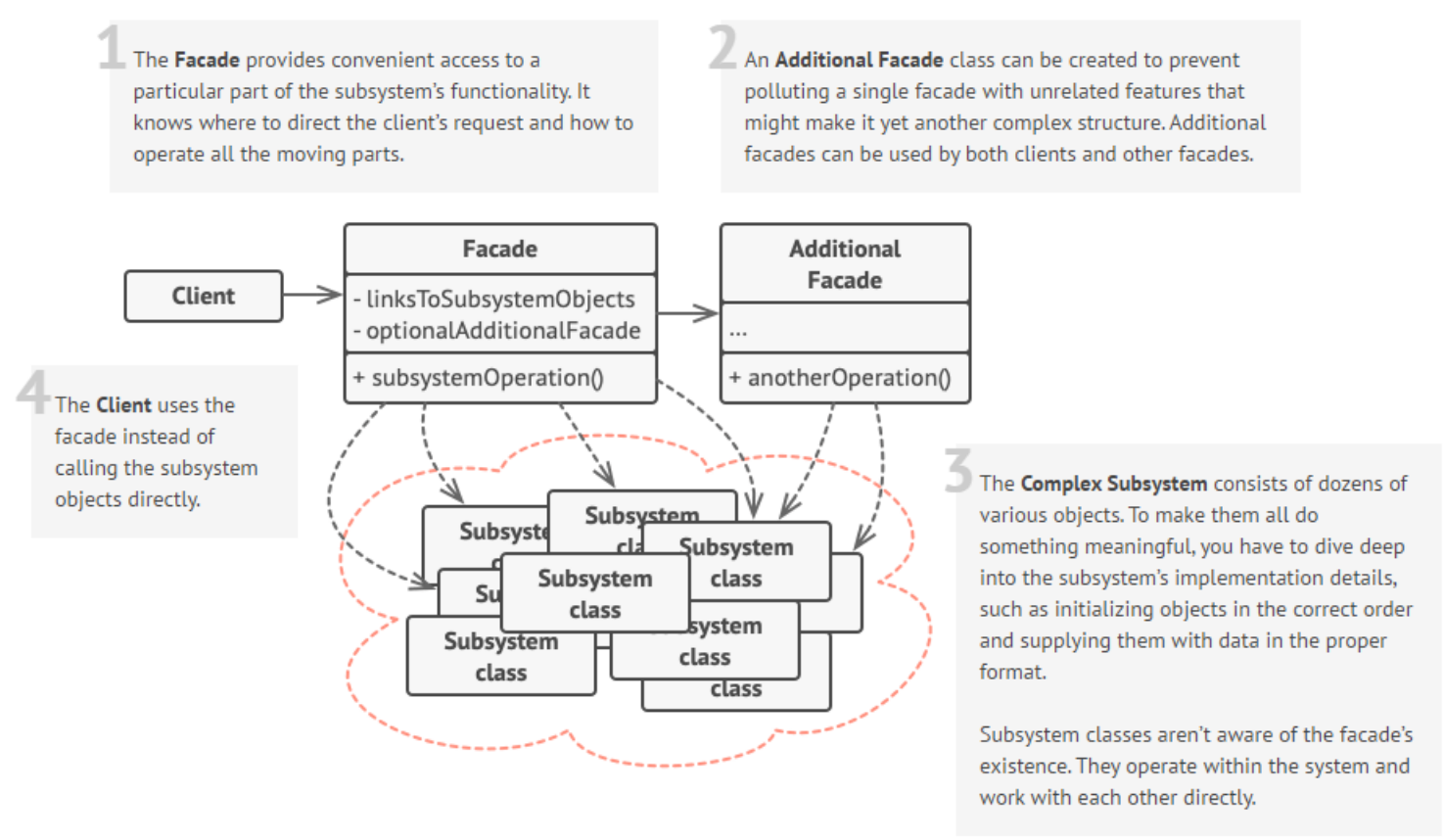
Builder

> Builder Pattern é destinado a resolver a construção de objetos complexos



Facade Pattern

Facade Pattern provê uma interface simplificada para uma biblioteca, framework ou qualquer outro tipo de conjunto de classes complexo.



Chain of Responsability Pattern

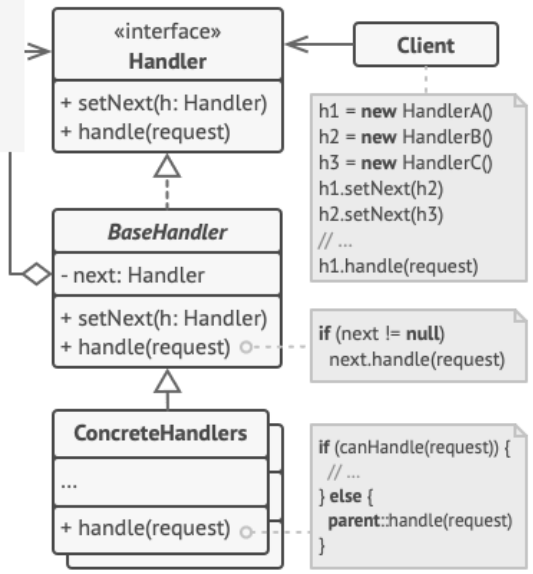
Chain of Responsability Pattern é uma especificaçaõ estrutural que garante a passagem de requisições em uma cadeia de validações.

Estrutura

1 O **Handler** declara a interface, comum a todos os handlers concretos. Ele geralmente contém apenas um único método para lidar com pedidos, mas algumas vezes ele pode conter outro método para configurar o próximo handler da corrente.

2 O **Handler Base** é uma classe opcional onde você pode colocar o código padrão que é comum a todas as classes handler.

Geralmente, essa classe define um campo para armazenar uma referência para o próximo handler. Os clientes podem construir uma corrente passando um handler para o construtor ou setter do handler anterior. A classe pode também implementar o comportamento padrão do handler: pode passar a execução para o próximo handler após checar por sua existência.



4 O **Cliente** pode compor correntes apenas uma vez ou compô-las dinamicamente, dependendo da lógica da aplicação. Note que um pedido pode ser enviado para qualquer handler na corrente —não precisa ser ao primeiro.

3 **Handlers Concretos** contém o código real para processar pedidos. Ao receber um pedido, cada handler deve decidir se processa ele e, adicionalmente, se passa ele adiante na corrente.

Os handlers são geralmente auto contidos e imutáveis, aceitando todos os dados necessários apenas uma vez através do construtor.

Quando não aplicar Design Patterns

- Cópia de código de outras linguagens
- Resulta em um código desnecessariamente mais complexo
- Exemplos
  - Observer Pattern em C#
  - Prototype em Javascript
  - Strategy em linguagens com recurso de Lambda Functions

🏠 Dever de casa - Design Patterns

Cada mentorado deverá implementar um Design Pattern em um projeto separado, tentando replicar um problema real.

O projeto deve ser desenvolvido em uma linguagem de programação acordada com o Mentor.

A apresentação irá consistir em:

- Explicação do Design Pattern escolhido
- Explicação da implementação
- Diagrama de classes
- Os testes para garantir a funcionalidade da solução

Design Patterns recomendados

- Factory Method
- Builder
- Adapter
- Composite
- Decorator
- Facade
- Chain of Responsibility

Os mentorados devem escolher um dos Design Patterns disponíveis. Apenas um Design Pattern deve ser apresentado por um mentorado.

Avaliação

Quesito	Estado
Relacionado a Apresentação	
Apresentação da Design Pattern	
Apresentação do diagrama de classes	
Apresentação de pseudo código	
Relacionado a Implementação	
Design Pattern implementado	
Uso do Design Pattern em um problema	