IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

Programação Avançada

Padrão de Desenho Strategy

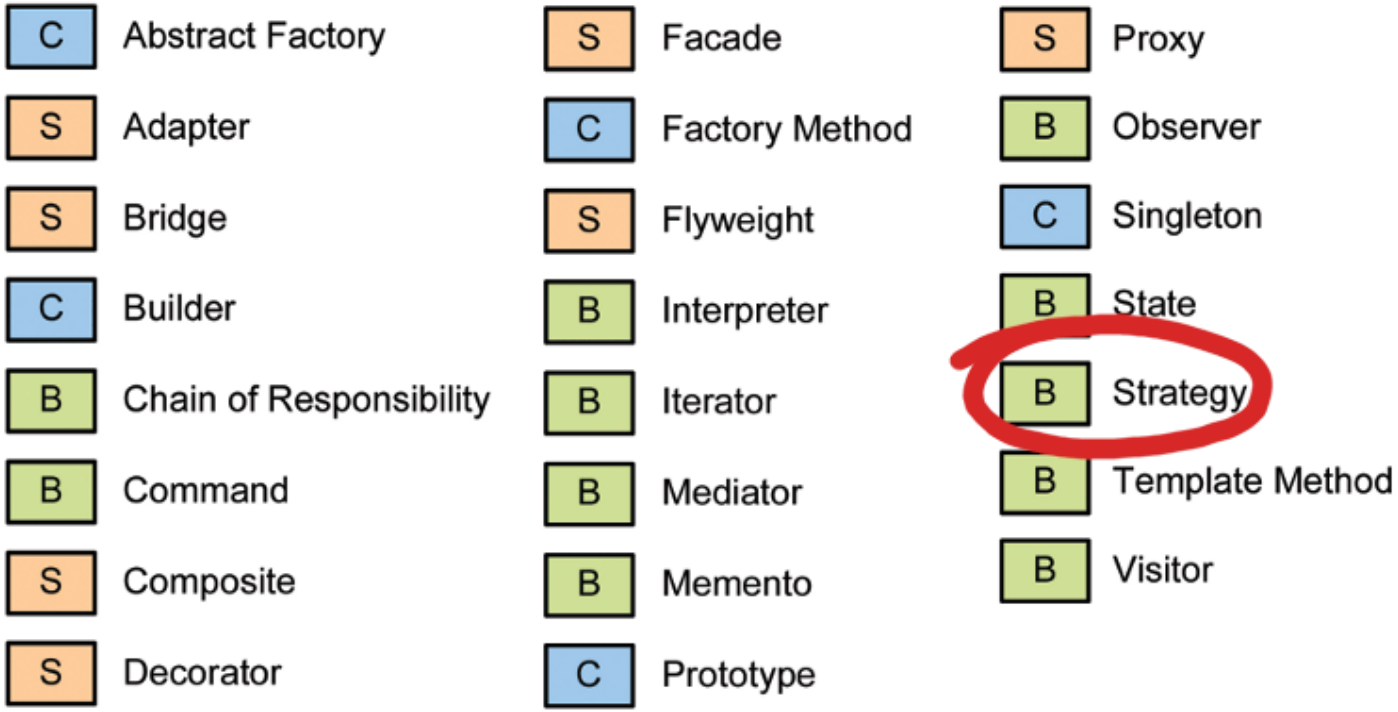
Bruno Silva, Patrícia Macedo

Sumário 📝

- Padrão de desenho **Strategy**
  - Enquadramento
  - Motivação
  - Solução Proposta (pelo padrão)
  - Exemplo de Aplicação
    - Exercícios
  - Prós e contras

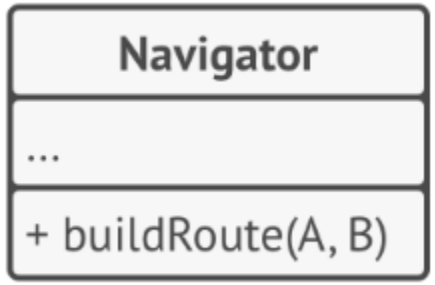
Enquadramento

Um dos originalmente propostos pelo *GoF* e classificado como **de comportamento** (behavioral)



Problema 😬

Temos uma classe `Navigator` que será responsável por calcular rotas entre dois locais **A** e **B**.



A rota pode ser calculada através de diferentes critérios:

- De carro;
- Transportes públicos;
- A pé;
- ...

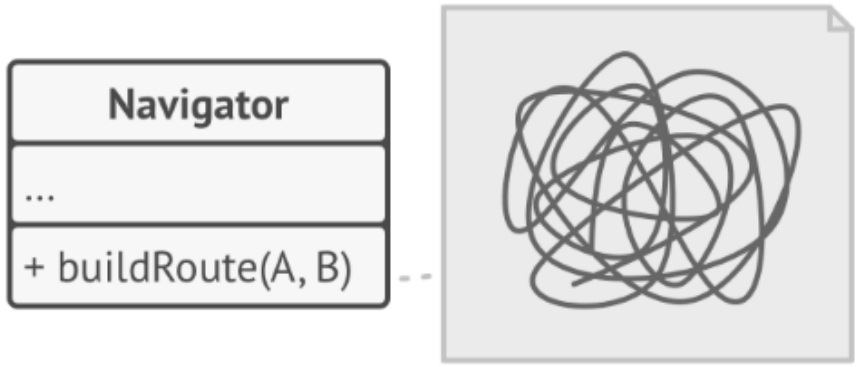
Problema 🤔

À partida, isto pode ser resolvido de três formas:

1. A classe `Navigator` possui um atributo `criteria` mutável (e.g., `enum Criteria {CAR, PUBLIC_TRANSPORT, WAL}`) e o método `buildRoute(A,B)` tem um *comportamento* distinto consoante o valor deste atributo.
    - O *cliente* altera o critério desejado e invoca o método.
2. Diferentes métodos para os diferentes critérios, e.g., `buildRouteByCar(A,B)`, `buildRouteByPublicTransportation(A,B)`, etc.
    - O *cliente* invoca o método respetivo ao critério desejado.
3. Através de herança/polimorfismo: e.g., sub-classes `NavigatorByCar`, `NavigatorByPublicTransportation`, etc.
    - O *cliente* **instancia** a classe apropriada e invoca o método.

Problema 🤔

👎 As "soluções" 1 e 2 implicam que sempre que seja adicionado um novo critério, e.g., de bicicleta, que se altere a própria classe `Navigation`. Ao longo do tempo esta classe irá ficar difícil de manter.



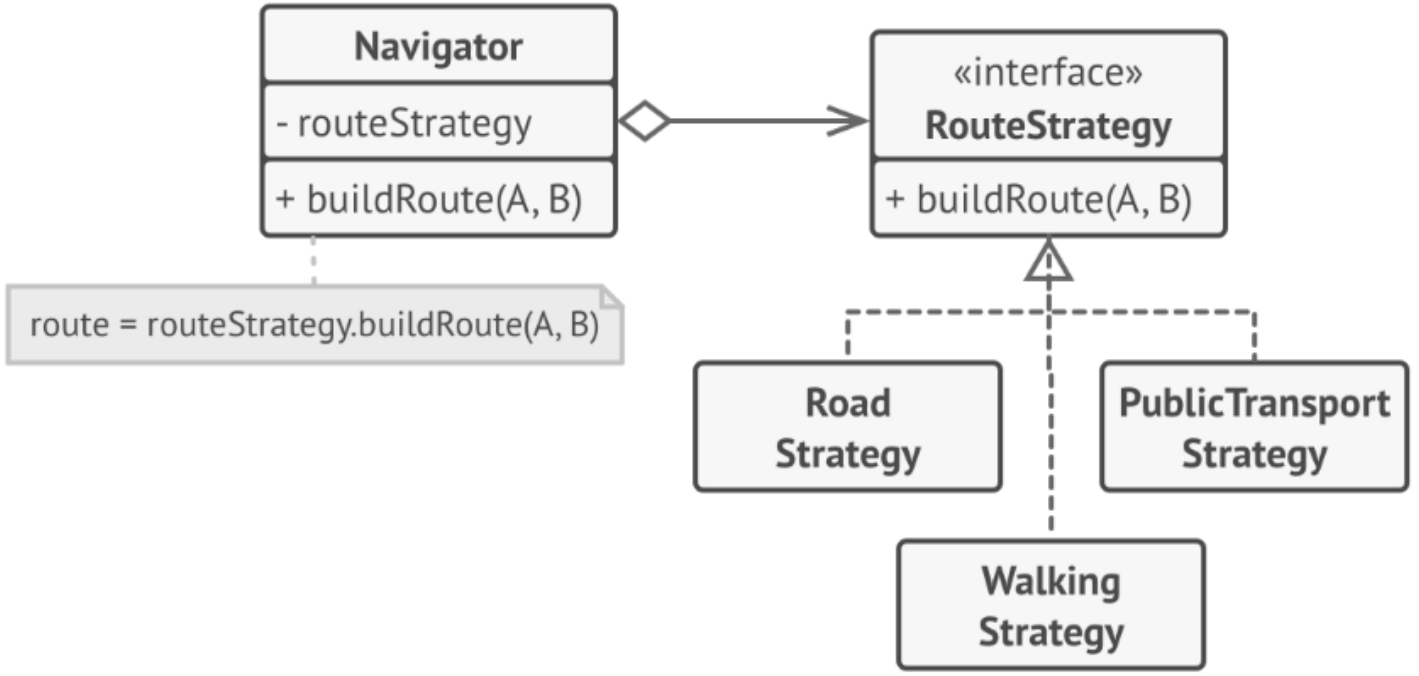
👎 A "solução" 3 implica criar uma hierarquia de classes, onde as subclasses apenas redefinem um método (*polimorfismo*).

Solução Proposta 😌

O padrão **Strategy** sugere pegar numa classe que faz uma **tarefa** específica de muitas formas diferentes e extrair estes "algoritmos" para classes separadas chamadas de *estratégias*.

- A classe original (**context**) guarda uma referência para uma das estratégias; irá *delegar* o trabalho da tarefa para este objeto.
    - O *context* não é responsável por escolher a estratégia apropriada à tarefa. O **cliente** passar-lhe-á a estratégia desejada.
- O *context* não conhece os detalhes das estratégias; trabalha com todas através de uma *interface* genérica comum (**strategy**) que expõe um método para desencadear o algoritmo encapsulado na estratégia selecionada (**concrete strategy**).

Solução Proposta 😌



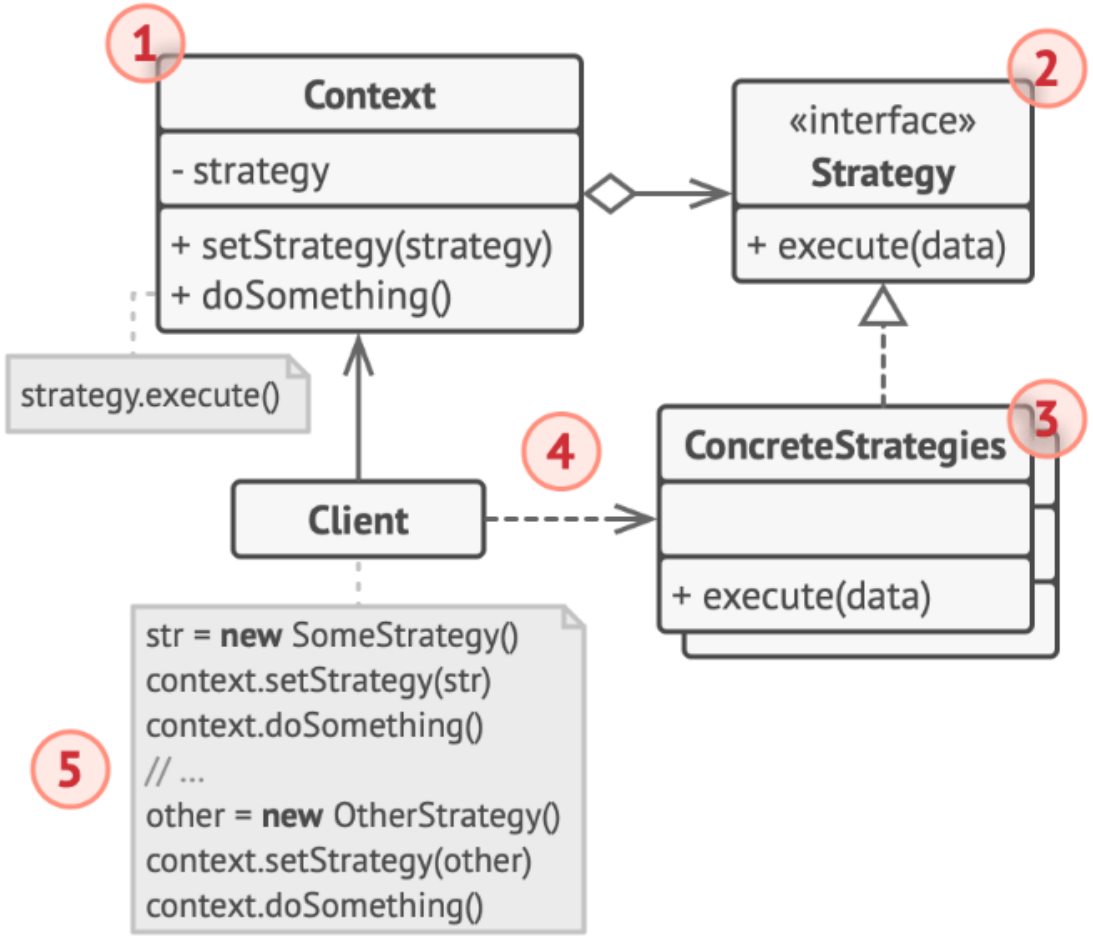
👍 o *context* torna-se independente das estratégias concretas, logo podemos adicionar novas estratégias sem modificar o código do *context*.

👍 podemos alterar o "comportamento" de `Navigator` em *tempo-de-execução*.

Padrão Strategy

Participantes do padrão:

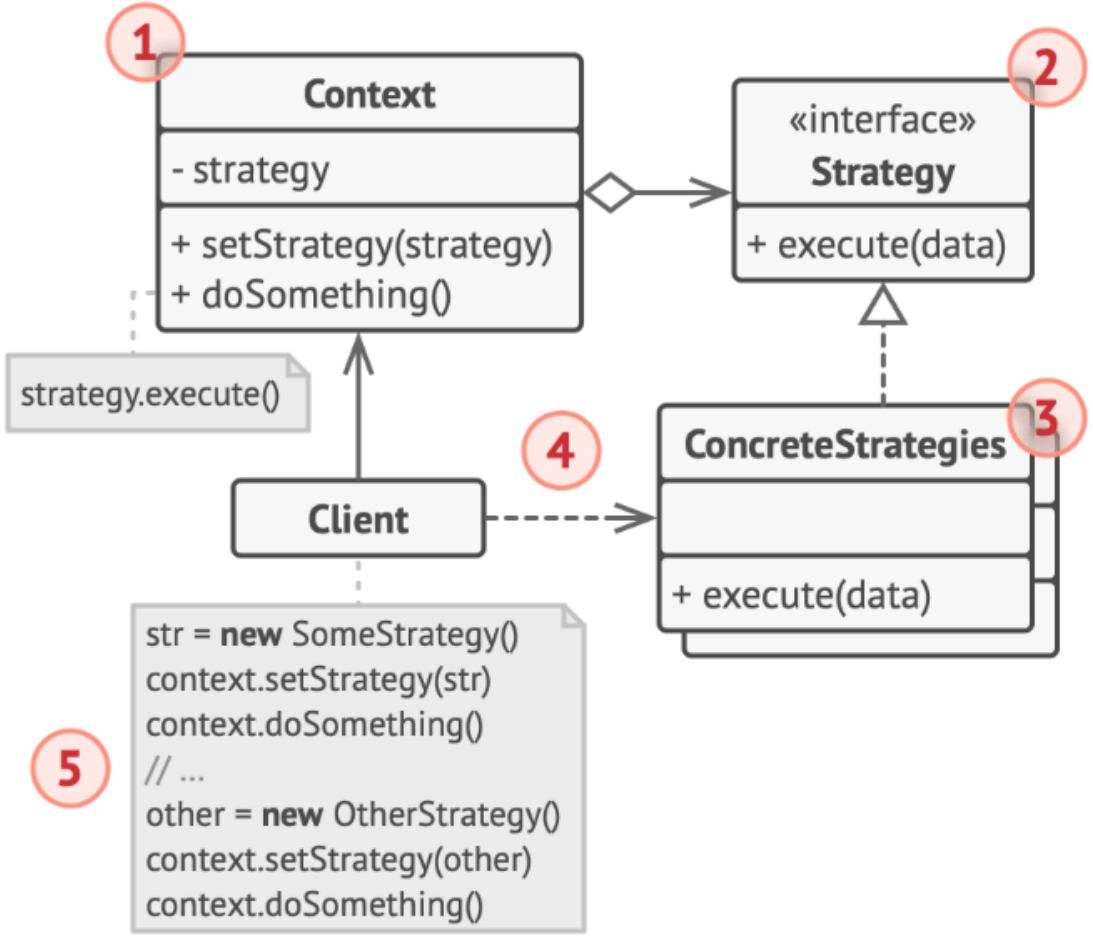
1. **Context**: mantém uma referência para estratégia (para a qual irá delegar a tarefa) e comunica através da interface.
2. **Strategy**: *interface* comum de todas as estratégias; contém o(s) método(s) utilizados para executar a estratégia.
3. **Concrete Strategies**: implementam as diferentes estratégias (algoritmos)
4. **Client**: Cria a estratégia desejada que é passada ao *context* (`setStrategy`); invoca a operação no *context*.



Padrão Strategy

⚠️ **Nota**

- Dado que as *concrete strategies* são classes separadas, o método `execute` tem de receber por parâmetros toda a informação necessária para desempenhar a tarefa.
- A interface tem de prever isto.



Exercício | Exemplo de aplicação

Ficha de Aluno 👨‍🎓

Repositório de apoio à aula:

- https://github.com/estsetubal-pa-geral/JavaPatterns_Strategy

---

Considere este programa que modela a ficha de um aluno através da classe `StudentRecord`, i.e., o seu percurso académico - unidades curriculares efetuadas e notas obtidas.

É possível obter a média aritmética das notas obtidas.

Exercício | Exemplo de aplicação

Ficha de Aluno 👨‍🎓

1. Pretende-se poder calcular a média do aluno de diferentes formas:
    - Média aritmética;
    - Média ponderada relativamente aos ECTS de cada UC.

   Aplique o padrão **strategy** para resolver este novo requisito.
2. Pretende-se que o método `toString()` também tenha um comportamento diferente consoante a estratégia de cálculo da nota
    - Quando é utilizada a média ponderada, deve constar na listagem os ECTS das UCs.

Padrão Strategy
Prós e contras

✓ Podemos alterar a estratégia em tempo-de-execução.

✓ Podemos substituir herança por composição.

✓ *Single Responsibility Principle*. Cada estratégia concreta é responsável por um único algoritmo.

✓ *Open/Closed Principle*. É fácil adicionar novas estratégias sem alterar o *context*.

✗ Se existir um número muito baixo e fixo de estratégias, pode não compensar a criação de novas interfaces e estratégias decorrentes do padrão.

✗ O *cliente* tem de conhecer as estratégias existentes para poder selecionar uma.

Webgrafia 📖

- https://refactoring.guru/design-patterns/strategy

  As imagens dos slides 4, 6, 8, 9 e 10 são creditadas a este *website*.

- What is the Strategy Pattern? (Software Design Patterns)

  https://www.youtube.com/watch?v=9uDFHTWCKkQ