# DES-SIS-II - A7 Texto de apoio

Site: EAD Mackenzie
Tema: DESENVOLVIMENTO DE SISTEMAS II {TURMA 03B} 2023/1
Livro: DES-SIS-II - A7 Texto de apoio

Impresso por: FELIPE BALDIM GUERRA .
Data: terça, 2 mai 2023, 21:24

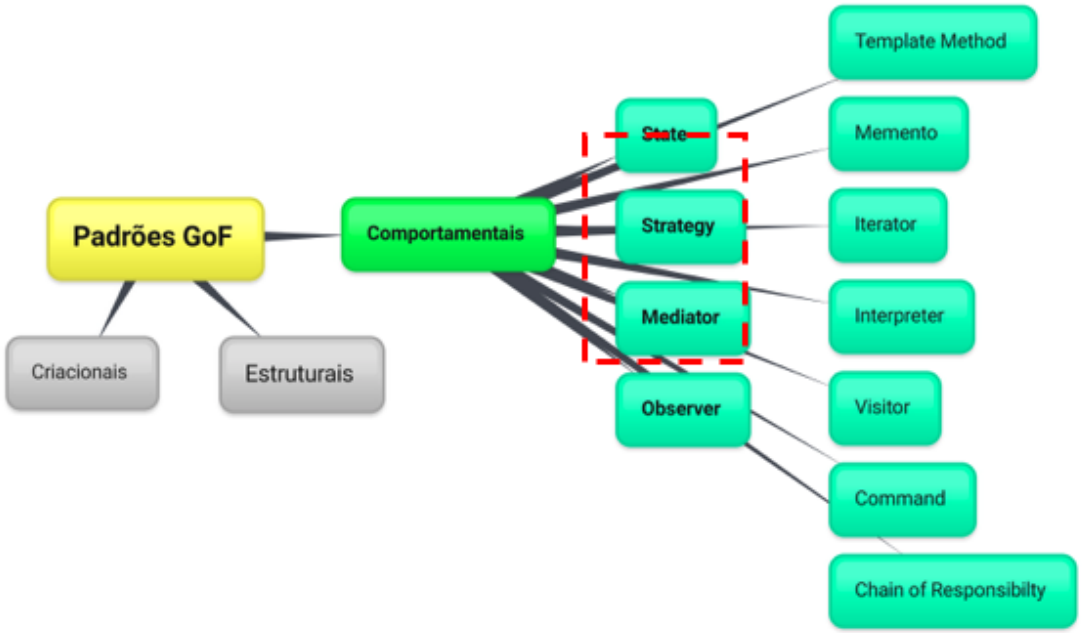
# Descrição

# Índice

# 1. PADRÕES GoF COMPORTAMENTAIS

Na aula anterior, aprendemos a respeito dos padrões GoF estruturais. Vimos que são padrões que respondem questões relacionadas à organização das classes em um projeto orientado por objetos, e então estudamos mais a fundo alguns desses padrões.

Nesta aula, trabalharemos outra família de padrões GoF, os comportamentais. Trata-se do maior conjunto de padrões de projeto (11, no total) que visam trazer soluções a problemas relacionados ao comportamento geral de objetos e classes em um projeto. São eles:




Fonte: Elaborada pelo autor.

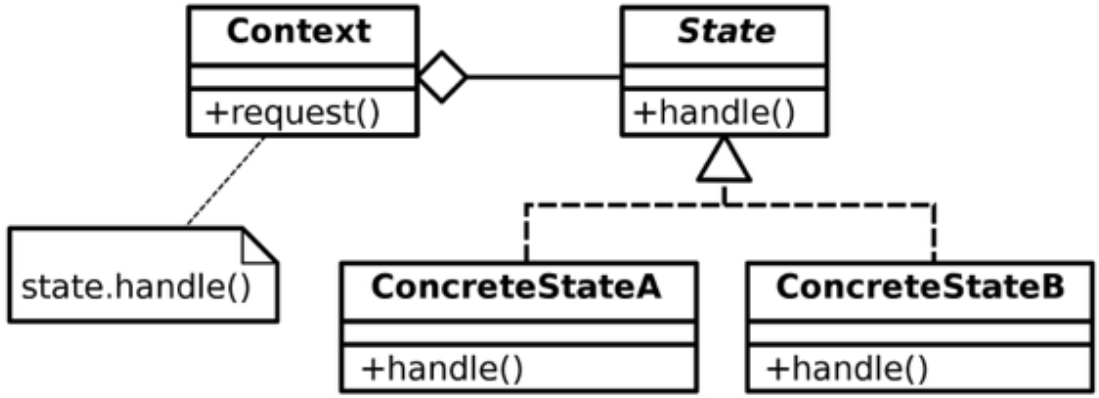
Nesta aula, focaremos em dois desses padrões comportamentais: State e Strategy. Vamos a eles!

# 2. STATE

Problema: Como fazer com que um objeto possa alterar seu comportamento de acordo com mudanças em seu estado interno?

Solução: Encapsule cada um de seus possíveis estados em classes.

Diagrama de classes:




Fonte:  Wikipédia.

Código em Java:

```
class Context {
    private State state;

    public Context() {
        state = new ConcreteStateA();
    }

    void setState(State newState) {
        state = newState;
    }

    public void request() {
        state.handle();
    }
}

interface State {
    void handle ();
}

class ConcreteStateA implements State {
    public void handle() {
            }
}

class ConcreteStateB implements State {
    public void handle() {
            }
}
```

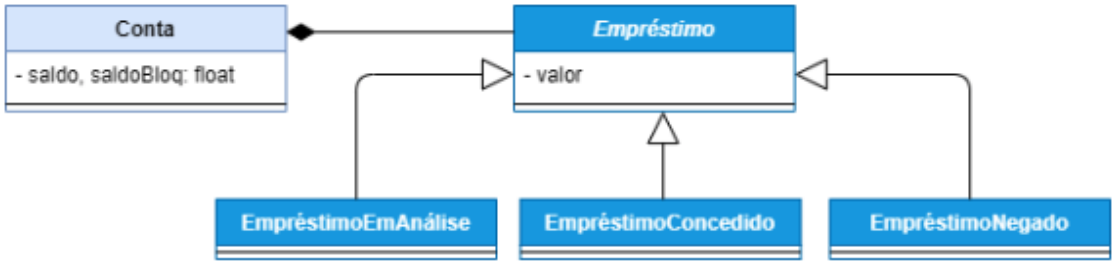
Pontos a considerar:

- O padrão State traz uma maneira flexível de fazer com que o comportamento de um objeto seja alterado em função de seu estado interno.
- Estruturalmente falando, o padrão State segue o padrão Bridge.
- Em situações nas quais as mudanças de estado de um objeto levam ao uso de condicionais, estas podem ser facilmente substituídas pelo uso adequado do polimorfismo.

Exemplo do mundo real:

Em um banco, um pedido de empréstimo de um cliente pode ter três estados possíveis: em análise, concedido ou recusado. Isso gera impactos no saldo da conta corrente. Enquanto o pedido de empréstimo está em análise, o valor do empréstimo pode aparecer como saldo na conta, porém bloqueado para movimentações. No caso de empréstimo concedido, o saldo da conta é automaticamente atualizado para o valor com o empréstimo, desbloqueando o valor do saldo – diferente do que ocorre quando o empréstimo é negado, zerando-se o valor bloqueado.

A aplicação do padrão State, neste caso, pode ser vista a seguir:




Veja mais em <https://refactoring.guru/design-patterns/state>.

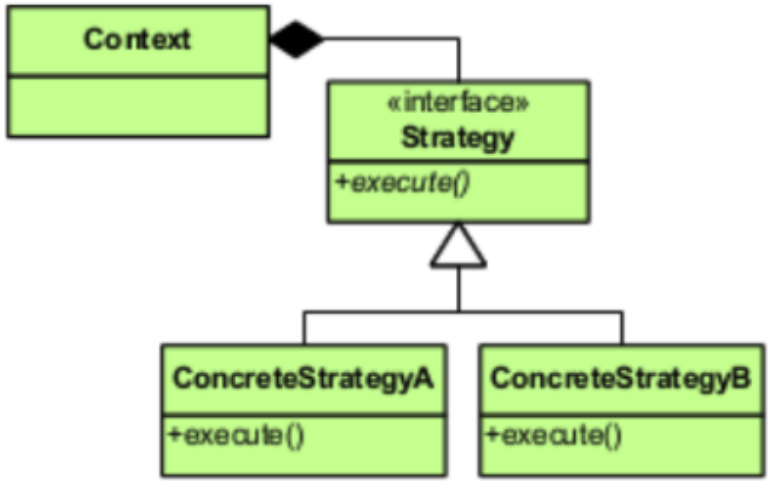
Aplique o que aprendeu agora na primeira atividade Praticando desta aula.

# 3. STRATEGY

**Problema**: Como variar entre comportamentos de uma classe (descritos por algoritmos)?

**Solução**: Encapsular cada um dos comportamentos (algoritmos) em uma classe, permitindo seu intercâmbio.

**Diagrama de classes:**




Fonte: Wikipédia.

Código em Java:

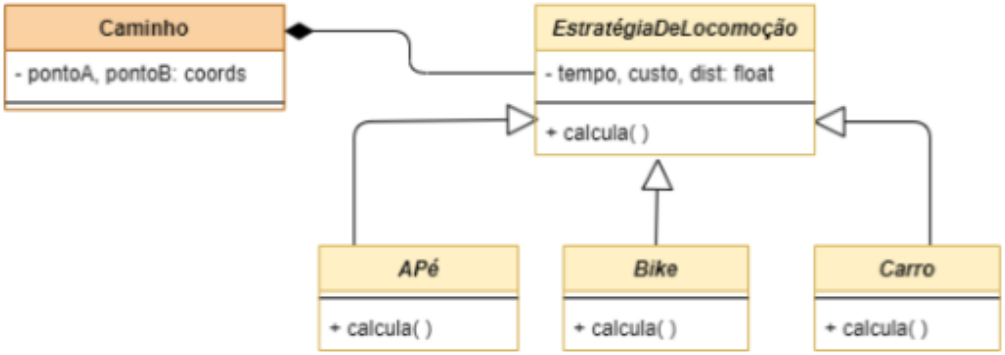
```
class Context {
    private Strategy str;

    public Context() {
        str = new ConcreteStrategyA();
    }

    void setStrategy(Strategy newStr) {
        str = newStr;
    }

    public void request() {
        str.execute();
    }
}

interface Strategy {
    void execute ();
}

class ConcreteStrategyA implements Stratategy {
    public void execute() {
            }
}

class ConcreteStrategyB implements Strategy {
    public void execute() {
            }
}
```

Pontos a considerar:

- A estrutura de State e Strategy é similar, o que muda é o que está sendo encapsulado em cada uma das classes (os diferentes estados de um objeto ou as distintas estratégias de solução de problemas).
- De forma similar ao State, o uso de Strategy pode evitar condicionais múltiplas, o que já se sabe ser um “mau cheiro” de código, de acordo com a terminologia de refatoração.

Exemplo do mundo real:

Considere um aplicativo de mapas em tempo real, que, para perfazer um dado caminho, apresenta diferentes estratégias de mobilidade: a pé, de bicicleta, transporte público, automóvel etc. Aplicando-se o padrão Strategy, cada uma dessas alternativas pode ser encapsulada em uma classe e escolhe-se a estratégia (algoritmo de locomoção) mais adequada, considerando-se fatores como tempo, distância e custo do trajeto.

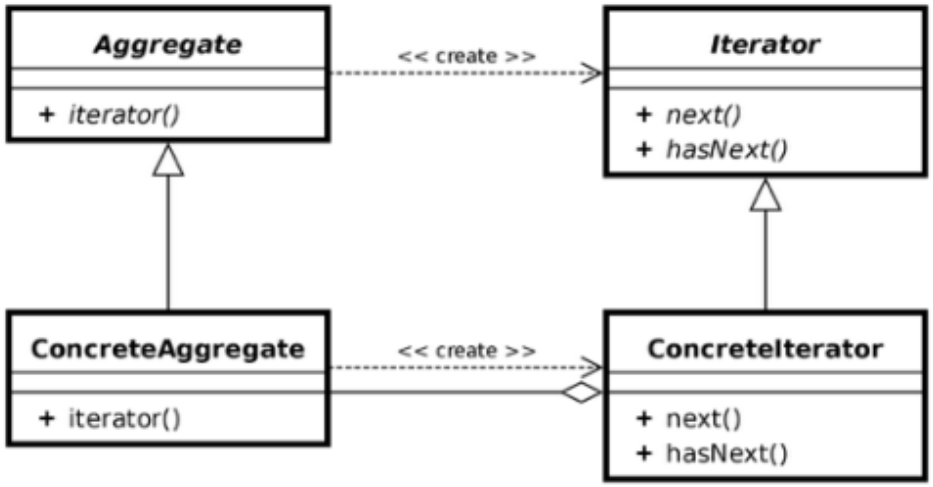



Veja mais em <https://refactoring.guru/design-patterns/strategy>.

Aplique o que aprendeu agora na segunda atividade Praticando desta aula.

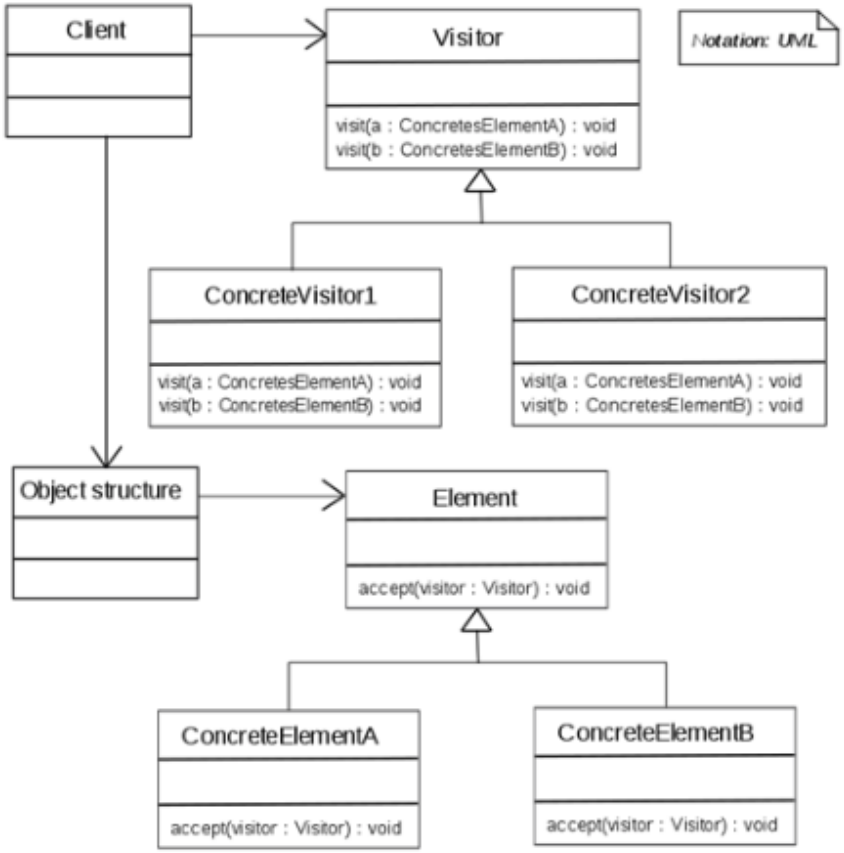
# 4. DEMAIS PADRÕES COMPORTAMENTAIS

A família de padrões comportamentais é bastante extensa e seu estudo requer atenção em relação a cada situação em que tais padrões são aplicáveis. Há padrões bastante específicos, como o Interpreter, que é aplicado quando é necessário realizar algum tipo de tradução e conversão (bastante útil na escrita de compiladores e interpretadores, por exemplo – situação em que o padrão Visitor também é empregável), assim como há padrões de uso mais geral, como o Iterator, que é utilizado para se implementar formas de percorrer uma determinada coleção (como um array ou uma lista, por exemplo. Seguem exemplos de seus diagramas de classe:



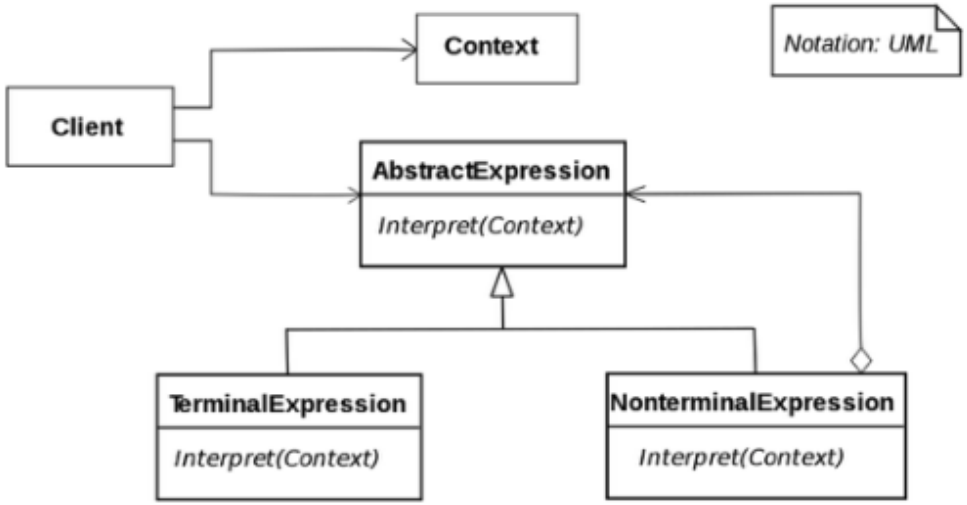

Fonte: Wikipédia.

**Padrão Iterator**




Fonte: Wikipédia.

**Padrão Interpreter**




Fonte: Wikipédia.