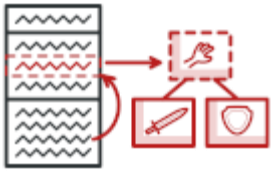


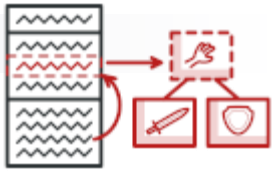
Modelos, Métodos e Técnicas de Engenharia de Software
Visão e análise de projeto Padrões Prática 3 – Strategy (21)

Prof. Osmar de Oliveira Braz Junior

Prof. Richard Henrique de Souza

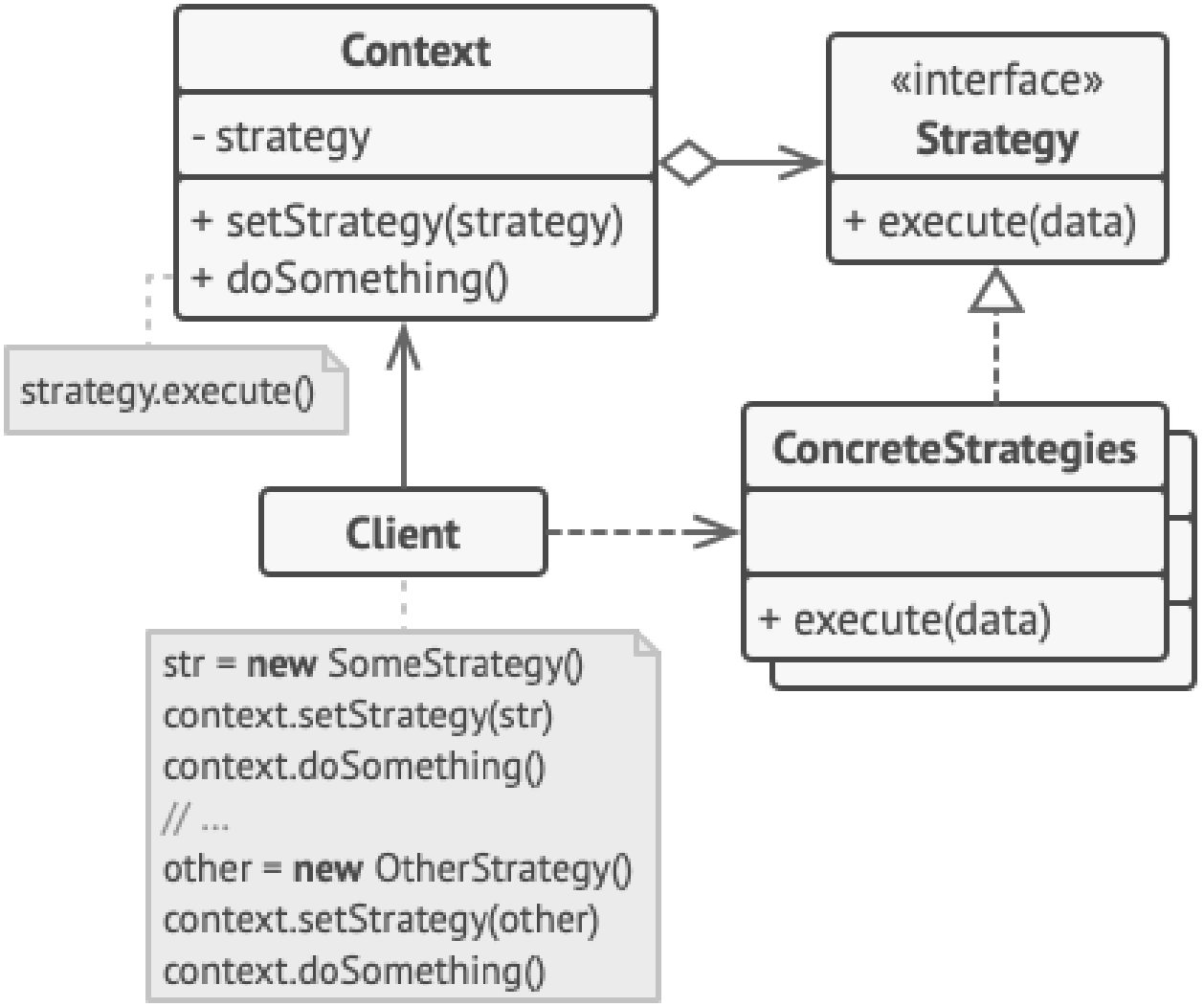
Objetivos

- Aplicar padrão comportamental ***Strategy*** em situação problema.



21. Stategy

Estrutura

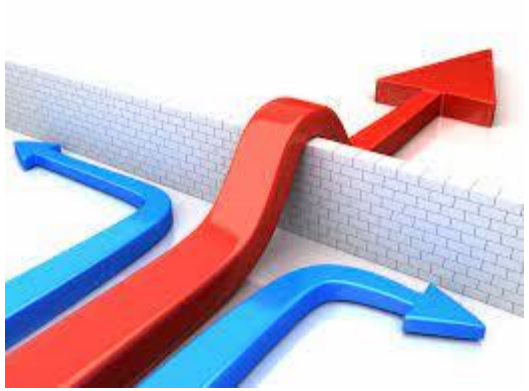


https://refactoring.guru/pt-br/design-patterns/state

https://github.com/osmarbraz/strategy

Importante

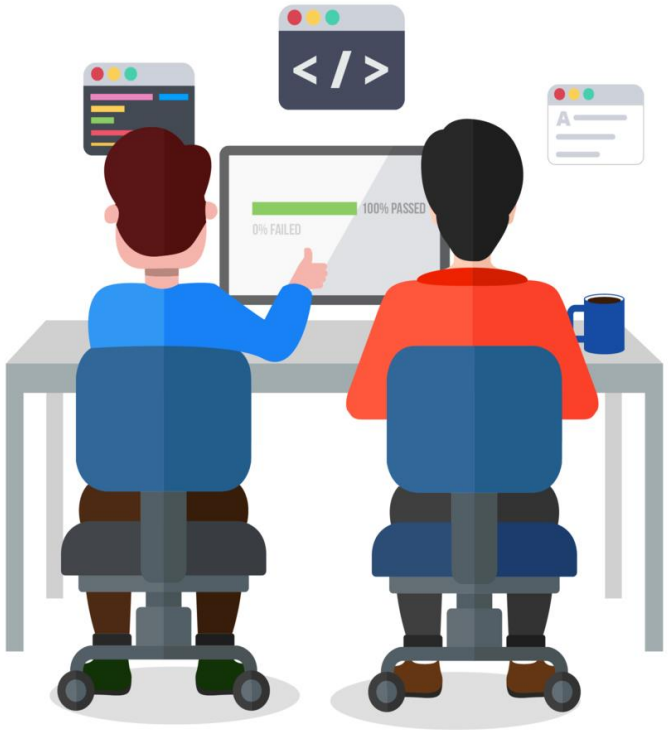
Siga os ROTEIROS !!!



Atividade em Grupo

Para esta atividade crie grupos de 2 alunos, para desenvolver a atividade segundo ***Pair Programming.***

Navegador



Piloto

Extreme Programming, metodologia ágil criada por Kent Beck em 1996.

Pair Programming

- Um é o **piloto**, responsável por escrever o código, o outro o navegador, acompanha a escrita de código e verificar se está de acordo com os **padrões do projeto** e de encontro à solução necessária.
- A intenção desta técnica é **evitar** erros de lógica, e ter um código mais confiável e melhor estruturado, utilizando-se para isso a máxima de que “**duas cabeças pensam melhor do que uma**”.

Preparação do ambiente

- Acesso a ferramenta **draw.io(https://app.diagrams.net/)** para realizar a modelagem.
- Escolha a sua linguagem de programação de preferência
- Escolha uma IDE ou o **git.dev**
- Crie um repositório no github(https://github.com/) para que todos os membros da equipe possam colaborar no desenvolvimento.

Atividade prática

1

21. Strategy
Propósito

- Definir uma família de algoritmos e permitir que um objeto possa escolher qual algoritmo da família utilizar em cada situação.
- Também conhecido como: Estratégia ou Policy.

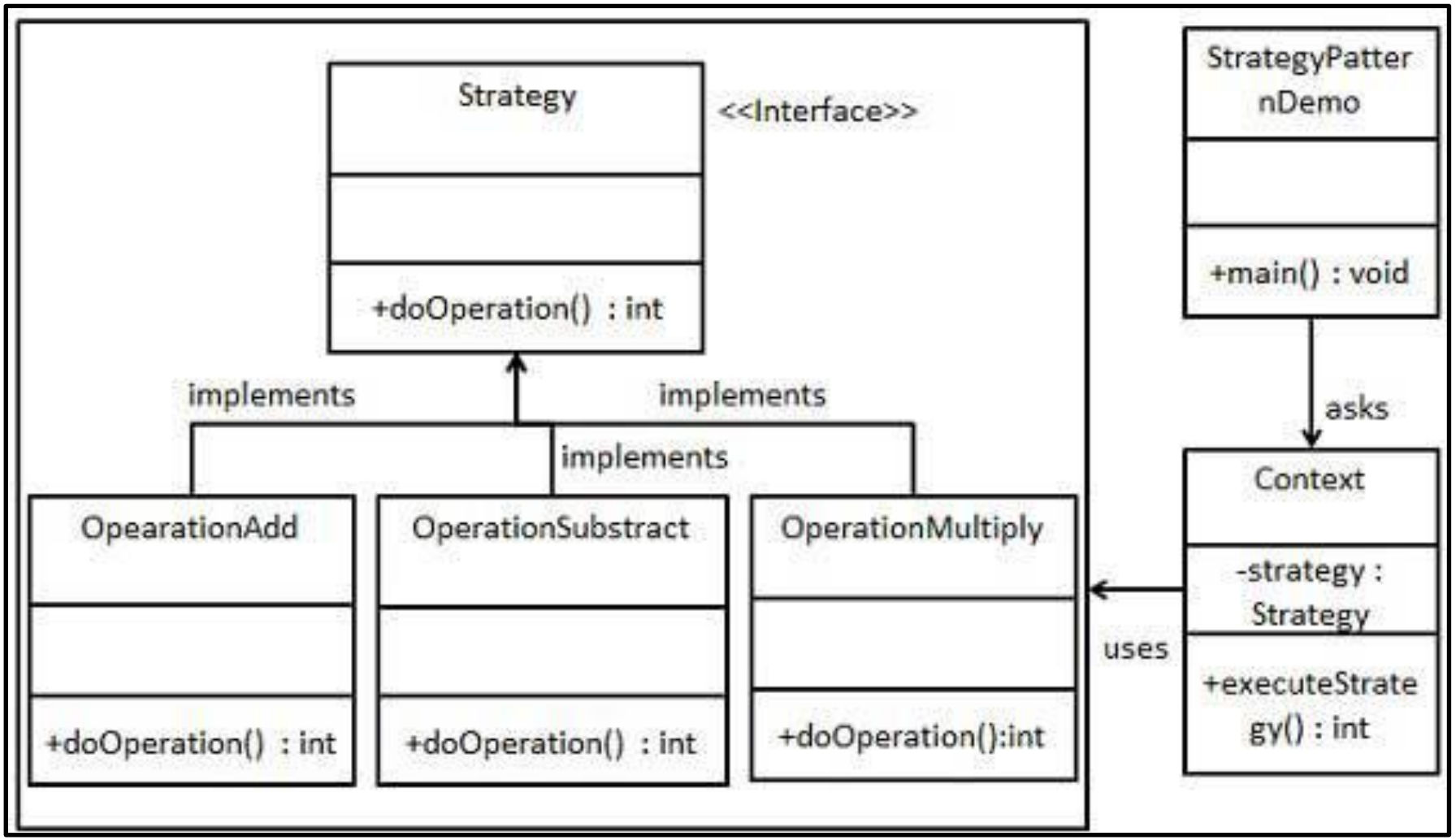
21. Stategy

- Usar este padrão quando…
  - Várias classes diferentes diferem-se somente no comportamento;
  - Você precisa de variantes de um mesmo algorítmo;
  - Um algoritmo utiliza dados que o cliente não deve conhecer;
  - Uma classe define múltiplos comportamentos, escolhidos num grande condicional.

21. Stategy

- Vantagens e desvantagens
  - Famílias de algorítmos:
    - Beneficiam-se de herança e polimorfismo.
  - Alternativa para herança do cliente:
    - Comportamento é a única coisa que varia.
  - Eliminam os grandes condicionais:
    - Evita código monolítico.
  - Escolha de implementações:
    - Pode alterar a estratégia em runtime.
  - Clientes devem conhecer as estratégias:
    - Eles que escolhem qual usar a cada momento.
  - Parâmetros diferentes para algoritmos diferentes:
    - Há possibilidade de duas estratégias diferentes terem interfaces distintas.
  - Aumenta o número de objetos:
    - Este padrão aumenta a quantidade de objetos pequenos presentes na aplicação.

21. Stategy


Strategy
<<Interface>>
+doOperation() : int
implements
implements
implements
OpearationAdd
+doOperation() : int
OperationSubstract
+doOperation() : int
OperationMultiply
+doOperation():int
StrategyPatternDemo
+main() : void
asks
Context
-strategy : Strategy
+executeStrategy() : int
uses

21. Strategy
Passo 1

Crie uma interface.

Strategy.java

```
public interface Strategy {
   public int doOperation(int num1, int num2);
}
```

21. Strategy
Passo 2

Crie classes concretas implementando a mesma interface.

OperationAdd.java

```
public class OperationAdd implements Strategy{
   @Override
   public int doOperation(int num1, int num2) {
      return num1 + num2;
   }
}
```

OperationSubstract.java

```
public class OperationSubstract implements Strategy{
   @Override
   public int doOperation(int num1, int num2) {
      return num1 - num2;
   }
}
```

21. Strategy
Passo 2 - Continuação

OperationMultiply.java

```
public class OperationMultiply implements Strategy{
   @Override
   public int doOperation(int num1, int num2) {
      return num1 * num2;
   }
}
```

21. Strategy
Passo 3

Criar classe de contexto.

Context.java

```
public class Context {
   private Strategy strategy;

   public Context(Strategy strategy){
      this.strategy = strategy;
   }

   public int executeStrategy(int num1, int num2){
      return strategy.doOperation(num1, num2);
   }
}
```

21. Strategy
Passo 4

Use o contexto para ver a mudança de comportamento quando muda sua estratégia.

StrategyPatternDemo.java

```
public class StrategyPatternDemo {
   public static void main(String[] args) {
      Context context = new Context(new OperationAdd());
      System.out.println("10 + 5 = " + context.executeStrategy(10, 5));

      context = new Context(new OperationSubstract());
      System.out.println("10 - 5 = " + context.executeStrategy(10, 5));

      context = new Context(new OperationMultiply());
      System.out.println("10 * 5 = " + context.executeStrategy(10, 5));
   }
}
```

21. Strategy
Passo 5

- Terminamos
  - Teste sua implementação

Compile e **Mostre** o código para o professor
  - Pense, o que você fez aqui ?

Lembre de salvar no seu github

Conclusão

Os padrões comportamentais tem como principal função designar responsabilidades entre objetos.

Referências

- PRESSMAN, Roger; MAXIM, Bruce. Engenharia de software: uma abordagem profissional. 8.ed. Bookman, 2016. E-book. Disponível em: https://integrada.minhabiblioteca.com.br/books/9788580555349

- SOMMERVILLE, Ian. Engenharia de software. 9. ed. São Paulo: Pearson Prentice Hall, 2011. E-book. Disponível em: https://plataforma.bvirtual.com.br/Leitor/Publicacao/2613/epub/0

- LARMAN, Craig. Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e desenvolvimento iterativo. 3. ed Porto Alegre: Bookman, 2007. E-book. Disponível em: https://integrada.minhabiblioteca.com.br/books/9788577800476

Fim