

Universidade Federal de Santa Catarina  
Departamento de Informática e Estatística  
INE5614 - Engenharia de Software

# PADRÃO STRATEGY

Membros:

Alceu Ramos Conceição Júnior  
João Pedro Santana  
Rafael Lazzaretti Madalóz

# Objetivo

- Tem como objetivo encapsular diferentes implementações de algoritmos, de forma que possa ser chamado a implementação conforme a estratégia do cliente.

# Motivação

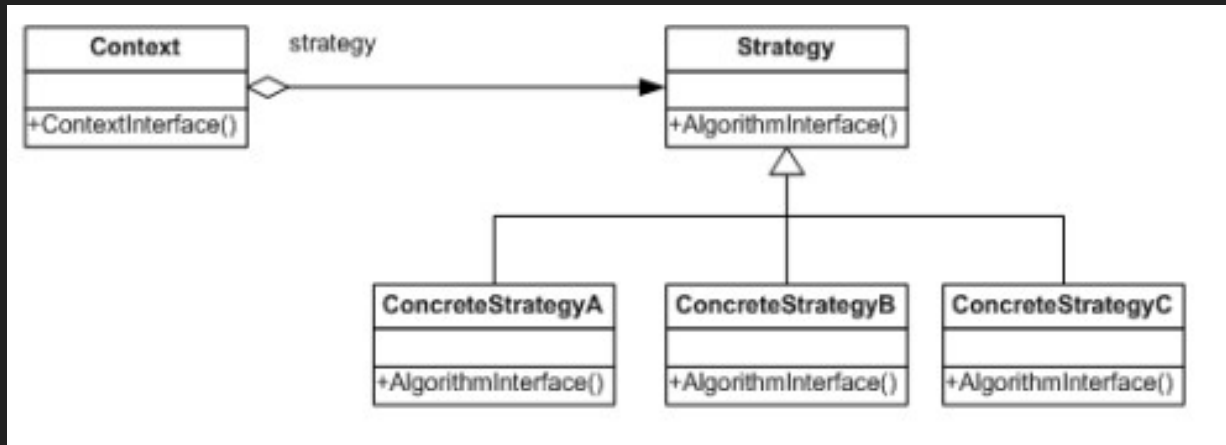
- Lógicas condicionais (if, else, switch case) tendem a crescer e tornar-se cada vez mais sofisticadas, maiores e mais difíceis de manter com o passar do tempo.
- O padrão Strategy ajuda a gerenciar toda essa complexidade. Como?
- Produzindo uma família de classes para cada variação do algoritmo e fornecendo para a classe hospedeira uma instância de Strategy para a qual ela delegará em tempo de execução.

# Aplicabilidade

- Quando necessita-se de variantes de um algoritmo;
- Quando se precisa ocultar do usuário a exposição das estruturas de dados complexas, específicas do algoritmo;
- Quando uma classe define muitos comportamentos e por sua vez eles aparecem como diversos “IFs”. Com isso esses comandos condicionais são movidos para sua própria classe Strategy.

# Estrutura

- Conforme a imagem abaixo, o contexto contém uma interface da estratégia, que define qual implementação de ser requisitada. Com isso, as classes ConcreteStrategyA, ConcreteStrategyB e ConcreteStrategyC são classes concretas que implementam os métodos da interface Strategy.



# Estrutura

- **Strategy:** É uma interface comum para todas as subclasses, ou para todos os algoritmos que são suportados. O Contexto usa essa interface para chamar uma das subclasses ConcreteStrategy ou um dos algoritmos definidos.
- **ConcreteStrategy:** A classe concreta que herda da Strategy abstrata está definida como as subclasses ConcreteStrategyA, ConcreteStrategyB e ConcreteStrategyC no diagrama.
- **Context:** É aquele que vai acessar um dos algoritmos das subclasses de interface Strategy.

# Vantagens

- Reutilização por parte do Contexto que permite escolher entre uma família de algoritmos que possuem funcionalidades em comum;
- Os algoritmos em classes Strategy possuem variações do seus algoritmos independentemente do seu contexto, assim é mais fácil utilizá-los, trocá-los, compreende-los e estende-los;
- Diminuição ou eliminação da lógica condicional clarificando ainda mais os algoritmos;
- Permite que se escolham diferentes implementações do mesmo comportamento;

# Vantagens

- Há uma grande simplificação na classe ao mover variações de um algoritmo para uma hierarquia;
- Habilita-se que um algoritmo seja substituído por outro em tempo de execução.



# Desvantagens

- Complicação que há de como os algoritmos obtém ou recebem dados de suas classes de contexto;
- O cliente deve conhecer como que os Strategies diferem, antes mesmo que ele possa selecionar um mais apropriado para o contexto da aplicação;
- O custo da comunicação entre o contexto e o Strategy é significativo, dado que os Strategies concretos não necessariamente usarão todas as informações da Strategy abstrata, portanto podem haver situações em que o contexto criará e inicializará parâmetros que nunca serão usados;

# Desvantagens

- Aumentam o número de objetos no sistema, que pode ser ruim em determinadas situações em termos de custo e por fim pessoas inexperiente podem ter dificuldade sobre o funcionamento do código por não entender o que é e como funciona o padrão.

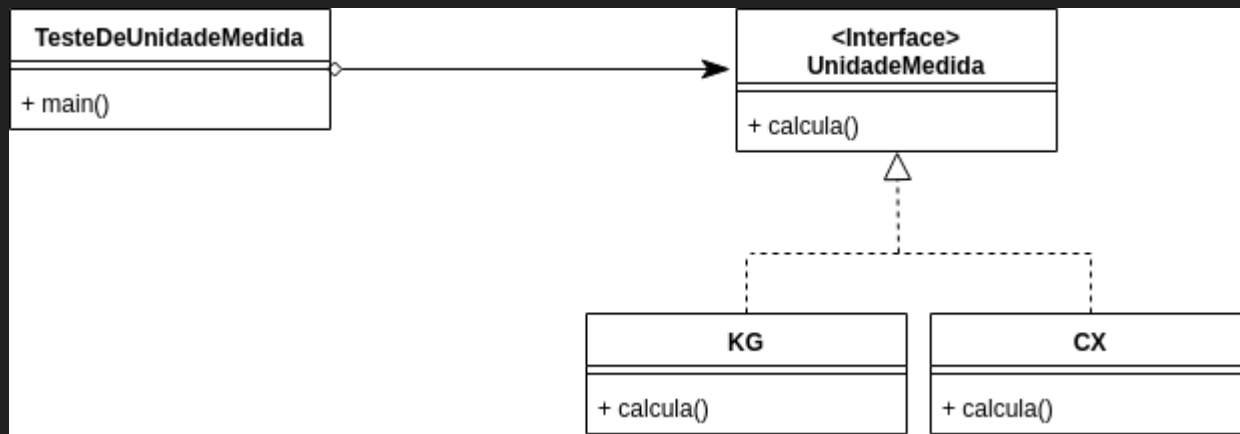
- Por exemplo, no código abaixo poderia ser aplicado o padrão Strategy:

```
1  public class CalculadorDeUnidades {
2
3      public static void main(String[] args) {
4          final Pedido pedido = new Pedido();
5          realizaCalculo(pedido, "Kg");
6          realizaCalculo(pedido, "Cx");
7      }
8
9      public void realizaCalculo(Pedido pedido, String unidadeMedida) {
10         if( "KG".equals(unidadeMedida) ) {
11             BigDecimal kg = pedido.getQuantidade().multiply(BigDecimal.valueOf(0.01));
12         } else if( "Cx".equals(unidadeMedida) ) {
13             BigDecimal cx = pedido.getQuantidade().multiply(BigDecimal.valueOf(0.045));
14         }
15     }
16 }
```

Há dois problemas nessa implementação:

- As implementações dos cálculos estão dentro de IFs, o que aumenta a complexidade do código e provê uma desorganização maior no código, onde em caso da existência de 10 unidades de medidas, teriam 10 IFs aninhados.
- A variável que define a unidade de medida é uma String, ou seja, não tem um tipo definido o que pode possibilitar um erro na chamada do cálculo, caso o texto contido não esteja exatamente igual ao definido dentro do IF.

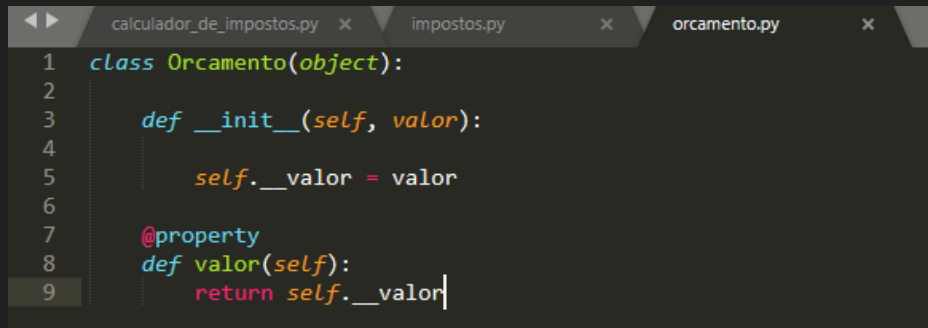
```
1 public interface UnidaMedida {
2     BigDecimal calcula(Pedido pedido);
3 }
4
5 public class Kg implements UnidaMedida {
6     public BigDecimal calcula(Pedido pedido) {
7         return pedido.getQuantidade().multiply(BigDecimal.valueOf(0.01));
8     }
9 }
10
11 public class Cx implements UnidaMedida {
12     public BigDecimal calcula(Pedido pedido) {
13         return pedido.getQuantidade().multiply(BigDecimal.valueOf(0.045));
14     }
15 }
16 public class TesteDeUnidadeMedida {
17
18     public static void main(String[] args) {
19         final UnidaMedida kg = new Kg();
20         final UnidaMedida cx = new Cx();
21
22         final Pedido pedido = new Pedido(10);
23
24         final CalculadorDeQuantidade calculador = new CalculadorDeQuantidade();
25         calculador.realizaCalculo(pedido, kg);
26         calculador.realizaCalculo(pedido, cx);
27     }
28 }
```



# Padrão STRATEGY

Classe 'Orçamento' é um objeto criado para fins de exemplo.

Possui um único atributo: 'valor', e um único método, utilizado para retornar 'valor'.



```
1 class Orcamento(object):
2
3     def __init__(self, valor):
4
5         self.__valor = valor
6
7     @property
8     def valor(self):
9         return self.__valor
```

# Padrão STRATEGY

Classes 'ISS' e 'ICMS' implementam a interface 'Imposto', e devem conter um método chamado 'calcula'.

```
calculador_de_impostos.py x impostos.py x orcamento.py x
1 class Imposto(object):
2
3     def calcula(self,orcamento):
4         raise NotImplementedError("Impostos precisam implementar 'calcula()'")
5
6 class ISS(Imposto):
7
8     def calcula(self, orcamento):
9         return orcamento.valor * 0.1
10
11 class ICMS(Imposto):
12     def calcula(self, orcamento):
13
14         return orcamento.valor * 0.06
15
16 class ITeste(Imposto):
17     pass
```



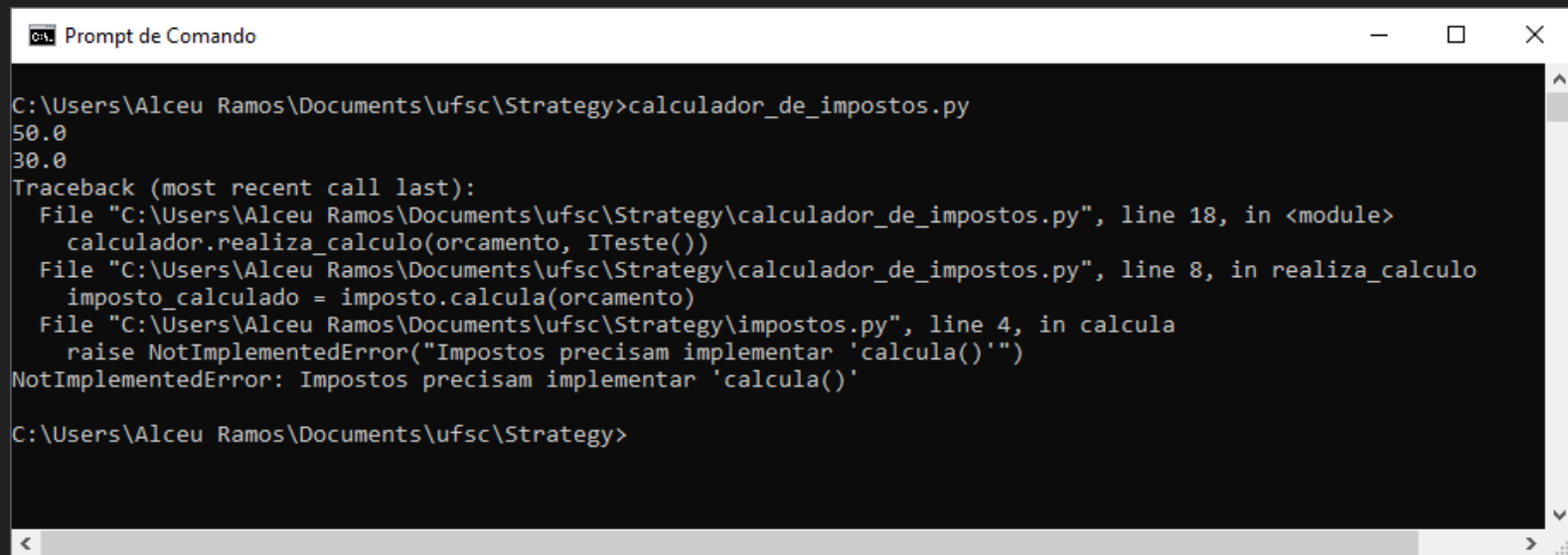
# Padrão STRATEGY

A Classe 'Calculador\_de\_impostos' representa um objeto que possui um único método, 'realiza\_calculo', o qual tem como parâmetros um 'Orcamento' e um 'Imposto'.

```
calculador_de_impostos.py x impostos.py x orcamento.py x
1 from impostos import ISS, ICMS, ITeste
2 from orcamento import Orcamento
3
4 class Calculador_de_impostos(object):
5
6     def realiza_calculo(self, orcamento, imposto):
7
8         imposto_calculado = imposto.calcula(orcamento)
9         print(imposto_calculado)
10
11 if __name__ == "__main__":
12
13     calculador = Calculador_de_impostos()
14     orcamento = Orcamento(500)
15
16     calculador.realiza_calculo(orcamento, ISS())
17     calculador.realiza_calculo(orcamento, ICMS())
18     calculador.realiza_calculo(orcamento, ITeste())
```

# Padrão STRATEGY

## Execução do código



```
Prompt de Comando

C:\Users\Alceu Ramos\Documents\ufsc\Strategy>calculador_de_impostos.py
50.0
30.0
Traceback (most recent call last):
  File "C:\Users\Alceu Ramos\Documents\ufsc\Strategy\calculador_de_impostos.py", line 18, in <module>
    calculador.realiza_calculo(orcamento, ITeste())
  File "C:\Users\Alceu Ramos\Documents\ufsc\Strategy\calculador_de_impostos.py", line 8, in realiza_calculo
    imposto_calculado = imposto.calcula(orcamento)
  File "C:\Users\Alceu Ramos\Documents\ufsc\Strategy\impostos.py", line 4, in calcula
    raise NotImplementedError("Impostos precisam implementar 'calcula()'")
NotImplementedError: Impostos precisam implementar 'calcula()'

C:\Users\Alceu Ramos\Documents\ufsc\Strategy>
```

# Referências

<https://pt.stackoverflow.com/questions/72685/existe-interfaces-no-python>

<https://github.com/kelvins/design-patterns-python/tree/master/comportamentais/strategy>

<https://emmanuelneri.com.br/2016/11/19/padrao-strategy/>

<https://www.devmedia.com.br/estudo-e-aplicacao-do-padrao-de-projeto-strategy/25856>