Design Patterns

Classificação (GoF)

		Propósito		
		1. Criação	2. Estrutura	3. Comportamento
Escopo	Classe	Factory Method	Class Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Object Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Strategy

"Definir uma família de algoritmos, encapsular cada um, e fazê-los intercambiáveis. Strategy permite que algoritmos mudem independentemente entre clientes que os utilizam." [GoF]

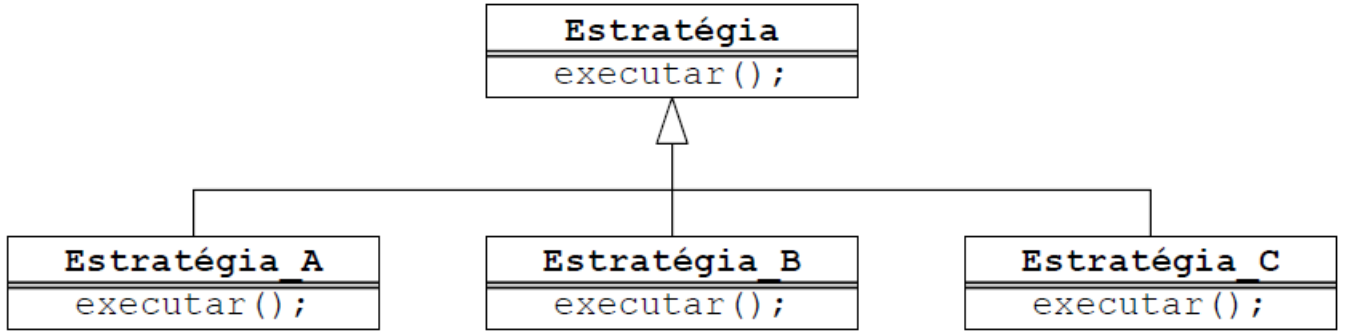
Problema

Várias estratégias, escolhidas de acordo com opções ou condições

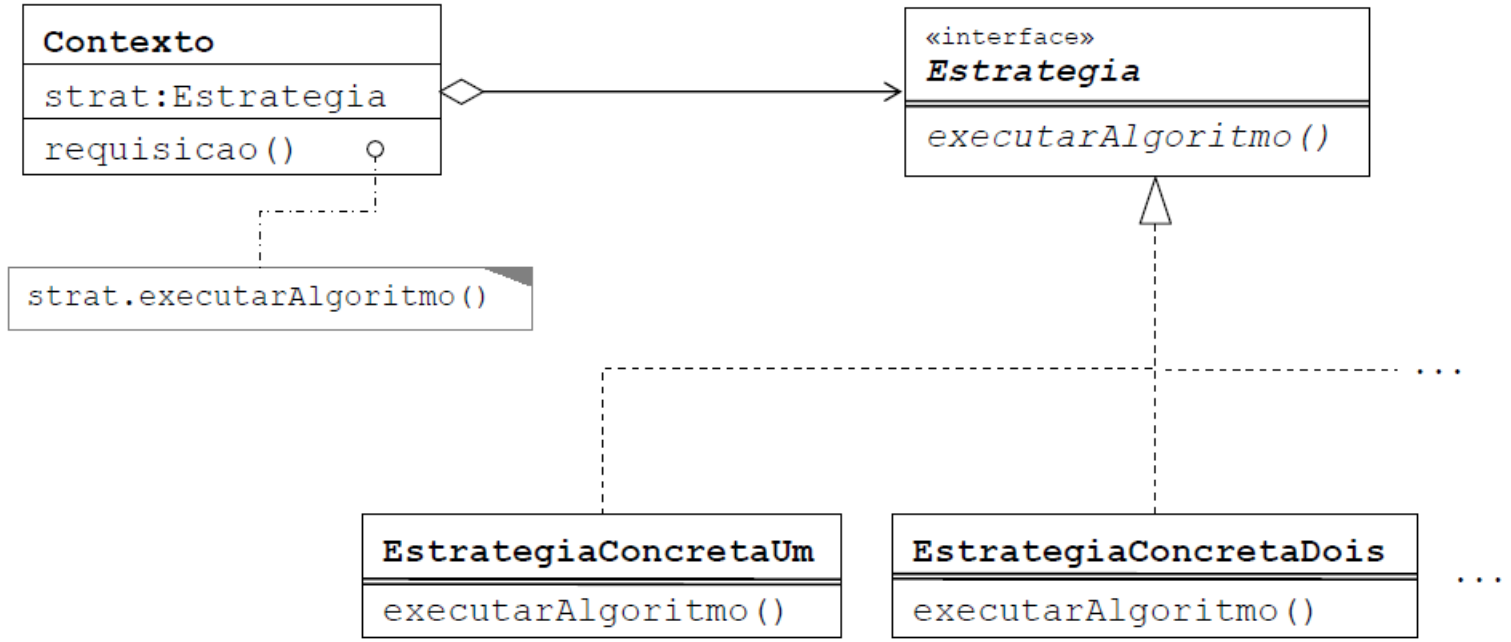
```
if (guerra && inflação > META) {
   doPlanoB();
else if (guerra && recessão) {
   doPlanoC();
} else {
   doPlanejado();
}
```

```
if (guerra && inflação > META) {
   plano = new Estrategia_C();
} else if (guerra && recessão) {
   plano = new Estrategia_B();
} else {
   plano = new Estrategia_A();
}
```

```
plano.executar();
```



Estrutura



- *Um contexto repassa requisições de seus clientes para sua estratégia. Clientes geralmente criam e passam uma EstategiaConcreta para o contexto. Depois, clientes interagem apenas com o contexto*
- *Estrategia e Contexto interagem para implementar o algoritmo escolhido. Um contexto pode passar todos os dados necessários ou uma cópia de si próprio*

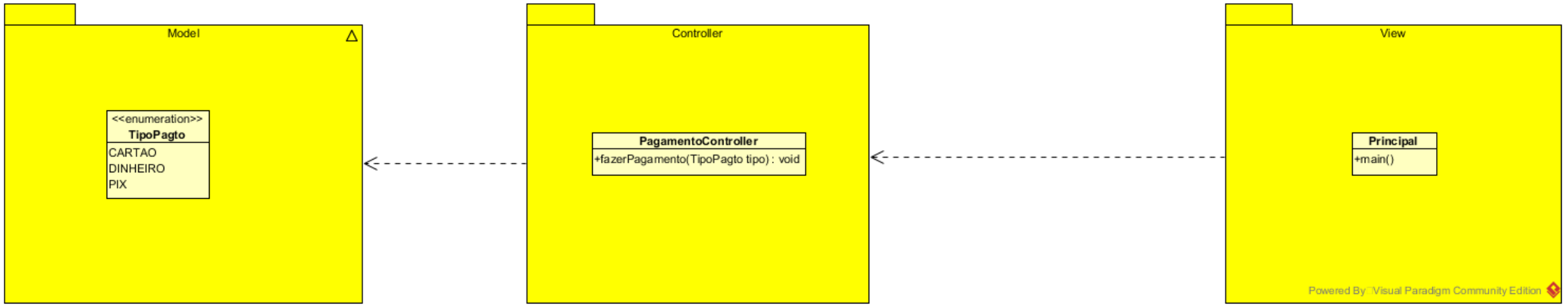
Quando usar ?

- *Quando classes relacionadas forem diferentes apenas no seu comportamento*
  - *Strategy oferece um meio para configurar a classe com um entre vários comportamentos*
- *Quando você precisar de diferentes variações de um mesmo algoritmo*
- *Quando um algoritmo usa dados que o cliente não deve conhecer*
- *Quando uma classe define muitos comportamentos, e estes aparecem como múltiplas declarações condicionais em suas operações*

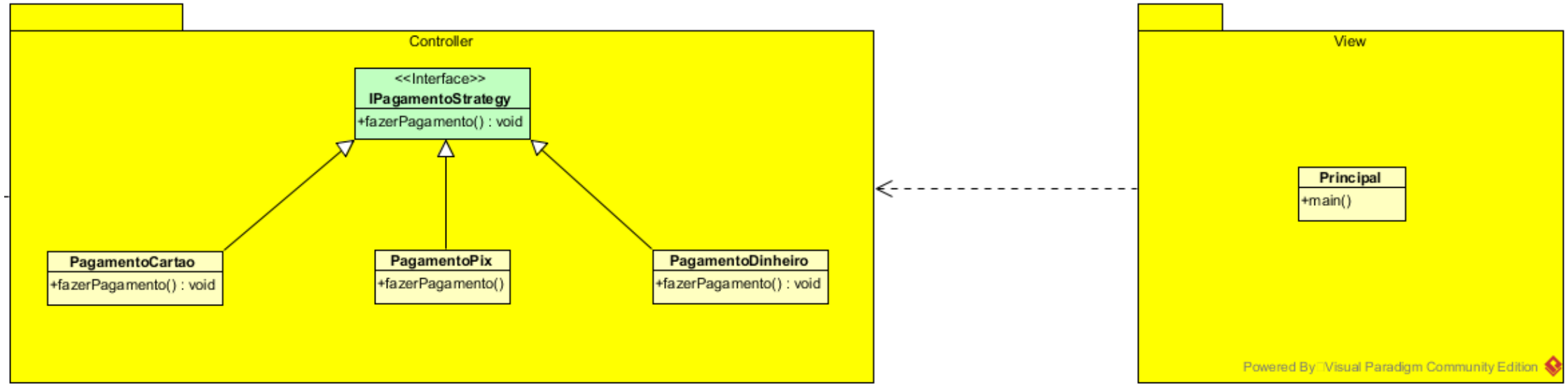
Prática e aplicação em Java

- Considere uma aplicação que permita realizar pagamento de diversas formas
  - Inicialmente, **Cartão de Crédito**, **Pix** e **Dinheiro**
- Para pagamento em cartão, se deve solicitar o nome do titular, o CPF, o número do cartão, o CVV e fazer o processamento
- Para pagamento via Pix, deve se exibir a chave para a transação
- Para pagamento em dinheiro, é necessário perguntar o valor em moeda para verificar a necessidade de troco
- O diagrama a seguir mostra a estrutura inicial

Implementar



Strategy


Controller
<<Interface>>
IPagamentoStrategy
+fazerPagamento() : void
PagamentoCartao
+fazerPagamento() : void
PagamentoPix
+fazerPagamento()
PagamentoDinheiro
+fazerPagamento() : void
View
Principal
+main()
Powered By Visual Paradigm Community Edition

Exercício

- Considere uma aplicação feita para calcular impostos urbanos nas cidades do país. O cálculo não é muito complexo, mas varia de cidade para cidade. Inicialmente, se fará para São Paulo, Belo Horizonte, Porto Alegre e Curitiba.
  - São Paulo ➔ Imposto = área total * 10 + número de cômodos * 2
  - Belo Horizonte ➔ Imposto = área total * 7 + número de quartos * 4
  - Porto Alegre ➔ Imposto = área total * 7,5 + área da garagem * 2,5
    - Se não houver garagem ➔ Imposto = área total * 8
  - Curitiba ➔ Imposto = área total * 5
    - Se idade do imóvel > 50 anos, somar idade * 3
    - Se idade do imóvel < 20 anos, somar idade * 2
    - Se 20 anos > idade do imóvel > 50 anos, somar idade * 2,5
- Fazer a solução em Java e criar o diagrama de pacotes da solução proposta usando o Strategy Design. Considere usar o Builder Pattern também