State 💻

Análise e Desenvolvimento de Sistemas - 6º Período

State

- Até o momento implementamos objetos que executam operações baseado em um **"estado"**
- Quando chamamos métodos desses objetos, seus estados, ou propriedades, internos podem ser alterados
- Um objeto pode ter a necessidade de **executar ações quando seu estado interno muda**

State

- Imagine o funcionamento de uma máquina de vendas automática
- Essa máquina **inicia com um estado**: **Sem crédito**
- Você deve inserir o dinheiro nessa máquina. Uma vez que você executa essa ação, seu estado interno muda para: **Com crédito**
- No primeiro estado, e execução da tarefa "Selecionar produto" não seria executada. Mas após o segundo estado, essa tarefa pode ser executada

State

- Isso significa que, com a mudança do estado interno do nosso objeto, representado aqui pela máquina de vendas, uma determinada ação (Selecionar produto) pode retornar diferentes resultados
- Esse é o comportamento que queremos implementar: **Ações diferentes para estados diferentes**

State

- Uma maneira comum de implementar isso seria usando condicionais ou switch-case
- No entanto, se a quantidade de estados for grande, a lógica das nossas condicionais pode ficar bastante complexa
- A manutenção desse código se torna muito trabalhosa, já que vamos precisar mudar a lógica das condições
- Portanto, o padrão **State sugere a criação de uma classe para cada estado possível**

State

- O objeto deve se comportar de maneiras diferentes para cada estado que se encontra
- Esses comportamentos são implementados em métodos. Cada classe que representa um estado fará sua implementação dos comportamentos

State

- O objeto original, chamado aqui de **Contexto** possui uma propriedade que representa seu estado interno
- Inicialmente esse estado faz referência a algum dos objetos estado que delega a execução para outros objetos estado de acordo com a mudança do estado interno
- Ou seja, a atualização dessa propriedade que representa o estado interno, representa a mudança de estado do objeto original

Strategy Vs State

- Perceba que o State possui **duas características principais:**
  - **Encapsulamento**. Delegamos a responsabilidade de implementação de cada estado para sua classe
  - **Estado interno**. Implementamos uma propriedade que representa o estado interno do objeto.

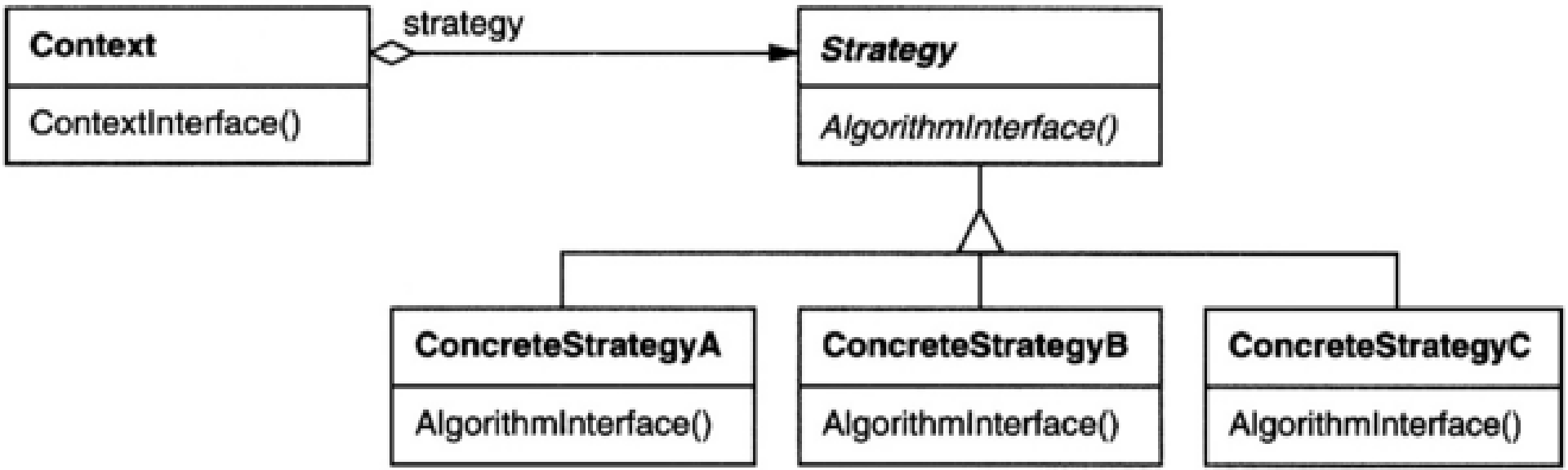
Strategy Vs State

- A característica de **Encapsulamento** nada mais é que a aplicação de um padrão chamado **Strategy**
- O Strategy propõe a divisão de uma **família de algoritmos em diferentes classes**
- Além desse encapsulamento, o Strategy permite que esses objetos que vão representar os algoritmos sejam intercambiáveis. Isto é, escolhemos o algoritmos em tempo de execução

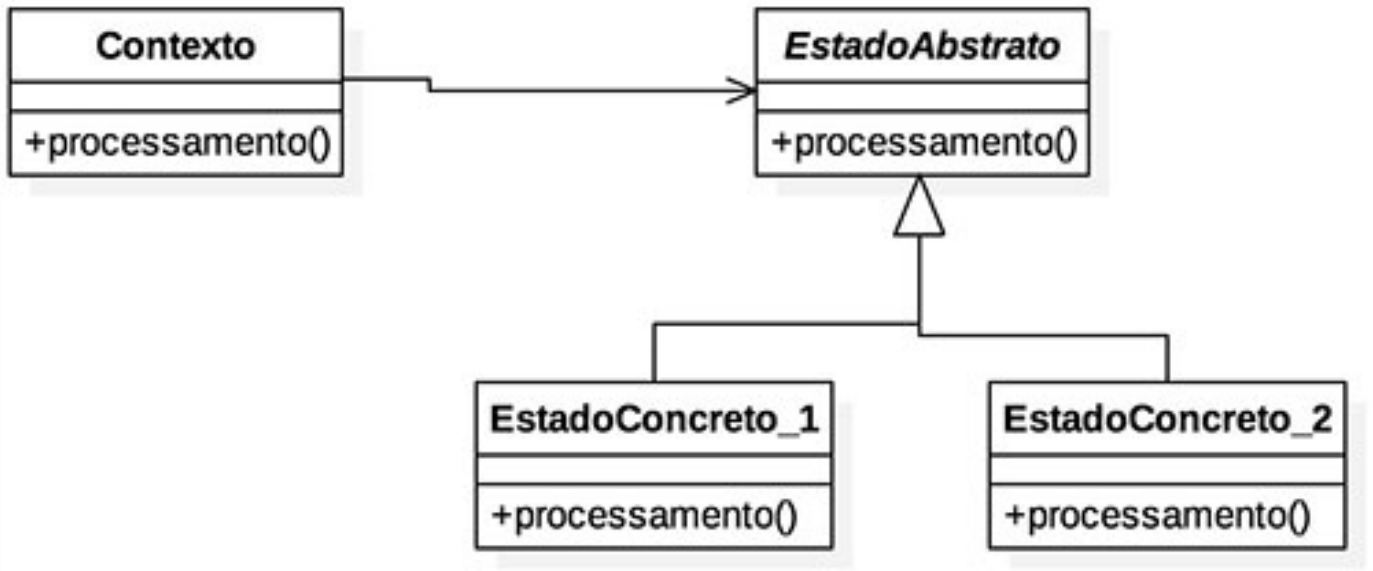
Strategy Vs State

- Na implementação que fizemos com o State, simulando uma máquina de vendas, **usamos o Strategy para encapsular cada estado e também possibilitar que um objeto estado chame o outro (intercambiáveis)**
- Por esse motivo, dizemos que o State **é uma extensão do Strategy**

Strategy


Context
ContextInterface()
strategy
Strategy
AlgorithmInterface()
ConcreteStrategyA
AlgorithmInterface()
ConcreteStrategyB
AlgorithmInterface()
ConcreteStrategyC
AlgorithmInterface()

State


Contexto
+processamento()
EstadoAbstrato
+processamento()
EstadoConcreto_1
+processamento()
EstadoConcreto_2
+processamento()