Padrões de Projeto de Software Orientados a Objetos

Tecnologia em Análise e Desenvolvimento de Sistemas

Paulo Mauricio Gonçalves Júnior

Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco

21 de maio de 2018

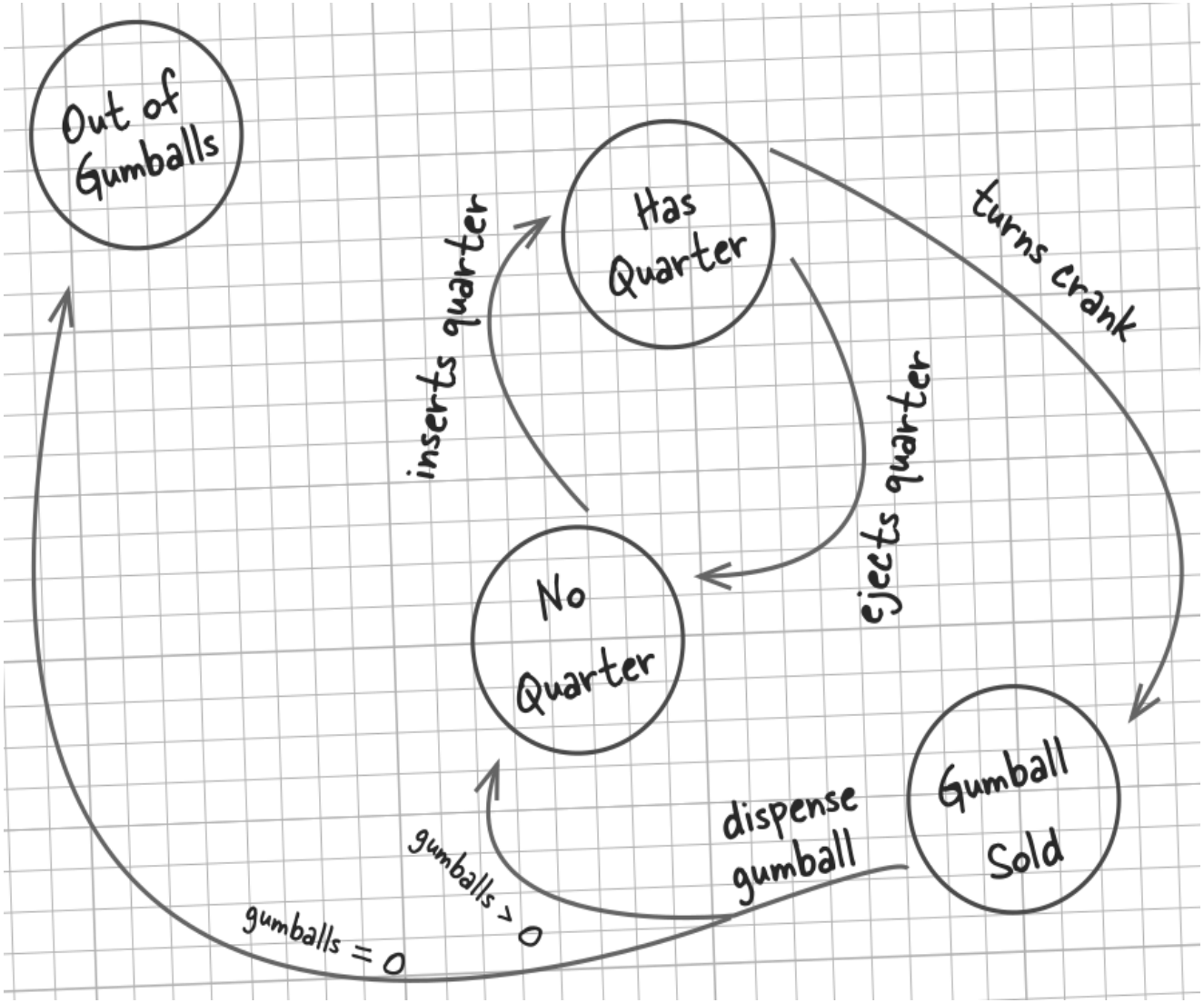
Parte I

State

Introdução I

- Strategy e State são padrões semelhantes. Strategy permite intercambiar algoritmos enquanto State permite controlar seu comportamento modificando seu estado interno.
- Vamos implementar uma máquina de chicletes.

Introdução II



Introdução III

- Solução inicial:
  1. Identificar os estados
  2. Criar uma variável para manter o estado atual, e definir valores para cada estado:

```
final static int SOLD_OUT = 0;
final static int NO_QUARTER = 1;
final static int HAS_QUARTER = 2;
final static int SOLD = 3;
int state = SOLD_OUT;
```

  3. Identificar todas as ações que podem acontecer no sistema.
  4. Criar uma classe que atua como uma máquina de estado. Para cada ação, um método é criado que usa condições para determinar qual comportamento é apropriado em cada estado.
- Uma modificação é solicitada: 10% das vezes, ao puxar a manivela o cliente obtém dois chicletes ao invés de um.
- Modificações:

Introdução IV

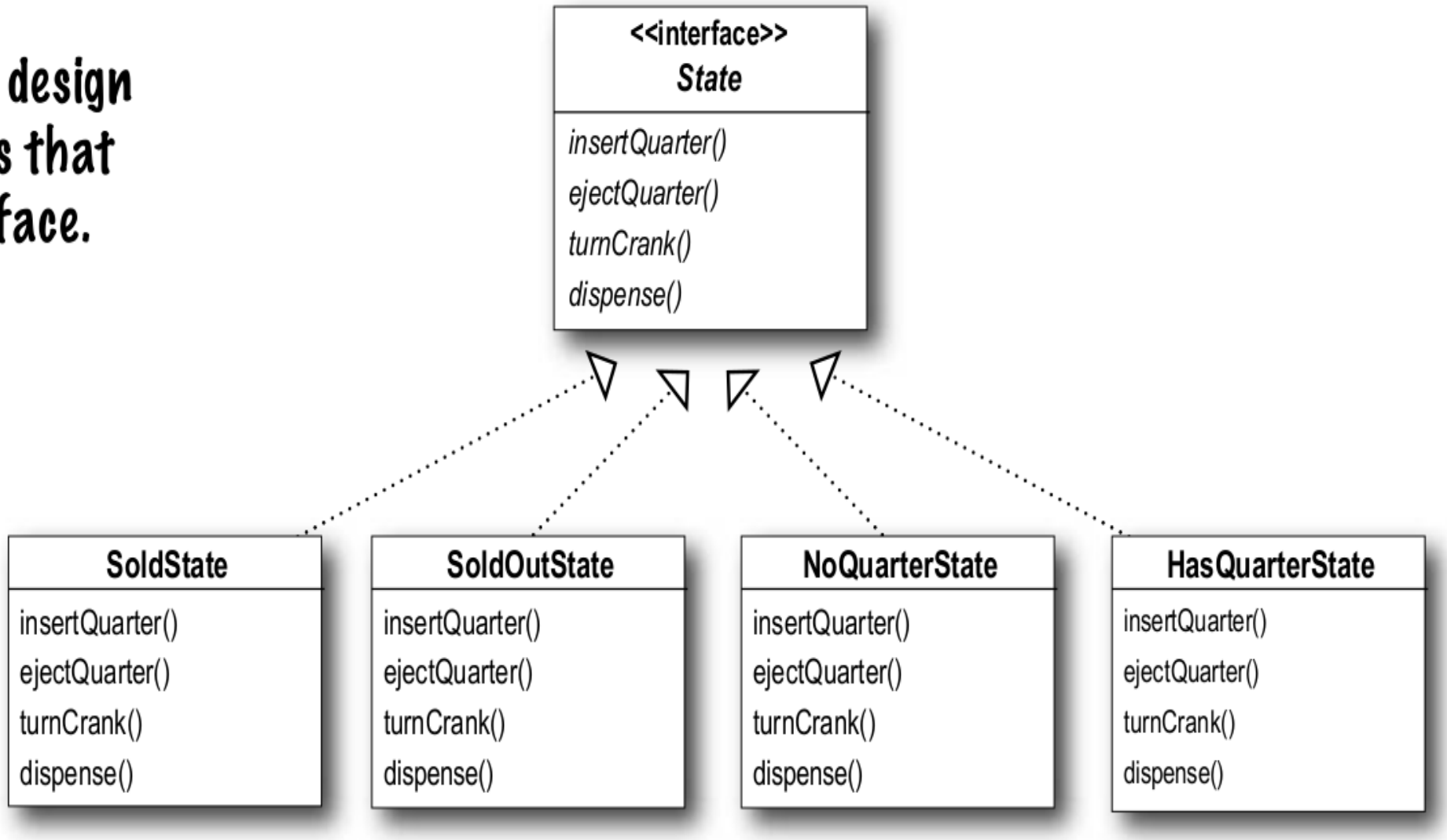
1. Adicionar uma variável para o estado winner.
2. Adicionar uma condição em todos os métodos para gerenciar este novo estado.
3. O método de puxar a manivela tem que ser modificado para acrescentar a verificação de ganhar um chiclete a mais.

Solução I

- Definir uma interface State que contém um método para cada ação na máquina de chicletes.
- Implementamos uma classe State para cada estado possível. Essas classes serão responsáveis pelo comportamento em cada estado.
- Remover todo o código condicional.

Solução II

design
s that
face.



Solução III

```
public class NoQuarterState implements State {
  GumballMachine gumballMachine;
  public NoQuarterState(GumballMachine gumballMachine) {
    this.gumballMachine = gumballMachine;
  }
  public void insertQuarter() {
    System.out.println("You inserted a quarter");
    gumballMachine.setState(gumballMachine.getHasQuarterState());
  }
  public void ejectQuarter() {
    System.out.println("You haven't inserted a quarter");
  }
  public void turnCrank() {
    System.out.println("You turned, but there's no quarter");
  }
  public void dispense() {
    System.out.println("You need to pay first");
  }
}
```

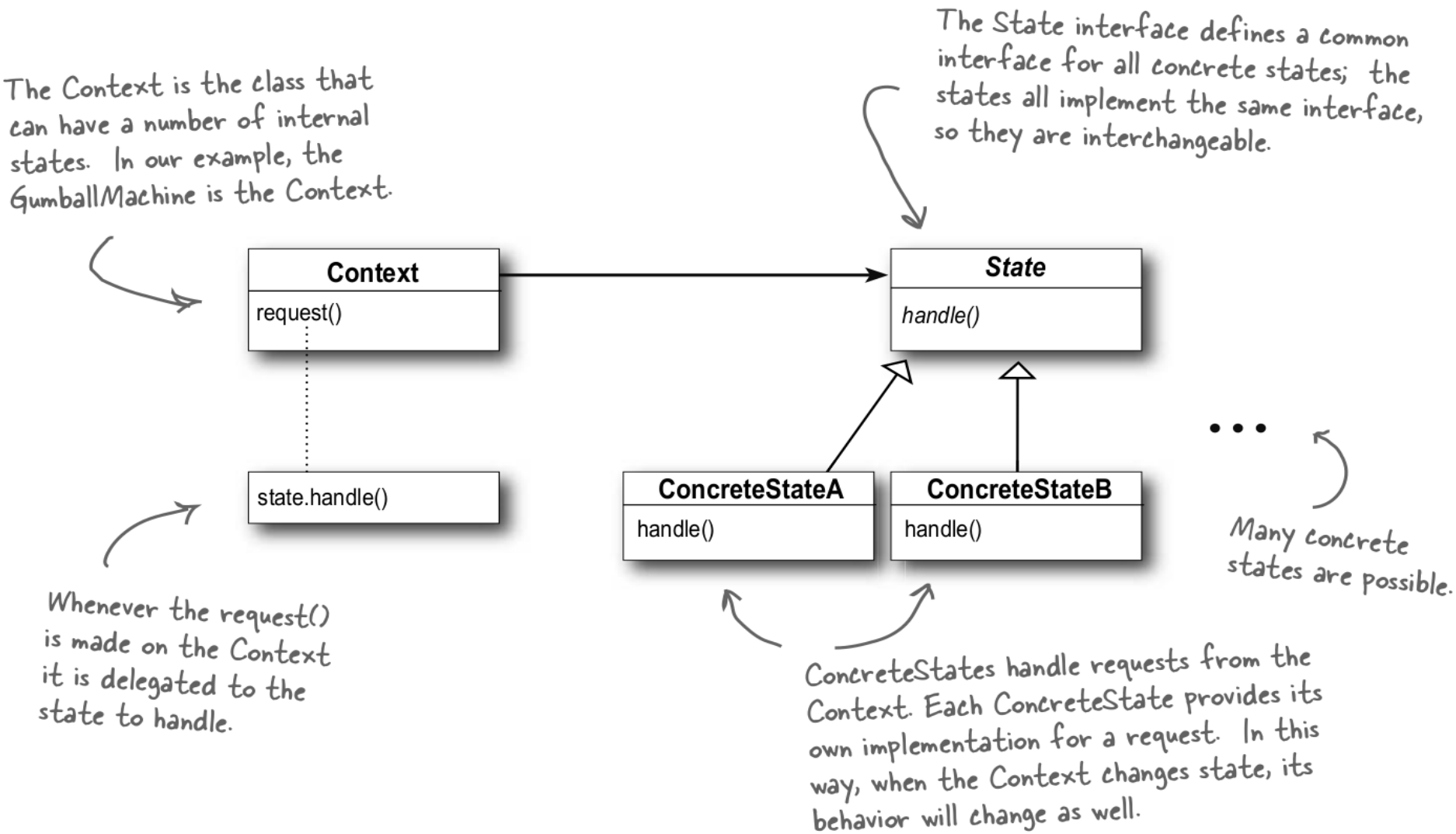
Solução IV

- O comportamento de cada estado agora está localizado em cada classe.
- Removemos todos os desvios condicionais.
- Cada estado está fechado para modificação, mas a máquina de chicletes abertos para extensão, adicionando novas classes.
- Código base criado e estrutura de classe que é mais condizente com o diagrama da máquina e mais fácil de ler e entender.

Definição

O padrão State permite a um objeto alterar seu comportamento quando seu estado interno muda. O objeto aparentará ter mudado sua classe.

Solução V


The Context is the class that can have a number of internal states. In our example, the GumballMachine is the Context.
The State interface defines a common interface for all concrete states; the states all implement the same interface, so they are interchangeable.
Context
request()
State
handle()
state.handle()
ConcreteStateA
handle()
ConcreteStateB
handle()
Many concrete states are possible.
Whenever the request() is made on the Context it is delegated to the state to handle.
ConcreteStates handle requests from the Context. Each ConcreteState provides its own implementation for a request. In this way, when the Context changes state, its behavior will change as well.