02

Códigos parecidos e o Template Method

Em nossa aplicação de orçamentos, diversos impostos possuem cálculos parecidos. Por exemplo, imagine dois impostos ICPP e IKCV. O imposto ICPP é calculado da seguinte forma: caso o valor do orçamento seja menor que 500,00, deve-se cobrar 5%; caso contrário, 7%.

Seguindo já a nossa abstração `Imposto`, criado no capítulo referente ao padrão Strategy, teríamos a seguinte implementação:

```
class ICPP implements Imposto {
    public double calcula(Orcamento orcamento) {
      if(orcamento.getValor() > 500) {
        return orcamento.getValor() * 0.07;
      } else {
        return orcamento.getValor() * 0.05;
      }
    }
}
```

Já o imposto IKCV, caso o valor do orçamento seja maior que 500,00 e algum item tiver valor superior a 100,00, o imposto a ser cobrado é de 10%; caso contrário 6%.

A implementação seria algo como:

```
class IKCV implements Imposto {
    public double calcula(Orcamento orcamento) {
      if(orcamento.getValor() > 500 && temItemMaiorQue100ReaisNo(orcamento)) {
        return orcamento.getValor() * 0.10;
      } else {
        return orcamento.getValor() * 0.06;
      }
    }

    private boolean temItemMaiorQue100ReaisNo(Orcamento orcamento) {
      for(Item item : orcamento.getItens()) {
        if(item.getValor() > 100) return true;
      }

      return false;
    }
}
```

Veja que os dois impostos, apesar de diferentes, possuem uma estrutura em comum. Ambos checam o orçamento para ver se devem cobrar a taxação máxima e, a partir daí, cobram a máxima ou a mínima.

Poderíamos escrever um algoritmo que generaliza os outros dois, algo como um "molde":

```
class TemplateDeImpostoCondicional implements Imposto {
    public double calcula(Orcamento orcamento) {
```

```
      if(deveUsarMaximaTaxacao(orcamento)) {
        return maximaTaxacao(orcamento);
      } else {
        return minimaTaxacao(orcamento);
      }

    }

    // e os três métodos necessários
}
```

Bastaria agora fazer com que os impostos X e Y possuam suas próprias implementações de `deveUsarMaximaTaxacao()`, `maximaTaxacao()` e `minimaTaxacao()`.

Podemos deixar explícito nesse código que cada um desses métodos são "buracos" e devem ser implementados por classes-filhas. Logo, podemos tornar esses métodos abstratos!

```
public abstract class TemplateDeImpostoCondicional implements Imposto {

    public double calcula(Orcamento orcamento) {

      if(deveUsarMaximaTaxacao(orcamento)) {
        return maximaTaxacao(orcamento);
      } else {
        return minimaTaxacao(orcamento);
      }
    }

    public abstract boolean deveUsarMaximaTaxacao(Orcamento orcamento);
    public abstract double maximaTaxacao(Orcamento orcamento);
    public abstract double minimaTaxacao(Orcamento orcamento);
}
```

Vamos fazer a implementação de Y, por exemplo. A classe `ImpostoY` vai herdar de `TemplateDeImpostoCondicional`, e escrever apenas os métodos abstratos; afinal, o método público que contém o algoritmo já está escrito na classe pai!

```
class ImpostoY extends TemplateDeImpostoCondicional {

  public boolean deveUsarMaximaTaxacao(Orcamento orcamento) {
    return orcamento.getValor() > 500 && temItemMaiorQue100ReaisNo(orcamento);
  }
  public double maximaTaxacao(Orcamento orcamento) {
    return orcamento.getValor() * 0.10;
  }
  public double minimaTaxacao(Orcamento orcamento) {
    return orcamento.getValor() * 0.06;
  }

  private boolean temItemMaiorQue100ReaisNo(Orcamento orcamento) {
    // retorna verdadeiro caso algum item seja maior que 100 reais
  }
}
```

A mesma coisa para X:

```
class ImpostoX extends TemplateDeImpostoCondicional {

  public boolean deveUsarMaximaTaxacao(Orcamento orcamento) {
    return orcamento.getValor() > 500;
  }
  public double maximaTaxacao(Orcamento orcamento) {
    return orcamento.getValor() * 0.07;
  }
  public double minimaTaxacao(Orcamento orcamento) {
    return orcamento.getValor() * 0.05;
  }
}
```

Veja que ambas as classes de impostos só implementam as partes "que faltam" do algoritmo! A classe `TemplateDeImpostoCondicional` possui um método que funciona como um template, ou seja, um molde, para as classes filhas. Daí o nome do padrão de projeto: Template Method.

Veja que o uso do padrão evitou a repetição de código, e ainda facilitou a implementação das diferentes variações do algoritmo.