Universidade Tecnológica Federal do Paraná

Engenharia de Software - Curso de Arquitetura de Software (AS27S)

INSTRUTOR: Prof. Dr. Gustavo Santos

Ibanez Fernandes da Silva Junior, 2033500

CCH - Design Patterns Template

Problema

O problema que estamos abordando é o processamento de diferentes tipos de documentos financeiros, como dinheiro, cheque, PIX e cartões. Cada tipo de documento possui etapas específicas que devem ser executadas em uma determinada ordem, como validação de dados, processamento de pagamento e emissão de comprovante. Além disso, algumas subclasses específicas de cheque e cartão também têm suas próprias etapas personalizadas.

Descrição da Solução

Para resolver esse problema, utilizamos o padrão de projeto Template. A classe abstrata DocumentoFinanceiro define o método de modelo processarPagamento(), que implementa a sequência de etapas comuns a todos os documentos financeiros. Essas etapas incluem a validação de dados, o processamento do pagamento e a emissão do comprovante.

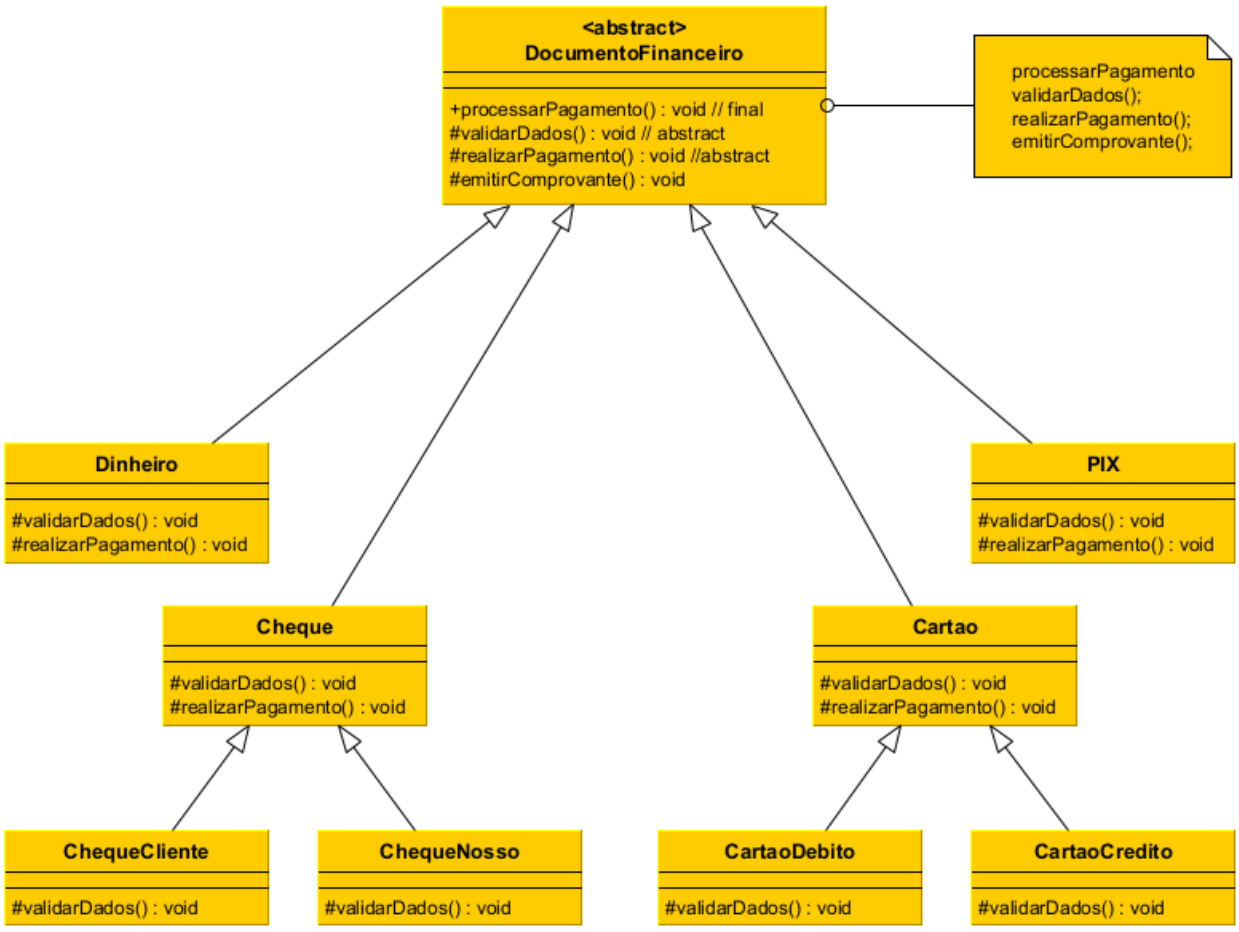
As subclasses concretas, como Dinheiro, Cheque, PIX e Cartao, herdam da classe DocumentoFinanceiro e fornecem implementações específicas para as etapas abstratas. Por exemplo, a classe Dinheiro implementou a validação de dados e o processamento do pagamento em dinheiro. A classe Cheque implementa as etapas para processar pagamentos em cheque, enquanto as subclasses ChequeCliente e ChequeNosso estendem a classe Cheque e adicionam etapas personalizadas de validação de dados.

Da mesma forma, as subclasses PIX, CartaoCredito e CartaoDebito fornecem suas próprias implementações para as etapas abstratas, permitindo que diferentes tipos de documentos financeiros sejam processados de acordo com suas necessidades específicas.

Com esse design, podemos criar instâncias das classes de documentos financeiros e chamar o método processarPagamento(), sabendo que cada tipo de documento seguirá a sequência de etapas correta e executará suas ações específicas.

O padrão de projeto Template nos permite reutilizar a estrutura geral do algoritmo de processamento de documentos financeiros, evitando a duplicação de código e promovendo a flexibilidade ao permitir que diferentes tipos de documentos personalizem seu comportamento dentro das etapas do algoritmo. Isso torna o código mais modular, fácil de entender e de manter.

Visão Geral



Exemplo de Código

Java

```java
abstract class DocumentoFinanceiro {
    public final void processarPagamento() {
        validarDados();
        realizarPagamento();
        emitirComprovante();
    }

    protected abstract void validarDados();

    protected abstract void realizarPagamento();

    protected void emitirComprovante() {
        System.out.println("Comprovante emitido.");
    }
}

class Dinheiro extends DocumentoFinanceiro {
    @Override
    protected void validarDados() {
        System.out.println("Validando dados para pagamento em dinheiro.");
    }

    @Override
    protected void realizarPagamento() {
        System.out.println("Processando pagamento em dinheiro.");
    }
}

class Cheque extends DocumentoFinanceiro {
    @Override
    protected void validarDados() {
        System.out.println("Validando dados para pagamento em cheque.");
    }

    @Override
    protected void realizarPagamento() {
        System.out.println("Processando pagamento em cheque.");
    }
}

class ChequeCliente extends Cheque {
    @Override
    protected void validarDados() {
        System.out.println("Validando dados para pagamento em cheque do cliente.");
    }
}

class ChequeNosso extends Cheque {
    @Override
    protected void validarDados() {
        System.out.println("Validando dados para pagamento em cheque nosso.");
    }
}
```

```
class PIX extends DocumentoFinanceiro {
    @Override
    protected void validarDados() {
        System.out.println("Validando dados para pagamento via PIX.");
    }

    @Override
    protected void realizarPagamento() {
        System.out.println("Processando pagamento via PIX.");
    }
}

class Cartao extends DocumentoFinanceiro {
    @Override
    protected void validarDados() {
        System.out.println("Validando dados para pagamento com cartão.");
    }

    @Override
    protected void realizarPagamento() {
        System.out.println("Processando pagamento com cartão.");
    }
}

class CartaoCredito extends Cartao {
    @Override
    protected void validarDados() {
        System.out.println("Validando dados para pagamento com cartão de
crédito.");
    }
}

class CartaoDebito extends Cartao {
    @Override
    protected void validarDados() {
        System.out.println("Validando dados para pagamento com cartão de débito.");
    }
}

public class Main {
    public static void main(String[] args) {
        DocumentoFinanceiro dinheiro = new Dinheiro();
        dinheiro.processarPagamento();
        System.out.println();

        DocumentoFinanceiro chequeCliente = new ChequeCliente();
        chequeCliente.processarPagamento();
        System.out.println();

        DocumentoFinanceiro pix = new PIX();
        pix.processarPagamento();
        System.out.println();

        DocumentoFinanceiro cartaoCredito = new CartaoCredito();
        cartaoCredito.processarPagamento();
    }
}
```

A classe abstrata DocumentoFinanceiro define o método de modelo processarPagamento(), que segue uma sequência de etapas fixas: validarDados(), realizarPagamento() e emitirComprovante(). As subclasses herdam desta classe e fornecem implementações específicas para as etapas abstratas.

As subclasses Dinheiro, Cheque, PIX e Cartao implementam a validação de dados e o processamento de pagamento de acordo com seus respectivos tipos de documentos financeiros.

As subclasses ChequeCliente e ChequeNosso estendem a classe Cheque e fornecem implementações específicas para a validação de dados, adicionando mais detalhes ao processamento de pagamento em cheque.

As subclasses CartaoCredito e CartaoDebito estendem a classe Cartao e também fornecem implementações específicas para a validação de dados.

No método main(), criamos instâncias de cada classe de documento financeiro e chamamos o método processarPagamento(), que segue a sequência de etapas definida na classe base DocumentoFinanceiro.

Dessa forma, o padrão de projeto Template permite que diferentes tipos de documentos financeiros sigam a mesma sequência de etapas, mas com comportamentos específicos implementados por cada classe.

Consequências

- Vantagens
    - Você pode deixar clientes sobrescrever apenas certas partes de um algoritmo grande, tornando-os menos afetados por mudanças que acontecem por outras partes do algoritmo.
    - Você pode elevar o código duplicado para uma superclasse.
- Desvantagens
    - Alguns clientes podem ser limitados ao fornecer o esqueleto de um algoritmo.
    - Você pode violar o *princípio de substituição de Liskov* ao suprimir uma etapa padrão de implementação através da subclasse.
    - Implementações do padrão Template Method tendem a ser mais difíceis de se manter quanto mais etapas eles tiverem.

Repositório: ***https://github.com/ibanez-junior-utfpr/as27s-2023-1***

Pasta: CCH-3