Relatório sobre a Implementação do Sistema Bancário

Gabriel da Silva Reboli

CC4M

1 Introdução

Este relatório descreve a implementação de um sistema bancário utilizando o padrão de projeto *Factory Method*. O sistema gerencia contas correntes, poupanças e contas salário, bem como operações realizadas por clientes, agências e o banco. Serão discutidas as estruturas de classes utilizadas, como o padrão foi aplicado e sugestões de melhorias.

2 Aplicação do Padrão *Factory Method*

O padrão *Factory Method* foi implementado para facilitar a criação de diferentes tipos de contas bancárias. Cada tipo de conta (`CurrentAccount`, `SavingsAccount`, `SalaryAccount`) é gerado por uma fábrica específica, implementando a interface `IAccountFactory`. Esse padrão oferece flexibilidade ao sistema, permitindo que novas contas possam ser adicionadas sem modificar o código existente.

2.1 Detalhes da Implementação

A interface `IAccountFactory` define um método `CreateAccount`, que é implementado por cada fábrica de contas. As fábricas `CurrentAccountFactory`, `SavingsAccountFactory` e `SalaryAccountFactory` encapsulam a lógica de criação de cada tipo de conta, fornecendo a separação clara entre a lógica de criação e o uso da conta no sistema.

O método `CreateAccount` recebe parâmetros variáveis para cada tipo de conta, como *service fee* e *limit* para contas correntes, e *rate* para contas poupanças. Isso facilita a criação de instâncias de contas com os parâmetros corretos sem modificar o código principal do programa (`Program.cs`).

2.2 Vantagens do *Factory Method*

- **Extensibilidade**: Novos tipos de contas podem ser adicionados facilmente ao sistema, criando novas fábricas que implementem a interface `IAccountFactory`, sem necessidade de alterar outras partes do código.
- **Manutenção**: O uso do padrão centraliza a lógica de criação das contas, permitindo que o código cliente apenas dependa das fábricas, o que facilita a manutenção e atualização do sistema.

3 Estrutura do Código

O código está estruturado em classes que representam entidades do sistema bancário:

- **Bank**: A classe `Bank` gerencia uma coleção de agências (`Branch`) e clientes (`Client`). Ela permite adicionar, remover e buscar agências e clientes pelo CPF.
- **Branch**: A classe `Branch` gerencia contas bancárias, permitindo operações de adição, remoção e busca de contas.
- **Client**: Representa um cliente do banco com informações pessoais (CPF, nome, endereço, telefone e email).

- **Account (abstrata)**: Define a interface para operações comuns de contas bancárias, como sacar, depositar, transferir e obter o saldo. As classes `CurrentAccount`, `SavingsAccount` e `SalaryAccount` herdam dessa classe e implementam regras de negócio específicas.
- **Fábricas**: As fábricas `CurrentAccountFactory`, `SavingsAccountFactory` e `SalaryAccountFactory` implementam a lógica para criação de diferentes tipos de contas, seguindo o padrão *Factory Method*.

4 Possíveis Melhorias

Embora o sistema esteja bem estruturado e siga o padrão de projeto *Factory Method*, algumas melhorias podem ser implementadas:

- **Validação de Dados**: Incluir mecanismos para validar corretamente os parâmetros de entrada, como número de conta e saldo, antes de criar as contas ou realizar operações.
- **Tratamento de Exceções**: Melhorar o tratamento de exceções, fornecendo mensagens de erro mais detalhadas e úteis ao usuário final.
- **Testes Automatizados**: Implementar testes unitários para garantir que as regras de negócio, como o limite de saques em contas poupança e a restrição de Pix para contas salário, estão sendo respeitadas.
- **Persistência de Dados**: O sistema pode ser aprimorado com um banco de dados para armazenar informações sobre contas, clientes e operações, permitindo persistência entre execuções.

5 Conclusão

O uso do padrão *Factory Method* neste sistema bancário traz flexibilidade e facilita a manutenção e a extensão do código. A estrutura atual atende aos requisitos de criação de contas e operações básicas, mas melhorias podem ser feitas, especialmente em termos de validação e tratamento de exceções, além da implementação de persistência de dados.