Sistema Bancário - Transformação para POO
Alunos: Ítalo Fernandes Gasparotto e Pedro Henrique Fochezzatto Savi

1. INTRODUÇÃO

Este relatório documenta a transformação de um sistema bancário desenvolvido em paradigma procedural para uma arquitetura orientada a objetos. O projeto original consistia em funções que manipulavam dicionários, enquanto a nova versão aplica os princípios de POO: Encapsulamento, Herança, Polimorfismo e Abstração.

A migração representa uma reformulação completa da arquitetura, proporcionando maior organização, reusabilidade de código e manutenibilidade. O sistema agora segue boas práticas de desenvolvimento e está preparado para expansões futuras.

2. ARQUITETURA DO SISTEMA

O sistema foi reorganizado em uma estrutura modular com separação clara de responsabilidades. A camada de entidades gerencia as classes de domínio do negócio bancário, incluindo a hierarquia de contas, clientes e transações. A camada de serviços contém a lógica de negócio complexa, orquestrando operações e gerenciando autenticação. A camada de utilitários fornece funções auxiliares para validações e formatações.

Foram desenvolvidas classes especializadas para cada entidade do domínio bancário. A classe abstrata Conta define a interface comum para todos os tipos de conta, estabelecendo métodos obrigatórios e comportamento base. As classes ContaCorrente, ContaPoupanca e ContaEmpresarial implementam especializações com comportamentos e regras específicas.

3. APLICAÇÃO DOS CONCEITOS DE POO

A abstração foi aplicada através de classes abstratas que definem contratos claros para as implementações concretas. A classe base Conta estabelece a interface comum que todas as contas devem seguir, com métodos abstratos que forçam a implementação específica em cada subtipo. Esta abordagem garante que novas funcionalidades possam ser adicionadas seguindo contratos predefinidos.

O encapsulamento foi rigorosamente aplicado para proteger o estado interno dos objetos e controlar o acesso aos dados sensíveis. Atributos como saldo, senhas e histórico de transações foram definidos como protegidos, acessíveis apenas através de métodos públicos específicos. Properties foram implementadas para fornecer acesso controlado a atributos, permitindo validações quando necessário.

A herança foi utilizada para criar uma hierarquia natural de tipos de conta, promovendo o reuso de código e a especialização de comportamentos. Cada tipo de conta especializada herda da classe base e adiciona ou modifica comportamentos específicos. Esta hierarquia permite que novos tipos de conta sejam adicionados com mínimo esforço.

O polimorfismo permite que objetos de diferentes tipos sejam tratados de forma uniforme através de interfaces comuns. O método de cálculo de taxas, por exemplo, possui implementações específicas em cada tipo de conta, mas é invocado através da mesma interface. O sistema de transações utiliza polimorfismo para executar operações financeiras de tipos diferentes através de uma interface comum.

4. MELHORIAS IMPLEMENTADAS

Diversas melhorias significativas foram implementadas na versão POO. O sistema de transações agora registra detalhadamente todas as operações, incluindo data e hora, tipo de operação, valores, taxas e saldos resultantes. Validações robustas foram centralizadas, incluindo verificação de CPF com algoritmo oficial e validação de formato de email.

A segurança foi fortalecida através do encapsulamento de senhas, controle de acesso e autenticação centralizada. A gestão de estado das sessões de usuário foi melhorada, proporcionando experiência mais consistente e segura. O sistema agora trata adequadamente casos especiais como transferências para a própria conta e saldos insuficientes.

A arquitetura incorpora padrões de projeto consagrados. O Service Layer Pattern separa claramente a lógica de domínio da lógica de aplicação. A arquitetura segue princípios SOLID, particularmente o Single Responsibility Principle, onde cada classe tem uma única responsabilidade bem definida.

5. COMPARAÇÃO ENTRE AS ABORDAGENS

A transformação resultou em melhorias significativas em múltiplas dimensões do sistema. Na organização, evoluiu-se de funções soltas para classes organizadas por responsabilidade. Na manutenibilidade, saiu-se de uma estrutura complexa com alto acoplamento para uma arquitetura com baixo acoplamento e fácil manutenção.

Quanto à extensibilidade, a abordagem procedural exigia modificação direta do código, enquanto a POO permite extensão através de herança e composição. No reuso, substituiu-se a prática de copiar e colar código pela utilização de herança e componentes reutilizáveis.

A testabilidade, antes difícil devido a dependências embutidas, tornou-se facilitada pelo isolamento das classes. A legibilidade do código melhorou significativamente através de nomes significativos e estrutura auto-documentada.

6. USO DE FERRAMENTAS E METODOLOGIAS

O desenvolvimento utilizou Python como linguagem base. A Inteligência Artificial foi utilizada estrategicamente para revisão de código quanto à conformidade com princípios SOLID, sugestões de padrões de projeto apropriados e organização da estrutura de arquivos. Foram seguidas metodologias ágeis incluindo refactoring contínuo e boas práticas de desenvolvimento. O controle de versão foi gerenciado adequadamente para acompanhar a evolução do projeto.

7. CONCLUSÃO

A transformação para POO resultou em um sistema significativamente mais robusto, com tratamento consistente de erros e validações abrangentes. A escalabilidade foi amplamente melhorada, com facilidade para adicionar novas funcionalidades e tipos de conta. A manutenibilidade tornou-se uma característica forte, com código organizado e compreensível.

Os benefícios incluem redução de custos de manutenção, agilidade na implementação de novas features e maior confiabilidade operacional. O sistema agora está adequadamente preparado para expansões como interface gráfica, persistência em banco de dados e integrações com sistemas externos.

A arquitetura orientada a objetos provou ser fundamental para criar um sistema bancário profissional, escalável e de alta qualidade, demonstrando na prática a importância dos princípios de POO no desenvolvimento de software corporativo.

Preparado por: Ítalo Fernandes Gaparotto
Data: Novembro 2025