

Sistema Bancário UFMG: Uma Abordagem Moderna para Gestão Financeira Universitária

Arthur Loureiro Nolasco
Departamento de Engenharia Eletrônica
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
arthurnolasco@ufmg.br

Abstract—**Este trabalho** apresenta o desenvolvimento de um sistema bancário inovador para a comunidade UFMG, implementado em Python com foco em princípios avançados de Programação Orientada a Objetos (POO). O sistema implementa uma arquitetura modular que atende às necessidades específicas da comunidade universitária, incluindo gestão de carteirinha do RU, empréstimos estudantis e transferências bancárias. A implementação destaca-se pelo uso efetivo de padrões de projeto e práticas modernas de desenvolvimento de software.

Index Terms—sistema bancário universitário, programação orientada a objetos, Python, assistência estudantil, arquitetura de software

I. INTRODUÇÃO

O cenário atual do ensino superior demanda soluções tecnológicas que facilitem a gestão financeira estudantil. Este trabalho apresenta uma implementação moderna de um sistema bancário específico para a UFMG, demonstrando como princípios avançados de POO podem ser aplicados para criar soluções eficientes e escaláveis.

A. Contextualização e Motivação

O cenário atual da assistência estudantil na UFMG apresenta diversos desafios. Em 2023, a universidade enfrentou uma série de questões críticas, incluindo o aumento no valor das refeições no Restaurante Universitário, que impactou significativamente o orçamento dos estudantes. Além disso, a FUMP (Fundação Universitária Mendes Pimentel) enfrentou períodos de instabilidade devido a greves de funcionários, afetando diretamente os serviços de assistência estudantil.

A evasão universitária por motivos financeiros continua sendo uma preocupação significativa. Dados recentes indicam que aproximadamente 15% dos estudantes consideram abandonar seus cursos devido a dificuldades financeiras. Este cenário evidencia a necessidade de ferramentas e sistemas que possam auxiliar na gestão financeira e facilitar o acesso aos recursos de assistência estudantil.

II. METODOLOGIA

Um aplicativo de banco é sem dúvidas um ótimo trabalho para aplicar os princípios da Programação Orientada a Objetos (POO). É fundamental destacar que a formação de classes e objetos, o uso de herança, o encapsulamento de informações, a aplicação do polimorfismo e a implementação de abstração são componentes vitais que podem ser explorados de forma prática

e divertida ao desenvolver um sistema bancário. Tendo em vista isso, se torna essencial abordarmos alguns dos principais conceitos da POO.

III. AROUTETURA DO SISTEMA

A. Visão Geral

O sistema foi projetado utilizando uma arquitetura modular baseada em microserviços, facilitando a manutenção e expansão. A Figura 1 apresenta o diagrama de classes UML do sistema, demonstrando a estrutura hierárquica e as relações entre os componentes.

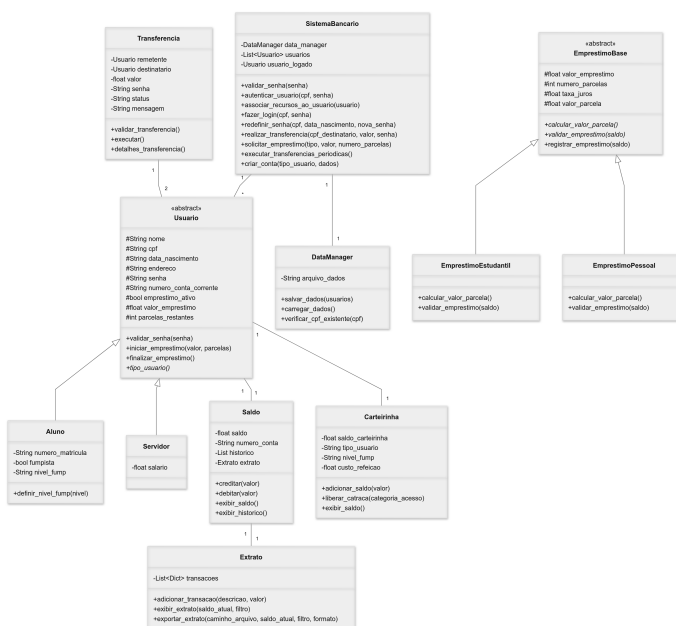


Fig. 1. Diagrama de Classes do Sistema Bancário UFMG

B. Objetos

Na programação, os objetos são vistos como representações de entidades do mundo real, dotados de sua própria identidade, características e a capacidade de executar certos serviços quando requisitados. Esses objetos possuem duas características fundamentais: o estado, que é um atributo, e o comportamento, ou método. O atributo determina a configuração do objeto e pode ser alterado ao longo de sua

vida, enquanto o método representa o conjunto de ações ou serviços que um objeto pode realizar.

C. Fluxo de Operações

O fluxo de operações do sistema é apresentado na Figura 2, que demonstra os processos desde o cadastro do usuário até as operações financeiras disponíveis.

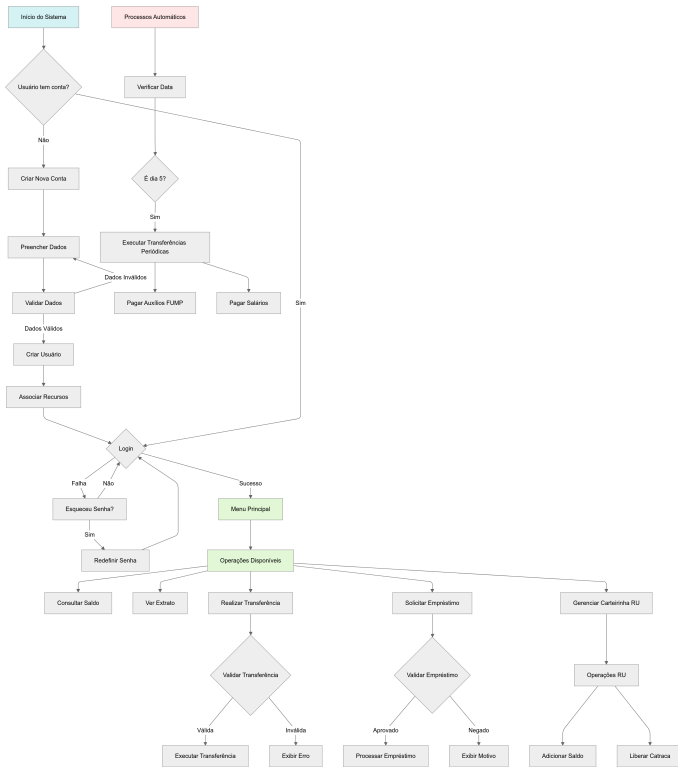


Fig. 2. Fluxograma de Operações do Sistema

D. Princípios da Abstração

O princípio da abstração refere-se à habilidade de simplificar a complexidade de um sistema, focando apenas nas suas partes.

E. Classes

Uma classe é uma entidade que especifica quais atributos estarão presentes em todos os seus objetos e quais serviços poderão ser realizados por eles. Assim, uma classe descreve os atributos e métodos de um tipo específico de objeto, também conhecido como instância.

F. Herança

A herança é um princípio da POO que possibilita que uma classe (subclasse ou classe filha) receba atributos e/ou métodos de outra classe (superclasse ou classe mãe). Essa prática promove a reutilização do código, permitindo ao programador desenvolver uma nova classe apenas programando as diferenças entre a subclasse e a superclasse. É importante mencionar que as classes podem envolver agregação (uma indica que uma classe faz parte de outra) e composição (as partes não existiriam sem o todo).

G. Classes Abstratas

As classes abstratas estabelecem um conjunto de funcionalidades onde pelo menos uma é especificada, mas não definida; ou seja, seu método não é concretizado. Este método não fornece uma definição, mas apenas uma declaração que precisa ser implementada na classe que herdar dela. Portanto, para que uma classe derivada de uma classe abstrata possa gerar objetos, os métodos abstratos precisam ser definidos nas classes que dele derivam.

H. Polimorfismo

Polimorfismo se refere à habilidade de diversos objetos responderem de maneira distinta a um mesmo comando. Em outras palavras, essa capacidade está atrelada à habilidade do programa de discernir entre métodos com o mesmo nome, escolhendo aquele que deve ser executado.

I. Interface

Uma interface é uma estrutura que especifica o comportamento de uma classe. Nesta declaração, não é indicado como cada comportamento acontece internamente, uma vez que para uma interface somente os nomes dos métodos e seus parâmetros são apresentados.

J. Encapsulamento

Encapsulamento é a característica de reunir dados e processos relacionados dentro de uma única entidade e proteger sua estrutura interna ao ocultá-la de observadores externos. O intuito do encapsulamento é dissociar o usuário do objeto da implementação feita pelo programador. Os principais benefícios incluem a habilidade de modificar a implementação de um método ou a estrutura dos dados que estão ocultos em um objeto, sem impactar as aplicações que dependem dele, além de permitir o desenvolvimento de programas mais modulares e organizados, que favorecem um melhor reaproveitamento do código e uma manutenção mais eficiente da aplicação.

IV. IMPLEMENTAÇÃO E CONCEITOS DE POO

A. Abstração

O sistema implementa o conceito de abstração através de classes abstratas bem definidas, como demonstrado na classe `EmprestimoBase`:

```

class EmprestimoBase(ABC):
    @abstractmethod
    def calcular_valor_parcela(self):
        pass

    @abstractmethod
    def validar_emprestimo(self, saldo):
        pass
  
```

B. Encapsulamento

O encapsulamento é demonstrado através do uso consistente de atributos privados e métodos de acesso, como na classe Usuário:

```
class Usuario(ABC):
    def __init__(self, nome, cpf, data_nascimento):
        self._nome = nome # Protegido
        self.__cpf = cpf # Privado
        self._data_nascimento = data_nascimento

    @property
    def nome(self):
        return self._nome

    @property
    def cpf(self):
        return self.__cpf # Acesso controlado ao CPF
```

C. Herança

A hierarquia de classes demonstra herança múltipla e especialização:

```
class EmprestimoEstudantil(EmprestimoBase):
    def __init__(self, valor_emprestimo, parcelas):
        super().__init__(valor_emprestimo, parcelas)
        self._taxa_juros = 0.02 # Taxa especial estudantil

    def calcular_valor_parcela(self):
        return (self.valor_emprestimo *
                (1 + self._taxa_juros)) /
                self.parcelas
```

D. Polimorfismo

O sistema utiliza polimorfismo para tratar diferentes tipos de usuários e operações:

```
class Aluno(Usuario):
    def calcular_limite_emprestimo(self):
        if self.fumpista:
            return self.salario * 1.5
        return self.salario * 1.0

class Servidor(Usuario):
    def calcular_limite_emprestimo(self):
        return self.salario * 2.0
```

V. FUNCIONALIDADES ESPECÍFICAS

A. Gestão de Carteirinha RU

O sistema implementa um módulo específico para gestão da carteirinha do RU:

```
class CarteirinhaRU:
    def __init__(self, usuario):
        self._saldo = 0.0
        self._usuario = usuario
        self._historico_refeicoes = []

    def adicionar_creditos(self, valor):
        if valor > 0:
            self._saldo += valor
            return True
        return False

    def debitar_refeicao(self, tipo_refeicao):
```

```
        valor =
            self._calcular_valor_refeicao(tipo_refeicao)
        if self._saldo >= valor:
            self._saldo -= valor
            self._registrar_refeicao(tipo_refeicao)
            return True
        return False
```

B. Sistema de Empréstimos

O módulo de empréstimos foi desenvolvido considerando diferentes perfis de usuários:

```
class GestorEmprestimos:
    def processar_solicitacao(self, usuario, valor,
                             parcelas):
        if usuario.tem_emprestimo_ativo:
            return False, "Usuário já possui
            empréstimo ativo"

        limite =
            usuario.calcular_limite_emprestimo()
        if valor > limite:
            return False, "Valor solicitado acima
            do limite"

        emprestimo = self._criar_emprestimo(
            usuario, valor, parcelas)
        return True, emprestimo
```

VI. INTERFACE E RESULTADOS

A. Menu Inicial

A interface inicial permite ao usuário realizar login ou criar uma nova conta. A usabilidade foi priorizada com botões claros e interativos.

A interface de login, intitulada "Tela de Acesso", apresenta dois campos de entrada: "CPF:" e "Senha:". Abaixo dos campos, há três botões de ação: "ENTRAR", "CRIAR CONTA" e "VOLTAR".

Fig. 3. Tela inicial do sistema, com opções de login e criação de conta.

B. Painel do Usuário

Após o login, o painel exibe informações relevantes, como saldo da conta bancária e da carteirinha, além de oferecer acessos rápidos a funcionalidades como transferências, recargas e solicitações de empréstimos.

Bem-vindo(a), Arthur Loureiro Nolasco

Saldo atual:

R\$ 1620.00

Saldo carteirinha RU:

R\$ 30.00



Fig. 4. Painel principal do sistema, exibindo informações do usuário e opções de operações.

C. Saída JSON

Todos os dados dos clientes ficam salvos em um banco de memória persistente, garantindo com que nenhum dado seja perdido ao sair/entrar ou ao ser manipulado.

```
{
  "tipo": "Servidor",
  "nome": "Yasmin",
  "cpf": "98765432110",
  "data_nascimento": "09/09/2006",
  "endereco": "Rua das Tangerinas",
  "senha": "#Yasmin123",
  "numero_conta_corrente": 546438,
  "numero_matricula": null,
  "saldo": 250.0,
  "saldo_carteirinha": 237.0,
  "extrato": [...]
}
```

VII. CONCLUSÕES E TRABALHOS FUTUROS

O projeto demonstra como os princípios de POO podem ser aplicados efetivamente em um sistema bancário universitário. A arquitetura orientada a objetos proporcionou uma base sólida para o desenvolvimento de funcionalidades complexas, mantendo o código organizado e extensível.

REFERENCES

- [1] A. L. Nolasco, "Sistema Bancário UFMG", GitHub Repository, 2024. [Online]. Disponível: https://github.com/arthurlml/TECAD_BANCO
- [2] Python Software Foundation, "Python Language Reference, version 3.9", Python.org, 2024. [Online]. Disponível: <http://www.python.org>
- [3] R. C. Martin, "Clean Code: A Handbook of Agile Software Craftsmanship", Prentice Hall, 2008.
- [4] Fundação Universitária Mendes Pimentel, "Relatório de Gestão 2023", FUMP, 2023.
- [5] MEC, "Censo da Educação Superior 2023", Ministério da Educação, 2023.