SMARTBANK: RELATO DE EXPERIÊNCIA SOBRE O DESENVOLVIMENTO DE UM SISTEMA DE UM BANCO DIGITAL

Gabriel Targa[1]
Hugo Anísio [2]
Luiz Carlos S. Andrade [3]
Thiago Henrique Gomes[4]

RESUMO

Este artigo relata a experiência de desenvolvimento do SmartBank, um banco digital inovador. O estudo teve como objetivo proporcionar aos alunos uma oportunidade de aprendizado sobre práticas ágeis de desenvolvimento de software. A metodologia adotada baseou-se no Paradigma de Orientação a Objetos, com foco em uma abordagem modular e flexível. Reuniões de discussão foram realizadas para a definição dos requisitos e o planejamento do projeto. Diagramas de caso de uso e diagramas de classes foram utilizados para a modelagem do sistema. O código foi desenvolvido a partir da abstração dos diagramas de classes e das análises de requisitos. Além disso, práticas de versionamento com o Git foram empregadas para controle de versões e colaboração entre os desenvolvedores. O livro "UML - Uma Abordagem Prática" de Gilleanes T. A. Guedes foi utilizado como referência teórica para a modelagem do sistema. O uso de técnicas de inteligência artificial foi explorado para auxiliar na produção de código e no desenvolvimento de modelos de classes, visando otimizar tempo e recursos. Os resultados obtidos foram positivos, evidenciando a formação de profissionais capazes de atuar em equipes multidisciplinares e se adaptar às demandas do mercado de trabalho. Este projeto demonstrou a viabilidade e eficácia da aplicação de práticas ágeis no contexto de um banco digital, proporcionando uma experiência enriquecedora para os participantes.

Palavras-chaves: BANCO DIGITAL; MODELAGEM DE SISTEMAS; PARADIGMA DE ORIENTAÇÃO A OBJETOS.

[1] Graduando do Curso de Sistemas para Internet E-mail: 2022210220008@iesp.edu.br
[2] Graduando do Curso de Sistemas para Internet E-mail: hugoanisio2@gmail.com
[3] Graduando do Curso de Sistemas para Internet E-mail: luizcarlossantana12345@gmail.com
[4] Graduando do Curso de Sistemas para Internet E-mail: thiagohgcoutinho@gmail.com

ABSTRACT

This article reports on the development experience of SmartBank, an innovative digital bank. The study aimed to provide students with an opportunity to learn about agile software development practices. The methodology adopted was based on the Object-Oriented Paradigm, focusing on a modular and flexible approach. Discussion meetings were held to define requirements and plan the project. Use case diagrams and class diagrams were used for system modeling. The code was developed based on the abstraction of class diagrams and requirements analysis. Additionally, version control practices with Git were employed for versioning and collaboration among developers. The book "UML - Uma Abordagem Prática" by Gilleanes T. A. Guedes was used as a theoretical reference for system modeling. Techniques of artificial intelligence were explored to assist in code production and the development of class models, aiming to optimize time and resources. The obtained results were positive, demonstrating the formation of professionals capable of working in multidisciplinary teams and adapting to market demands. This project showcased the feasibility and effectiveness of applying agile practices in the context of a digital bank, providing an enriching experience for participants.

Keywords: DIGITAL BANK; SYSTEM MODELING; OBJECT-ORIENTED PARADIGM.

1 INTRODUÇÃO

No atual cenário de constante evolução tecnológica, surgem conceitos inovadores que impulsionam o desenvolvimento de programas e sistemas. Um desses conceitos é o Paradigma de Orientação a Objetos, introduzido pelo matemático e biólogo Alan Kay. Essa abordagem revolucionou a forma como os programas são criados, permitindo o aproveitamento dos conceitos por trás dela para aprimorar a produção de códigos e sistemas.

Dentro desse contexto, baseado na Programação Orientada a Objetos (POO), é possível criar uma variedade de funcionalidades utilizando classes, que possuem atributos e métodos. Inspirados por essa abordagem, propomos a criação de um sistema similar ao utilizado em bancos, utilizando a linguagem de programação Java, Diagramas de Classes e Diagramas de Caso de Uso. O objetivo é proporcionar aos estudantes uma experiência prática e familiar aos conhecimentos adquiridos em sala de aula.

A criação desse sistema é fundamentada em uma pesquisa realizada no mercado financeiro atual, com foco nas grandes fintechs e corporações financeiras do Brasil. Essas empresas desempenham um papel crucial na área tecnológica, sendo reconhecidas pela sua importância na economia e sociedade. Dessa forma, justifica-se o enfoque do projeto, que busca incorporar características semelhantes às dessas empresas, reconhecendo o impacto positivo que elas têm no setor.

Além disso, a criação de um sistema bancário proporciona aos estudantes a oportunidade de aprender sobre práticas ágeis de desenvolvimento de software, como o Scrum. O Scrum é uma metodologia de gerenciamento de projetos que utiliza ciclos iterativos e incrementais, permitindo o desenvolvimento colaborativo de soluções e a rápida adaptação às mudanças. Essa abordagem contribui para a formação de profissionais capazes de atuar em equipes multidisciplinares e se adaptar às constantes transformações do mundo tecnológico.

Neste trabalho de curso, exploraremos os conceitos do Paradigma de Orientação a Objetos, a criação de um sistema bancário e as práticas ágeis de desenvolvimento de software. Buscaremos aplicar esses conhecimentos em um projeto prático, visando ampliar nossa compreensão e preparar-nos para os desafios do mercado de trabalho em constante evolução.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, serão apresentadas as tecnologias, conceitos e referência bibliográfica utilizados neste projeto.

1. Tecnologias

1.1 Linguagem de Programação

A linguagem de programação utilizada neste projeto foi Java (versão 17). Java é uma linguagem de programação orientada a objetos amplamente utilizada na indústria de desenvolvimento de software. Ela é conhecida por sua portabilidade, segurança e robustez. Para obter mais informações sobre Java.

1.2 Ambiente de Desenvolvimento Integrado (IDE)

A IDE utilizada neste projeto foi o IntelliJ IDEA. O IntelliJ IDEA é uma poderosa e popular IDE para desenvolvimento Java. Ele oferece recursos avançados de edição, depuração e refatoração de código, além de integração com várias ferramentas e frameworks.

1.3 Versionamento com Git

O Git foi utilizado para o versionamento do código-fonte neste projeto. O Git é um sistema de controle de versão distribuído amplamente utilizado, que permite o rastreamento das alterações feitas no código ao longo do tempo. Ele oferece recursos como ramificação (branching), mesclagem (merging) e histórico de alterações. Para obter mais informações sobre o Git.

2. Conceitos

2.1 Encapsulamento

O encapsulamento é um conceito fundamental da programação orientada a objetos. Ele envolve o agrupamento de dados e métodos relacionados em uma unidade coesa, conhecida como classe. O encapsulamento permite controlar o acesso aos dados, protegendo-os contra alterações indevidas e garantindo a consistência do sistema.

2.2 Composição

A composição é um princípio de design que permite construir objetos complexos combinando outros objetos menores. Ela é baseada na ideia de que um objeto pode ser composto por outros objetos como partes constituintes. A composição promove a reutilização de código, modularidade e flexibilidade no design de sistemas.

2.3 Herança

A herança é um conceito que permite criar novas classes baseadas em classes existentes. Uma classe derivada (filha) herda os atributos e métodos da classe base (pai), além de adicionar ou modificar seu comportamento. A herança promove a reutilização de código e estabelece relações hierárquicas entre classes.

2.4 Polimorfismo

O polimorfismo é um conceito que permite que objetos de diferentes classes sejam tratados de maneira uniforme. Ele permite que um mesmo método tenha diferentes comportamentos dependendo do tipo do objeto em tempo de execução. O polimorfismo melhora a flexibilidade e a extensibilidade do código.

2.5 Classes e Métodos Abstratos

Classes e métodos abstratos são componentes fundamentais na programação orientada a objetos. Uma classe abstrata serve como base para outras classes, definindo a estrutura e o comportamento básico que as classes filhas devem implementar. Isso promove a reutilização de código e a organização hierárquica das classes. Os métodos abstratos, por sua vez, são declarações de métodos sem implementação, estabelecendo um contrato que as classes filhas devem cumprir ao implementar esses métodos. Esses conceitos proporcionam maior flexibilidade e modularidade no desenvolvimento de sistemas.

Referência bibliográfica:

- UML - Uma Abordagem Prática, Gilleanes T. A. Guedes, Editora Novatec.

3 METODOLOGIA

Neste tópico, serão apresentados os procedimentos adotados para a realização deste projeto, incluindo a seleção de tecnologias, reuniões de discussão, utilização de guia do professor, criação de diagramas de caso de uso e diagramas de classes, codificação baseada na abstração dos diagramas, análise de requisitos, pesquisa no livro "UML - Uma Abordagem Prática", além do uso de sistemas de inteligência artificial para auxiliar no desenvolvimento de código e modelos de classes.

3.1 Seleção de Tecnologias

Descrever as etapas de reuniões realizadas para decidir quais tecnologias seriam utilizadas no projeto, levando em consideração fatores como requisitos do projeto, compatibilidade, recursos disponíveis e experiência da equipe.

3.2 Utilização do Guia do Professor

Explicar como o guia fornecido pelo professor do projeto foi utilizado como referência para orientar as etapas de desenvolvimento, incluindo as melhores práticas e diretrizes a serem seguidas.

3.3 Criação de Diagramas de Caso de Uso e Diagramas de Classes

Detalhar o processo de criação dos diagramas de caso de uso e diagramas de classes, que foram utilizados para modelar e representar visualmente os requisitos e a estrutura do sistema.

3.4 Codificação baseada na Abstração dos Diagramas

Descrever como os diagramas de classes foram utilizados como base para a codificação do sistema, em que a abstração dos diagramas serviu como guia para a criação das classes e métodos.

3.5 Análise de Requisitos

Explicar como a análise de requisitos foi conduzida, incluindo a identificação e documentação dos requisitos funcionais e não funcionais do sistema.

3.6 Pesquisa no Livro "UML - Uma Abordagem Prática"

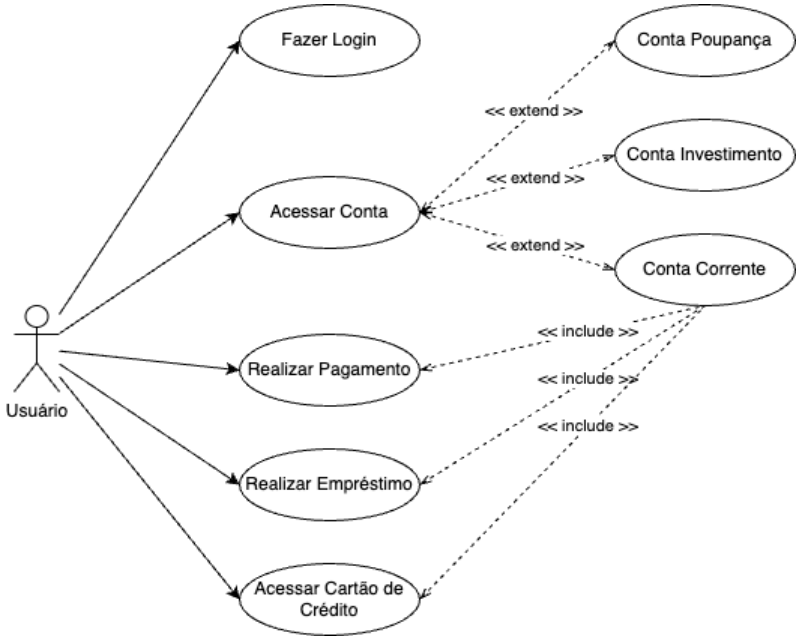
Descrever a pesquisa realizada no livro "UML - Uma Abordagem Prática" de Gilleanes T. A. Guedes, que serviu como fonte de referência para a compreensão e aplicação dos conceitos de UML (Unified Modeling Language) no desenvolvimento do projeto.

3.7 Utilização de Sistemas de Inteligência Artificial

Mencionar o uso de sistemas de inteligência artificial para auxiliar na produção de código e no desenvolvimento de modelos de classes, visando otimizar o tempo e os recursos envolvidos no projeto.

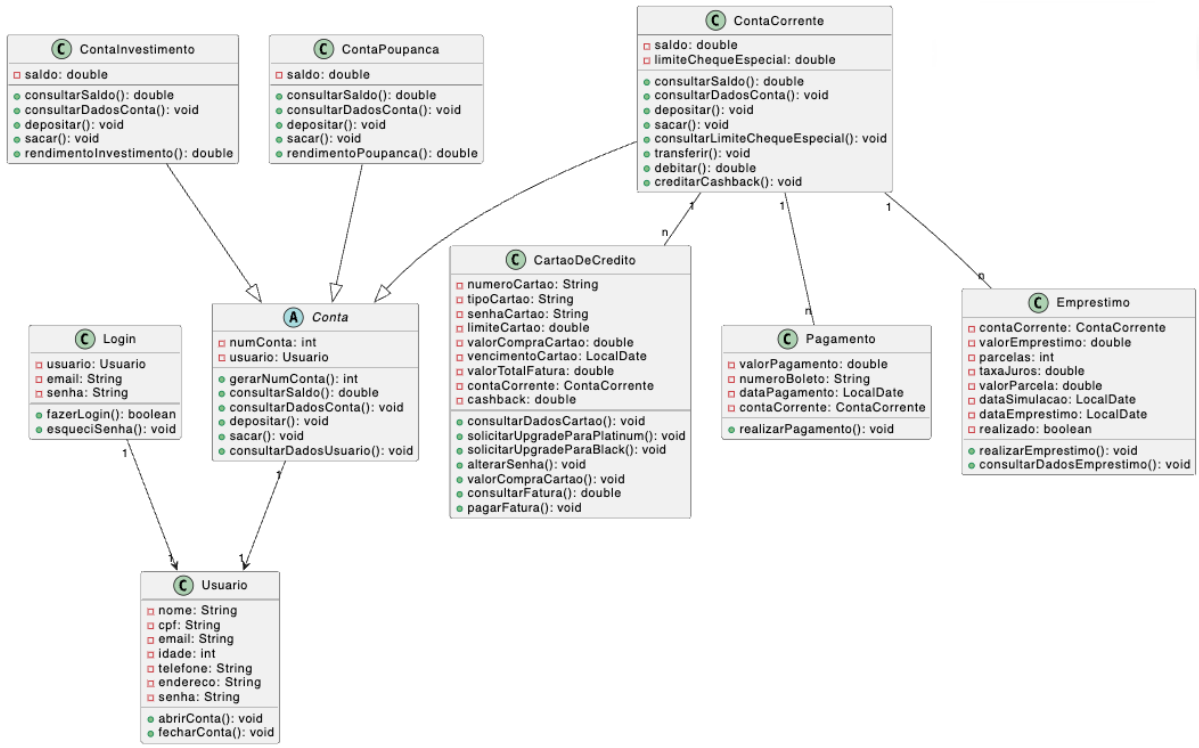
4 RESULTADO E DISCUSSÃO

4.1 Diagrama de Caso de Uso



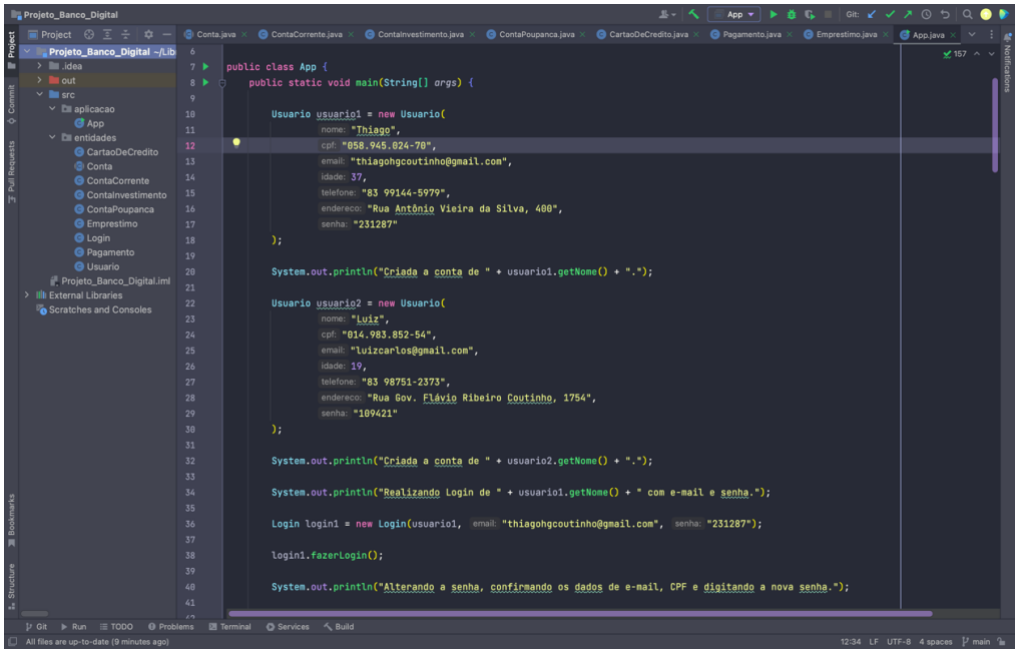
O diagrama de caso de uso apresentado descreve a interação entre ator e os principais casos de uso de um sistema. Ele ilustra as funcionalidades disponíveis para os usuários, representados pelo ator, e como eles interagem com o sistema por meio dos casos de uso. Essa representação visual ajuda a compreender as principais funcionalidades e interações do sistema.

4.2 Diagrama de Classe



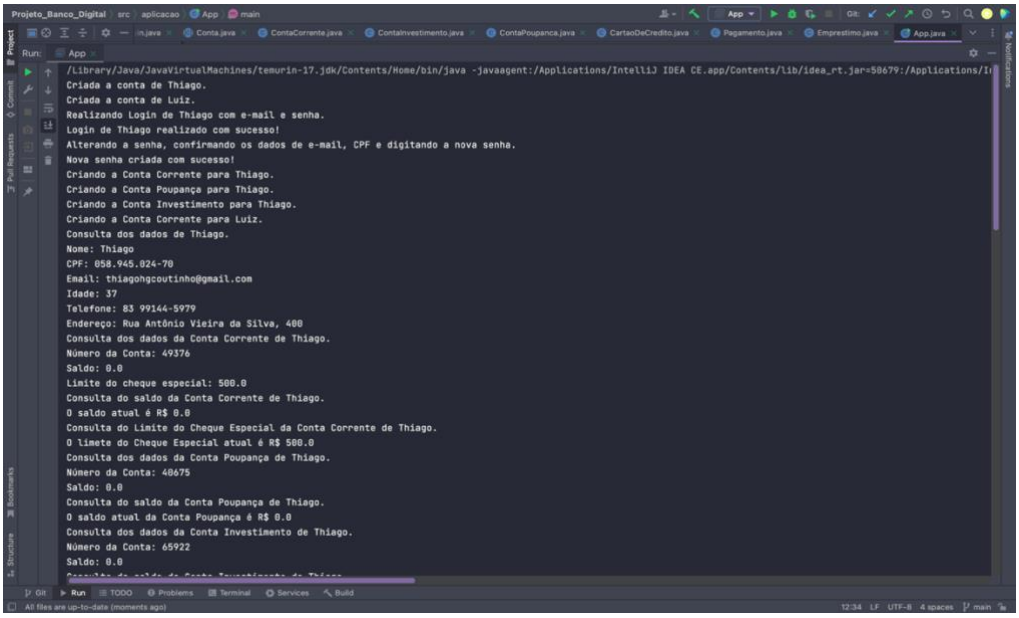
O diagrama de classes apresentado representa a estrutura de classes do sistema e as relações entre elas. Ele mostra as classes do sistema, seus atributos, métodos e as associações entre as classes. Esse diagrama é útil para visualizar a organização das classes e como elas se relacionam, facilitando o entendimento da arquitetura do sistema e a identificação das responsabilidades de cada classe.

4.3 Imagem da Organização dos Pacotes e Método *Main*



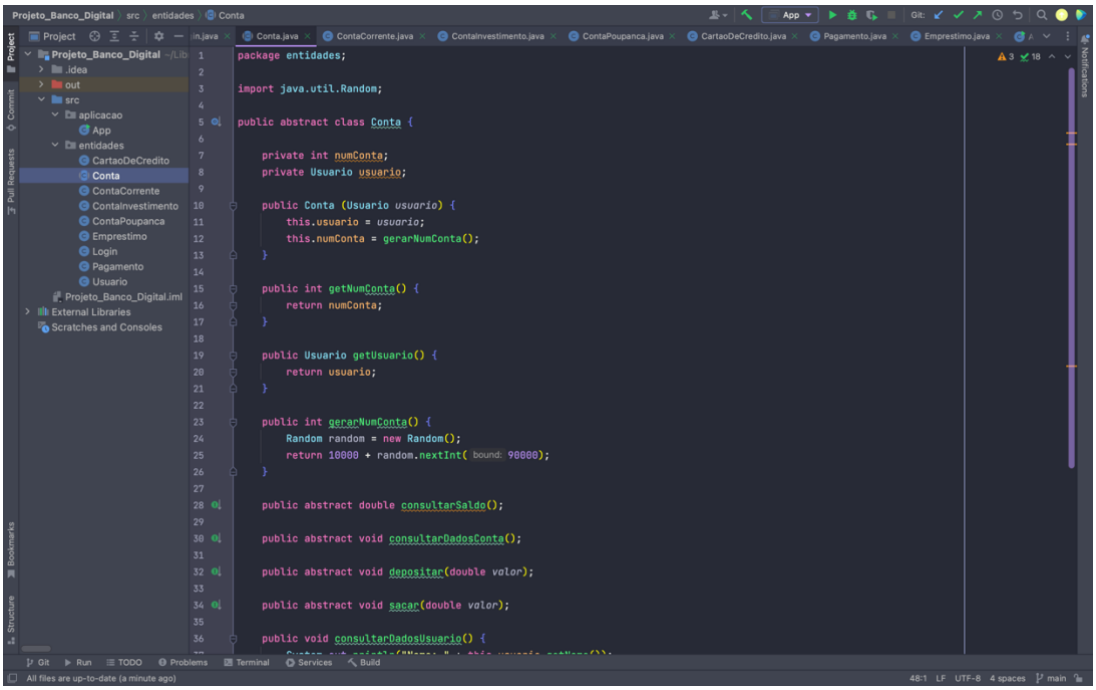
O método *Main* em Java é responsável pela implementação e utilização dos objetos criados ao longo do código, permitindo o acesso aos seus métodos. É onde a execução do programa começa e as ações específicas são definidas para iniciar a aplicação.

4.4 Imagem do Resultado do Método *Main*



No terminal, é possível visualizar os resultados de todos os métodos acionados, fornecendo informações e feedback sobre as ações realizadas durante a execução do programa.

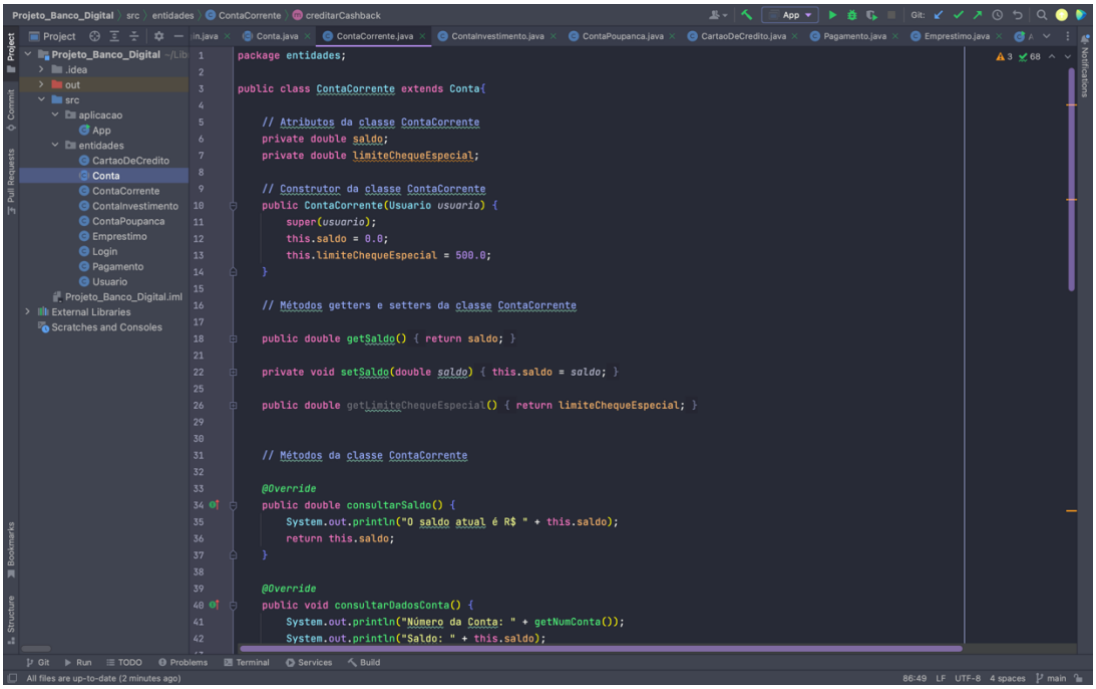
4.5 Imagem da Classe Abstrata Conta



```java
package entidades;

import java.util.Random;

public abstract class Conta {

    private int numConta;
    private Usuario usuario;

    public Conta (Usuario usuario) {
        this.usuario = usuario;
        this.numConta = gerarNumConta();
    }

    public int getNumConta() {
        return numConta;
    }

    public Usuario getUsuario() {
        return usuario;
    }

    public int gerarNumConta() {
        Random random = new Random();
        return 10000 + random.nextInt( bound: 90000);
    }

    public abstract double consultarSaldo();

    public abstract void consultarDadosConta();

    public abstract void depositar(double valor);

    public abstract void sacar(double valor);

    public void consultarDadosUsuario() {
```

A classe abstrata "Conta" é uma estrutura fundamental para as classes filhas, que herdam seus atributos e métodos. Isso proporciona uma base sólida e reutilizável, permitindo a criação de implementações específicas para cada tipo de conta.

4.5 Imagem da Classe Conta Corrente



```java
package entidades;

public class ContaCorrente extends Conta{

    // Atributos da classe ContaCorrente
    private double saldo;
    private double limiteChequeEspecial;

    // Construtor da classe ContaCorrente
    public ContaCorrente(Usuario usuario) {
        super(usuario);
        this.saldo = 0.0;
        this.limiteChequeEspecial = 500.0;
    }

    // Métodos getters e setters da classe ContaCorrente

    public double getSaldo() { return saldo; }

    private void setSaldo(double saldo) { this.saldo = saldo; }

    public double getLimiteChequeEspecial() { return limiteChequeEspecial; }


    // Métodos da classe ContaCorrente

    @Override
    public double consultarSaldo() {
        System.out.println("O saldo atual é R$ " + this.saldo);
        return this.saldo;
    }

    @Override
    public void consultarDadosConta() {
        System.out.println("Número da Conta: " + getNumConta());
        System.out.println("Saldo: " + this.saldo);
```

A classe "ContaCorrente" é uma subclasse da classe abstrata "Conta". Ao herdar atributos e métodos essenciais, ela pode adicionar funcionalidades específicas da conta corrente, mantendo a base comum fornecida pela classe abstrata. Isso promove a reutilização de código e a organização hierárquica das classes.

5 CONSIDERAÇÕES FINAIS

O desenvolvimento do projeto SmartBank resultou em um banco digital inovador, proporcionando uma oportunidade de aprendizado sobre práticas ágeis de desenvolvimento de software. A metodologia baseada no Paradigma de Orientação a Objetos e a modelagem do sistema por meio de diagramas de caso de uso e diagramas de classes contribuíram para uma abordagem modular e flexível. O uso de técnicas de inteligência artificial trouxe benefícios na produção de código e no desenvolvimento de modelos de classes.

5.1 Ameaças à arquitetura do software e padrão de desenvolvimento

Durante o projeto, foram identificadas algumas ameaças relacionadas à arquitetura do software e ao padrão de desenvolvimento adotado. A ausência do padrão MVC (Model-View-Controller) e a falta de aplicação de técnicas de clean code podem comprometer a manutenção e a escalabilidade do sistema a longo prazo. A falta de separação clara das responsabilidades entre os componentes do software pode dificultar a compreensão e a evolução do código, bem como a introdução de novas funcionalidades. Portanto, é importante considerar a revisão da arquitetura e a adoção de padrões de desenvolvimento mais robustos no futuro.

5.2 Trabalhos futuros

Para melhorar o desempenho e a qualidade do projeto, é recomendado investir na reestruturação da arquitetura do software seguindo os princípios do padrão MVC e aplicando técnicas de clean code. Isso ajudará a melhorar a organização do código, facilitar a manutenção e tornar o sistema mais modular e escalável. Além disso, a realização de testes mais abrangentes e a implementação de processos de revisão de código são práticas importantes para garantir a qualidade do software. A busca por atualizações tecnológicas e a exploração de novas tecnologias, como frameworks e bibliotecas, também podem contribuir para o aprimoramento contínuo do SmartBank.

REFERÊNCIAS

GUEDES, Gilleanes T. A. UML - Uma Abordagem Prática. Novatec Editora, 2006.