# RELATO DE EXPERIÊNCIA SOBRE O DESENVOLVIMENTO DE UM SISTEMA DE BANCO DIGITAL

Alfredo Guilherme Orlando Gracio [1]
Gabriel de Pádua de Almeida Costa Benício [2]
Gustavo de Oliveira Viana dos Santos [3]
Marcella Araújo C. R. Gomes [4]

## RESUMO

Java é uma linguagem de programação projetada para ser orientada a objetos. A orientação a objetos é um paradigma de programação que organiza o código em torno de objetos, que são instâncias de classes. Esses objetos possuem atributos (propriedades) e métodos (comportamentos). Em Java, as classes são utilizadas para criar objetos, fornecendo uma descrição ou molde para sua estrutura. Java suporta quatro pilares principais da programação orientada a objetos: encapsulamento, herança, polimorfismo e abstração. O encapsulamento envolve esconder os detalhes internos de um objeto e fornecer uma interface externa para interagir com ele. A herança permite criar classes baseadas em classes existentes, promovendo a reutilização de código. O polimorfismo permite que um objeto se comporte de várias maneiras diferentes, tratando objetos de classes diferentes de maneira uniforme. A abstração envolve identificar características essenciais de um objeto e modelá-las como uma classe abstrata ou uma interface. Nos quais foram utilizados para a criação do projeto de uma conta bancária e suas funcionalidades.

**Palavras-chaves**: Java. Orientação a objeto. Classe. Atributo.

## ABSTRACT

Java is a programming language designed to be object-oriented. Object-oriented programming is a programming paradigm that organizes code around objects, which are instances of classes. These objects have attributes (properties) and methods (behaviors). In Java, classes are used to create objects, providing a description or blueprint for their structure. Java supports four main pillars of object-oriented programming: encapsulation, inheritance, polymorphism, and abstraction. Encapsulation involves hiding the internal details of an object and providing an external interface to interact with it. Inheritance allows creating classes based on existing classes, promoting code reuse. Polymorphism allows an object to behave in different ways,

[1] Graduande do Curso de Sistemas de Informação E-mail: 2022210220028@iesp.edu.br
[2] Graduande do Curso de Sistemas de Informação E-mail: 2021210220011@iesp.edu.br
[3] Graduande do Curso de Sistemas para Internet E-mail: 2022211510069@iesp.edu.br
[4] Graduando do Curso de Sistemas para Internet E-mail: 20222110097@iesp.edu.br
[5] Professor Orientador, Graduade em.... ,pela Universidade.... , Doutorado em .... , Mestrado em.....,
Especialização em... pela Universidade.... Docente do Curso Superior em Tecnologia de Gestão de Terminais e OperaçõesPortuárias da disciplina de... E-mail:....

treating objects from different classes uniformly. Abstraction involves identifying the essential characteristics of an object and modeling them as an abstract class or an interface. These principles were used in the creation of a banking account project and its functionalities.

Keywords: Java. Object-Oriented. Classes. Attributes.

## 1 INTRODUÇÃO

A tecnologia tem revolucionado a forma como as pessoas lidam com suas finanças, e os bancos digitais têm ganhado cada vez mais espaço nesse cenário. Para acompanhar essas tendências, o desenvolvimento de aplicativos bancários se tornou cada vez mais importante e o Java se destaca como uma linguagem de programação ideal para a criação de softwares financeiros. Os bancos digitais surgiram como uma alternativa aos bancos tradicionais, oferecendo aos clientes uma experiência bancária moderna e mais conveniente. Os bancos digitais oferecem uma série de benefícios incluindo taxas de juros mais competitivas, baixas ou até mesmo ausência de tarifas, facilidade no processo de abertura de contas, além da possibilidade de realizar transações bancárias de forma simples e rápida através de aplicações de celular.

O objetivo deste projeto é apresentar uma aplicação em Java que simula as funcionalidades de um banco digital, oferecendo aos usuários uma experiência intuitiva. A aplicação permitirá a realização de operações bancárias, como consultas de saldo e extrato, transferências, pagamentos de contas, investimentos, entre outras funcionalidades.

## 2 FUNDAMENTAÇÃO TEÓRICA

A linguagem de programação Java foi projetada com base no paradigma da programação orientada a objetos. Esse paradigma organiza o código em torno de objetos, que são instâncias de classes. Cada objeto possui atributos, também conhecidos como propriedades, que representam suas características ou estados, e métodos, que definem as ações que o objeto pode executar. Em Java, as classes são usadas para criar objetos. Uma classe é um plano ou modelo que descreve os atributos e métodos que os objetos criados a partir dela terão. A criação de objetos ocorre através da instância da classe, ou seja, da criação de uma instância específica dessa classe.

A programação orientada a objetos em Java é baseada em quatro pilares principais: encapsulamento, herança, polimorfismo e abstração. O encapsulamento envolve ocultar os detalhes internos de um objeto, protegendo seus atributos e métodos e fornecendo uma interface externa para interagir com o objeto. Isso garante a segurança e integridade dos objetos. A herança permite criar classes derivadas com base em classes existentes, estabelecendo uma hierarquia de classes. Isso promove a reutilização de código, uma vez que as classes derivadas herdam os atributos e métodos da classe base.

O polimorfismo permite tratar um objeto de várias maneiras diferentes. Isso significa que um objeto pode ser referenciado por meio de uma classe base, mas executar comportamentos específicos de suas classes derivadas. O polimorfismo aumenta a flexibilidade e extensibilidade do código. Por fim, a abstração envolve identificar as características essenciais de um objeto e modelá-las como uma classe abstrata ou uma interface. A abstração permite

criar uma representação simplificada dos objetos, focando apenas nos detalhes relevantes para o sistema em questão.

Todos esses conceitos da programação orientada a objetos podem ser aplicados na criação de projetos em Java, como, por exemplo, um sistema de conta bancária. O encapsulamento pode ser utilizado para proteger as informações da conta, a herança para criar diferentes tipos de conta com base em uma classe base, o polimorfismo para tratar as contas de forma uniforme, independentemente do tipo específico, e a abstração para simplificar a representação das contas no sistema.

Em resumo, a linguagem de programação Java é orientada a objetos e oferece suporte aos princípios fundamentais desse paradigma. Esses princípios permitem organizar o código de forma modular, reutilizável e extensível, facilitando o desenvolvimento de projetos complexos.

## 3 METODOLOGIA

A metodologia utilizada no desenvolvimento do projeto foi a Orientação a Objetos, que é um paradigma de programação que organiza o código em torno de objetos, encapsulando dados e funcionalidades relacionadas em classes.

O projeto foi estruturado em diversas classes, cada uma desempenhando um papel específico. A classe principal, chamada Application, foi responsável pelo funcionamento geral do aplicativo. Ela coordenou a interação entre as demais classes e gerenciou o fluxo de execução.

Uma classe abstrata, denominada Conta, foi criada como uma classe mãe para as classes filhas Conta Corrente, Investimento e Conta Poupança. A classe Conta possuía um construtor e atributos que eram comuns a todas as contas bancárias. As classes filhas, por sua vez, tinham seus próprios construtores e métodos específicos, como sacar, depositar e verificar o saldo, que eram adaptados às características de cada tipo de conta.

Além das classes relacionadas às contas, foram implementadas as classes Titular e Empréstimos. A classe Titular tinha a finalidade de armazenar informações sobre o titular da conta, enquanto a classe Empréstimos lidava com funcionalidades específicas relacionadas a empréstimos bancários. Ambas as classes possuíam construtores e métodos de acesso (getter and setter) para manipular os dados pertinentes.

A utilização de classes e objetos proporcionou uma organização clara e modularizada do projeto, permitindo a reutilização de código, facilitando a manutenção e proporcionando um maior nível de abstração. Através dessa abordagem, foi possível desenvolver um aplicativo bancário com uma estrutura sólida, flexível e de fácil compreensão.

## 4 RESULTADO E DISCUSSÃO

Durante a fase de desenvolvimento, a aplicação foi construída com base em classes e objetos, seguindo os princípios da Orientação a Objetos. A classe principal denominada Application desempenhou um papel central no funcionamento do aplicativo, coordenando as interações entre as diferentes classes e garantindo a correta execução das funcionalidades.

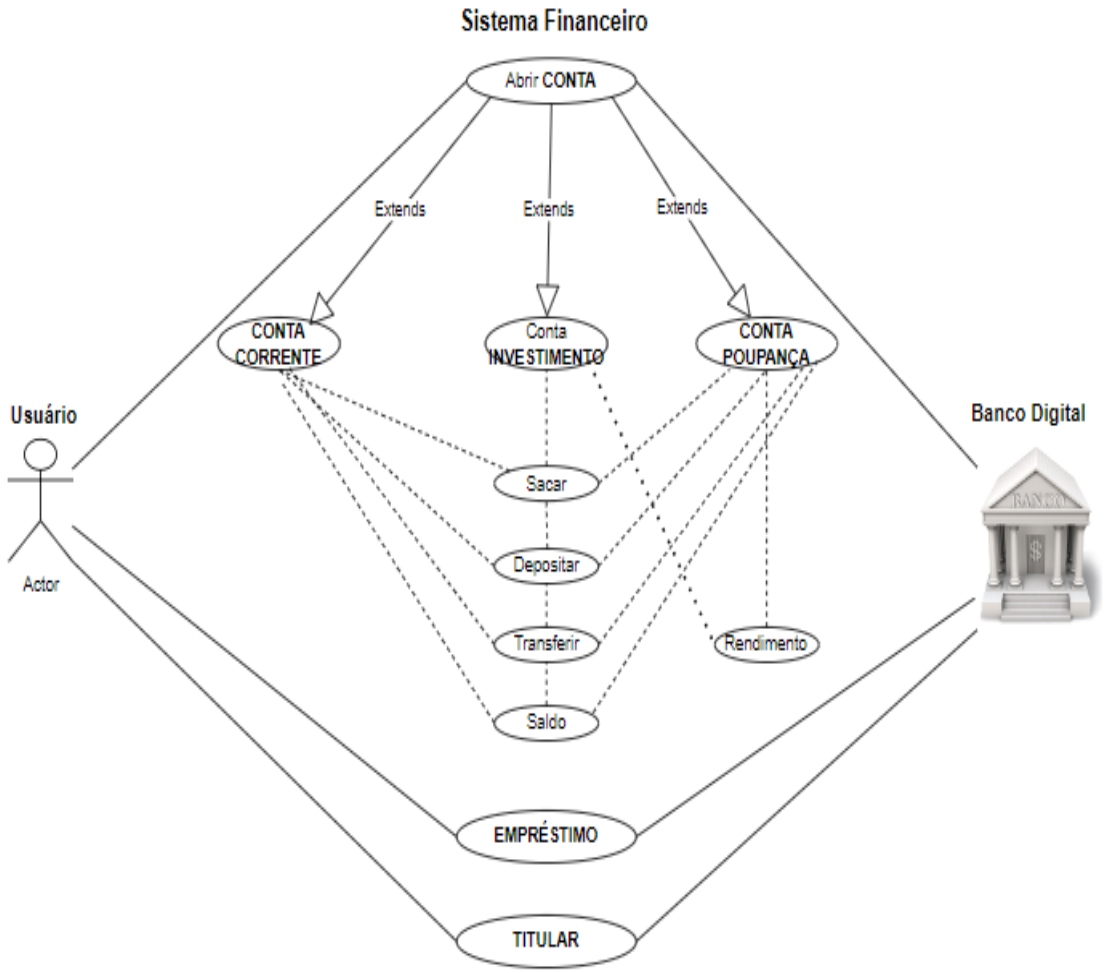
A utilização da classe abstrata Conta permitiu uma estruturação eficiente das contas bancárias, com atributos e métodos comuns a todas elas. Por meio da herança, foram criadas as classes filhas Conta Corrente, Investimento e Conta Poupança, que possuíam construtores e métodos específicos para atender às necessidades particulares de cada tipo de conta. Isso proporcionou uma maior flexibilidade e reutilização de código, evitando a duplicação de implementações semelhantes.

As classes Titular e Empréstimos foram implementadas para armazenar informações sobre os titulares das contas e permitir o gerenciamento de empréstimos bancários. Essas classes, juntamente com seus construtores e métodos de acesso, contribuíram para uma estrutura mais completa do sistema.

Ao aplicar a metodologia de Orientação a Objetos, observamos vantagens significativas. A abstração permitiu uma melhor modelagem dos objetos do mundo real, representando de forma clara e concisa as entidades e suas relações. Além disso, a encapsulação garantiu a integridade dos dados, evitando acessos indevidos e garantindo a consistência das informações.

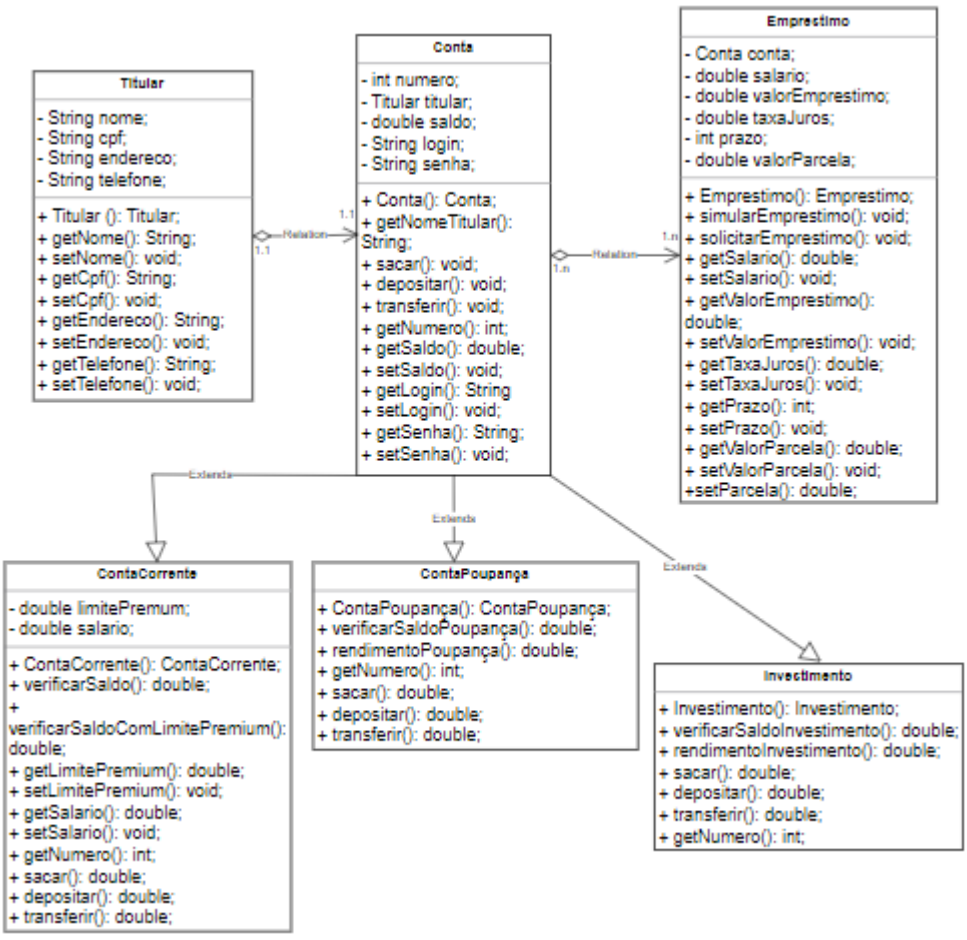
No entanto, algumas limitações foram identificadas durante o desenvolvimento do projeto. A complexidade do sistema bancário exigiu um cuidadoso planejamento e análise prévia, a fim de garantir a correta estruturação das classes e a adequação das funcionalidades. Além disso, a utilização correta dos conceitos de herança e polimorfismo se mostrou crucial para evitar duplicações de código e garantir a flexibilidade do sistema.

## 4.1 Diagrama de Caso de Uso




O diagrama de casos de uso é uma das ferramentas mais importantes para o processo de modelagem de um sistema, permitindo identificar os atores envolvidos e as funcionalidades que o sistema deve oferecer. Neste contexto ele demonstra todos os cenários que o usuário (titular) possui junto ao banco.

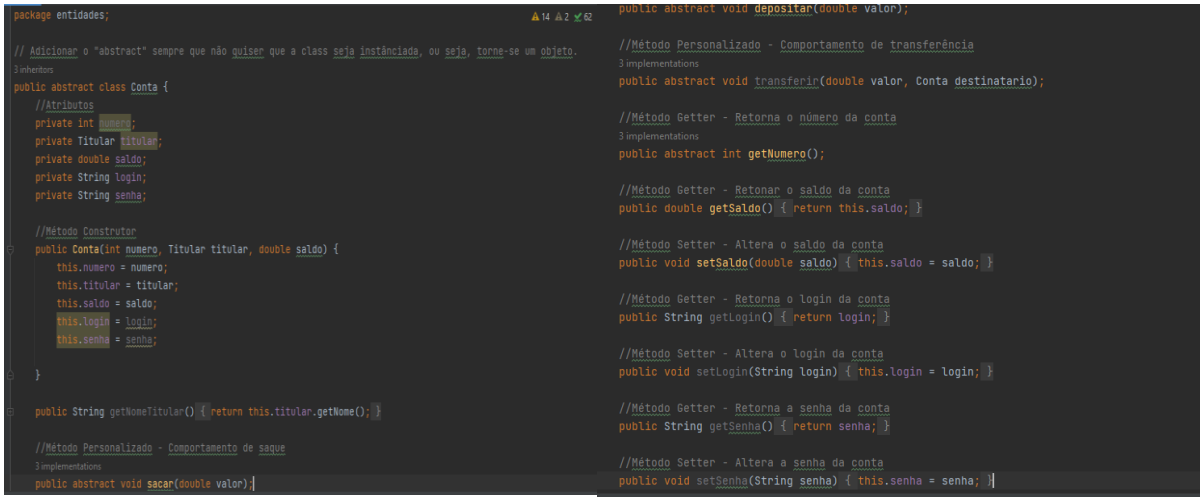
## 4.2 Diagrama de Classe



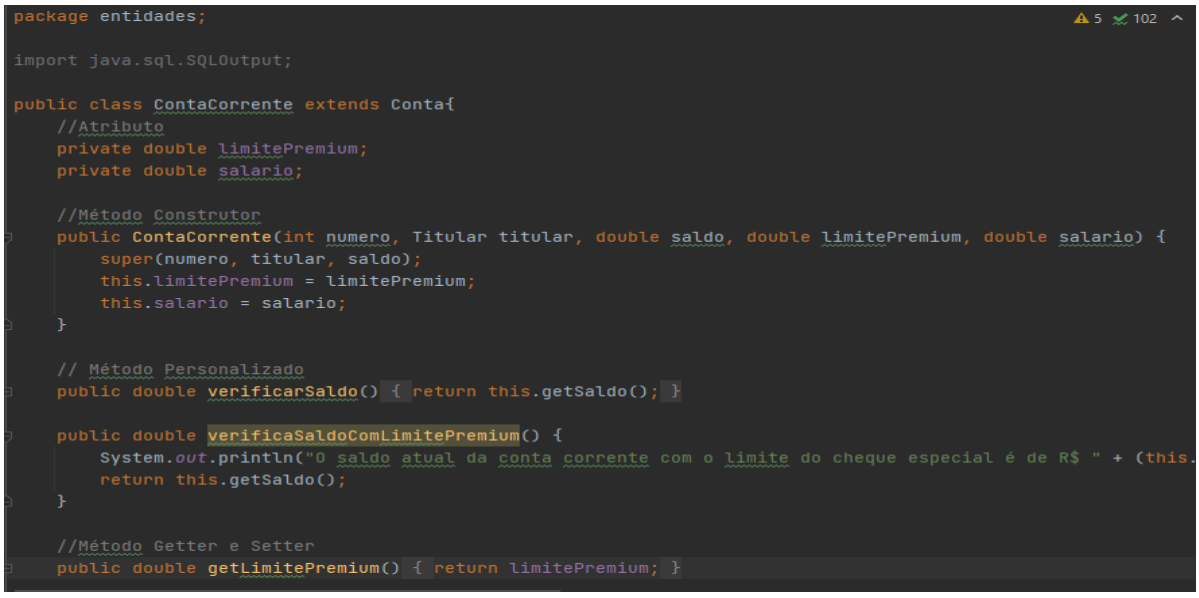

O diagrama de classes é uma ferramenta de modelagem que permite representar a estrutra de classes de um sistema, bem como a relação entre elas. Com a utilização deste diagrama pode-se notar as funções e relações entre as classes:

- Titular: possui os dados dos titulares (clientes);
- Empréstimo: possui funções de simulação e efetivação de um empréstimo;
- Conta: abstrata, não pode ser instanciada e classe mãe das próximas classes;
- Conta corrente: extend, herda as funções e atributos da classe conta;
- Conta poupança: extend, herda as funções e atributos da classe conta, também como a verificação de saldo, sacar, depositar e rentabilidade;
- Investimento: extend, possui os métodos referente a calcular rendimento e calcular montante.

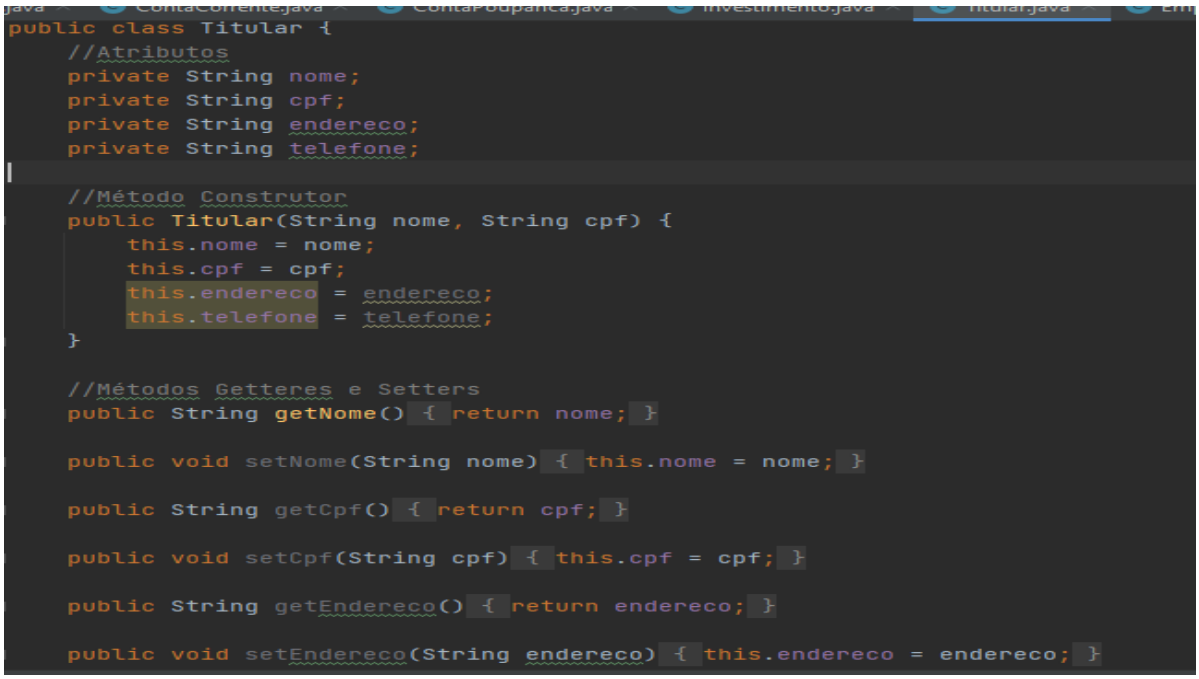
## 4.3 Trechos do Código



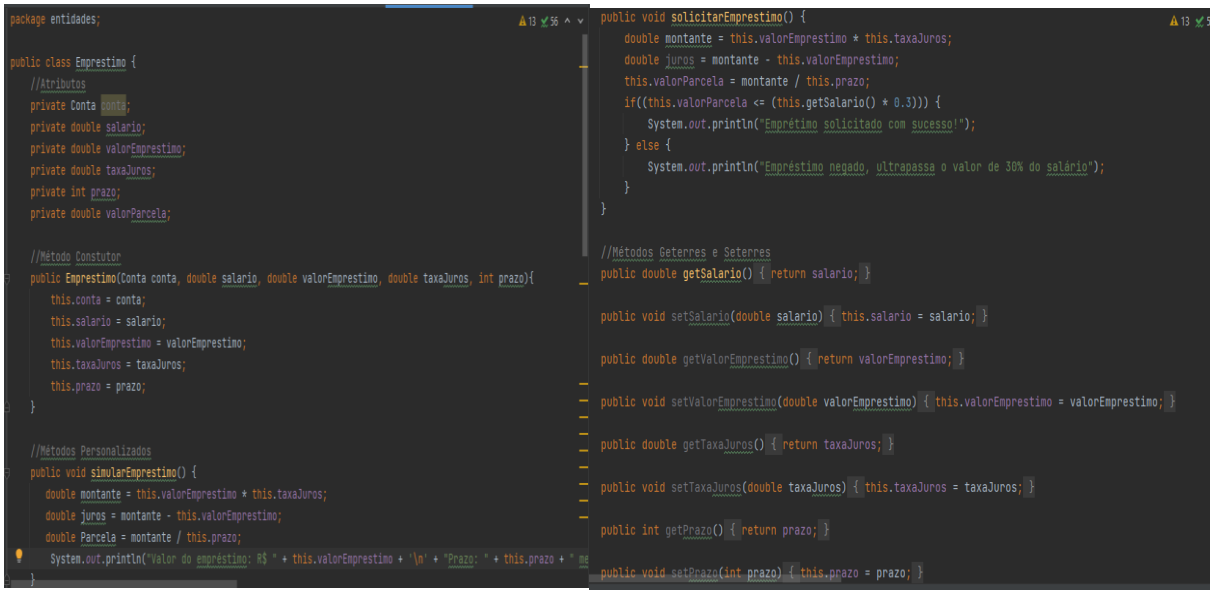
```java
package entidades;

// Adicionar o "abstract" sempre que não quiser que a class seja instânciada, ou seja, torne-se um objeto.
3 inheritors
public abstract class Conta {
    //Atributos
    private int numero;
    private Titular titular;
    private double saldo;
    private String login;
    private String senha;

    //Método Construtor
    public Conta(int numero, Titular titular, double saldo) {
        this.numero = numero;
        this.titular = titular;
        this.saldo = saldo;
        this.login = login;
        this.senha = senha;

    }

    public String getNomeTitular() { return this.titular.getNome(); }

    //Método Personalizado - Comportamento de saque
    3 implementations
    public abstract void sacar(double valor);
    public abstract void depositar(double valor);

    //Método Personalizado - Comportamento de transferência
    3 implementations
    public abstract void transferir(double valor, Conta destinatario);

    //Método Getter - Retorna o número da conta
    3 implementations
    public abstract int getNumero();

    //Método Getter - Retonar o saldo da conta
    public double getSaldo() { return this.saldo; }

    //Método Setter - Altera o saldo da conta
    public void setSaldo(double saldo) { this.saldo = saldo; }

    //Método Getter - Retorna o login da conta
    public String getLogin() { return login; }

    //Método Setter - Altera o login da conta
    public void setLogin(String login) { this.login = login; }

    //Método Getter - Retorna a senha da conta
    public String getSenha() { return senha; }

    //Método Setter - Altera a senha da conta
    public void setSenha(String senha) { this.senha = senha; }
```



```java
package entidades;

import java.sql.SQLOutput;

public class ContaCorrente extends Conta{
    //Atributo
    private double limitePremium;
    private double salario;

    //Método Construtor
    public ContaCorrente(int numero, Titular titular, double saldo, double limitePremium, double salario) {
        super(numero, titular, saldo);
        this.limitePremium = limitePremium;
        this.salario = salario;
    }

    // Método Personalizado
    public double verificarSaldo() { return this.getSaldo(); }

    public double verificaSaldoComLimitePremium() {
        System.out.println("O saldo atual da conta corrente com o limite do cheque especial é de R$ " + (this.
        return this.getSaldo();
    }

    //Método Getter e Setter
    public double getLimitePremium() { return limitePremium; }
```

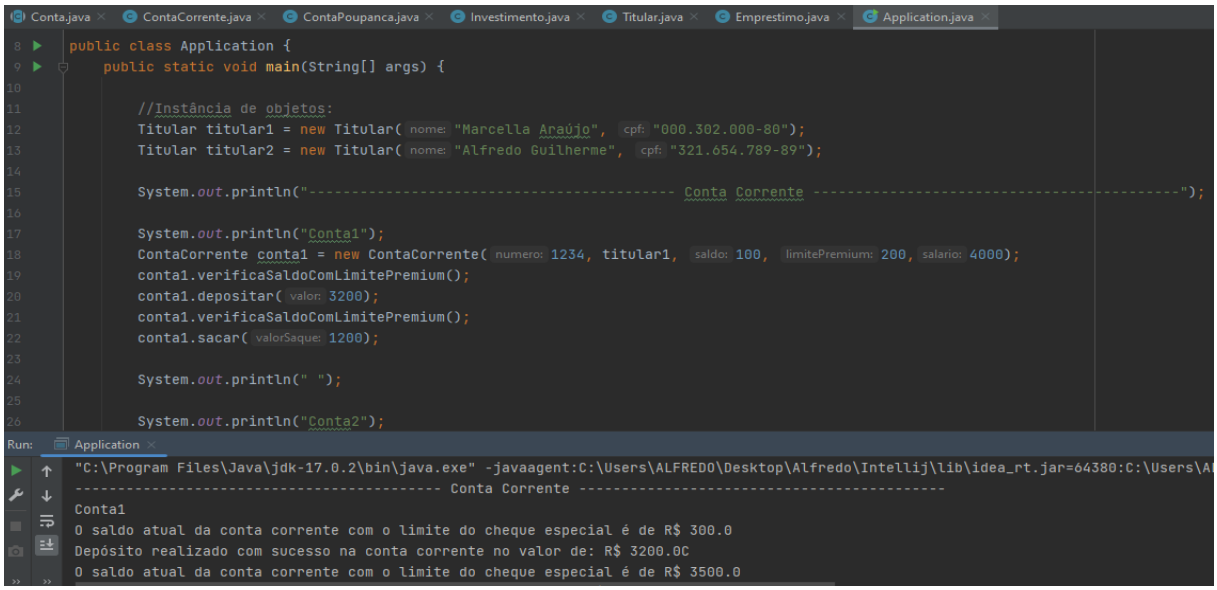


```java
public class Titular {
    //Atributos
    private String nome;
    private String cpf;
    private String endereco;
    private String telefone;

    //Método Construtor
    public Titular(String nome, String cpf) {
        this.nome = nome;
        this.cpf = cpf;
        this.endereco = endereco;
        this.telefone = telefone;
    }

    //Métodos Getteres e Setters
    public String getNome() { return nome; }

    public void setNome(String nome) { this.nome = nome; }

    public String getCpf() { return cpf; }

    public void setCpf(String cpf) { this.cpf = cpf; }

    public String getEndereco() { return endereco; }

    public void setEndereco(String endereco) { this.endereco = endereco; }
```



```java
package entidades;

public class Emprestimo {
    //Atributos
    private Conta conta;
    private double salario;
    private double valorEmprestimo;
    private double taxaJuros;
    private int prazo;
    private double valorParcela;

    //Método Constutor
    public Emprestimo(Conta conta, double salario, double valorEmprestimo, double taxaJuros, int prazo){
        this.conta = conta;
        this.salario = salario;
        this.valorEmprestimo = valorEmprestimo;
        this.taxaJuros = taxaJuros;
        this.prazo = prazo;
    }

    //Métodos Personalizados
    public void simularEmprestimo() {
        double montante = this.valorEmprestimo * this.taxaJuros;
        double juros = montante - this.valorEmprestimo;
        double Parcela = montante / this.prazo;
        System.out.println("Valor do empréstimo: R$ " + this.valorEmprestimo + '\n' + "Prazo: " + this.prazo + " me
    }
```

```java
public void solicitarEmprestimo() {
    double montante = this.valorEmprestimo * this.taxaJuros;
    double juros = montante - this.valorEmprestimo;
    this.valorParcela = montante / this.prazo;
    if((this.valorParcela <= (this.getSalario() * 0.3))) {
        System.out.println("Emprétimo solicitado com sucesso!");
    } else {
        System.out.println("Empréstimo negado, ultrapassa o valor de 30% do salário");
    }
}

//Métodos Geterres e Seterres
public double getSalario() { return salario; }

public void setSalario(double salario) { this.salario = salario; }

public double getValorEmprestimo() { return valorEmprestimo; }

public void setValorEmprestimo(double valorEmprestimo) { this.valorEmprestimo = valorEmprestimo; }

public double getTaxaJuros() { return taxaJuros; }

public void setTaxaJuros(double taxaJuros) { this.taxaJuros = taxaJuros; }

public int getPrazo() { return prazo; }

public void setPrazo(int prazo) { this.prazo = prazo; }
```



```java
public class Application {
    public static void main(String[] args) {

        //Instância de objetos:
        Titular titular1 = new Titular( nome: "Marcella Araújo", cpf: "000.302.000-80");
        Titular titular2 = new Titular( nome: "Alfredo Guilherme", cpf: "321.654.789-89");

        System.out.println("---------- Conta Corrente ----------");

        System.out.println("Conta1");
        ContaCorrente conta1 = new ContaCorrente( numero: 1234, titular1, saldo: 100, limitePremium: 200, salario: 4000);
        conta1.verificaSaldoComLimitePremium();
        conta1.depositar( valor: 3200);
        conta1.verificaSaldoComLimitePremium();
        conta1.sacar( valorSaque: 1200);

        System.out.println(" ");

        System.out.println("Conta2");
```

```
Run: Application
"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" -javaagent:C:\Users\ALFREDO\Desktop\Alfredo\Intellij\lib\idea_rt.jar=64380:C:\Users\A
---------- Conta Corrente ----------
Conta1
O saldo atual da conta corrente com o limite do cheque especial é de R$ 300.0
Depósito realizado com sucesso na conta corrente no valor de: R$ 3200.0C
O saldo atual da conta corrente com o limite do cheque especial é de R$ 3500.0
```

## 5 CONSIDERAÇÕES FINAIS

Ao finalizar o projeto, podemos refletir sobre as conquistas e desafios enfrentados ao longo do processo. O objetivo principal era criar um sistema eficiente, seguro e fácil de usar, que oferecesse aos usuários uma experiência bancária completa e conveniente. Destacamos a importância de uma arquitetura bem definida. Ao utilizar a linguagem Java, foi possível criar uma estrutura modular e escalável para o banco digital. A orientação a objetos possibilitou a criação de classes e interfaces que representam as entidades e funcionalidades do sistema, facilitando o desenvolvimento e a manutenção do código.

### 5.1 AMEAÇAS AO PROJETO

Ao desenvolver um projeto de banco digital em Java, é essencial estar ciente das possíveis ameaças que podem comprometer a segurança, a eficiência e o sucesso do projeto. Com isso em mente entendemos que, erros de programação e bugs são as principais ameaças

que podem afetar o funcionamento correto do sistema bancário. Bugs não identificados podem levar a falhas no processamento de transações, perda de dados e inconsistências nos saldos dos usuários. A adoção de práticas de desenvolvimento de software de qualidade, como testes rigorosos, revisões de código e depuração, é fundamental para minimizar essas ameaças.

## 5.2 TRABALHOS FUTUROS

Após a conclusão do projeto é importante considerar os trabalhos futuros para aprimorar e expandir o sistema. Temos como ideia a implementação de novas funcionalidades. À medida que o setor bancário e as demandas dos clientes continuam a evoluir, é essencial acompanhar as tendências e implementar novas funcionalidades que atendam às necessidades em constante mudança. Isso pode incluir recursos como pagamento por aproximação (NFC), integração com assistentes virtuais, suporte a criptomoedas e investimentos automatizados. O desenvolvimento contínuo de novas funcionalidades manterá o banco digital competitivo e relevante no mercado.

## REFERÊNCIAS

VICTOR, A. **Programacao Orientada a Objetos com Java**. Disponível em: <https://www.devmedia.com.br/programacao-orientada-a-objetos-com-java/18449>.

**Java in a nutshell**. Cambridge ; Köln ; Paris ; Sebastopol ; Tokyo: O’reilly, 1997.

GILBERT, S.; MCCARTY, B. **Object Oriented Programming in Java**. [s.l: s.n.].