



UNIVERSIDADE PITÁGORAS UNOPAR

Engenharia de Software
Linguagem Orientada a Objetos

AUTOR: ALLAS MAYCON DO VALLE

Gerenciador de Sistema Bancário Java

AUTOR: ALLAS MAYCON DO VALLE

Gerenciador de Sistema Bancário Java

Este projeto é um simulador de banco feito em Java, onde tudo está em um só arquivo. Criamos uma “carteira virtual” que guarda seu nome, CPF e saldo. O programa funciona como um menu simples: você digita a opção (1 para saldo, 2 para depósito, etc.) e ele repete até você pedir para sair. Isso mostrou como organizar informações e fazer o computador seguir comandos de forma segura.

Orientador:

Tutor à Distância: Frederico Aparecido Faedo Pinto.

Prof. Leonardo Santiago

Cidade de Serrana, SP
2025

SUMÁRIO

1	INTRODUÇÃO	3
2	DESENVOLVIMENTO	4
3	ARQUITETURA E ESTRUTURA DO CÓDIGO	5
3.1	A CLASSE PRINCIPAL (GERENCIABANCO)	6
3.2	A CLASSE DE MODELO (CONTABANCARIA)	7
3.3.	MODELAGEM (ATRIBUTOS E ENCAPSULAMENTO)	13
4	LÓGICA DE CONTROLE DE FLUXO	14
5	CONCLUSÃO	15
6	REFERÊNCIAS.....	16

1 INTRODUÇÃO

Este projeto é o resultado do desenvolvimento acadêmico “Sistema de Gerenciamento Bancário Simples”, realizado na disciplina de programação Orientada a Objetos (POO) em Java. O projeto visa simular as operações básicas de uma conta bancária, como cadastro de cliente, consulta de saldo, depósito e saque, utilizando conceitos fundamentais da linguagem Java e da organização de projetos.

O objetivo principal deste trabalho é desenvolver uma aplicação de console em Java capaz de gerenciar procedimentos básicos de uma conta bancária.

Os objetivos são, desenvolver uma aplicação de console em Java capaz de gerenciar procedimentos básicos de uma conta bancária., aplicando conceitos de Programação Orientada a Objetos (POO) através da criação de classes e encapsulamento, utilizando o **Maven** para estruturação e gerenciamento do projeto, implementando as estruturas de controle de fluxo (**do... while e switch... case**) para a navegação do usuário através de um menu interativo, trabalhando com entradas de dados do usuário via Scanner.

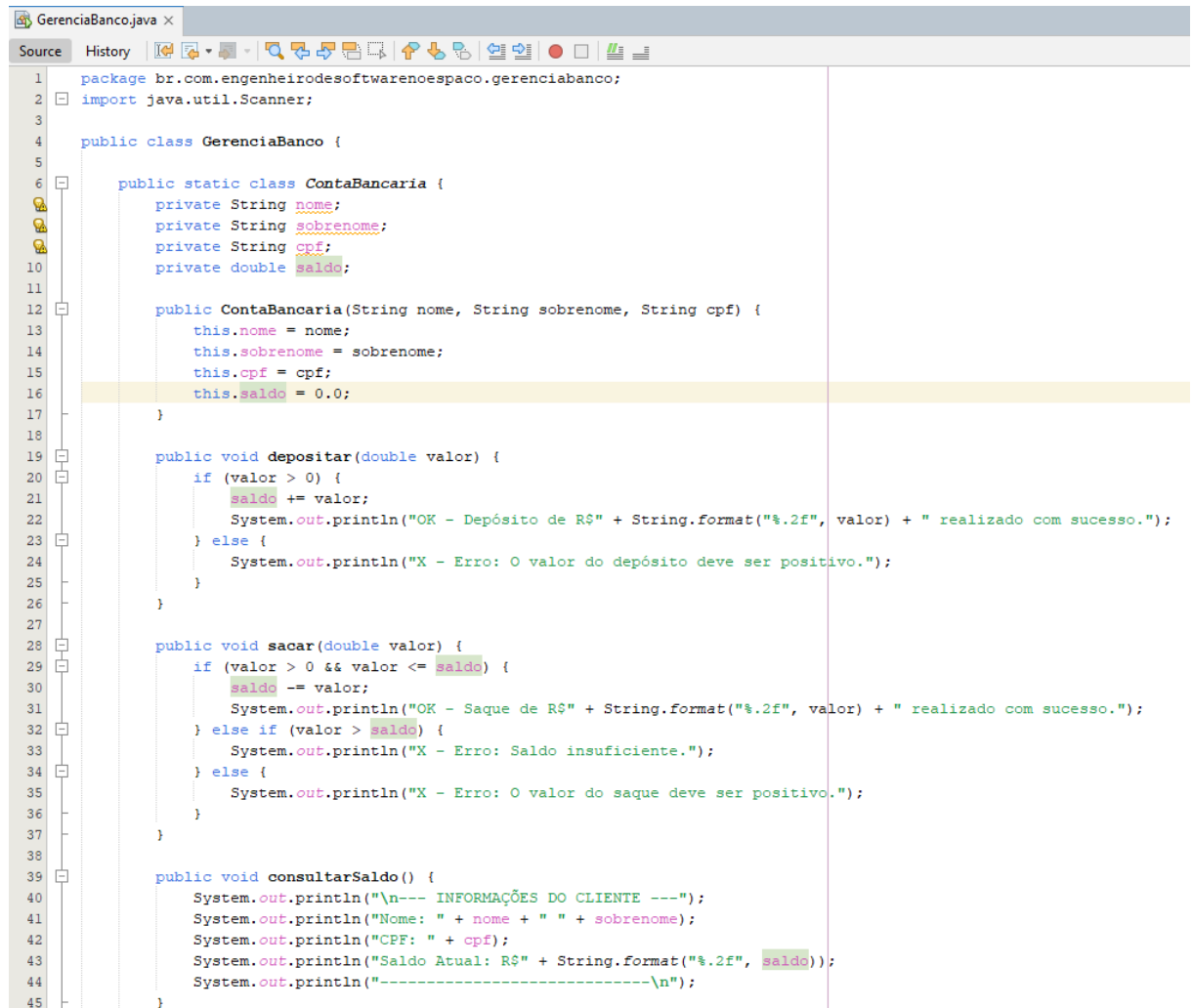
No projeto não utilizamos nada tão robusto para que o código se manter limpo e de fácil leitura e interpretação de qualquer usuário, usamos operações básicas matemáticas, alocação de memória local mesmo e encapsulando toda as informações.

2 DESENVOLVIMENTO

Tecnologias e Ferramentas usadas neste projeto foram: **Java**, uma **Linguagem de programação** usada na disciplina durante o andamento do aprendizado e muito usada no mercado para sistemas robustos, **Gerenciado Apache Maven**, utilizado para padronizar a estrutura do projeto (src/main/java) e gerenciar dependências (embora o projeto atual não utilize dependências externas, o uso do **Maven** foi uma exigência feita para conclusão da disciplina), e a **IDE NetBeans**, mas poderíamos usar para obter o mesmo resultado as **IDEs IntelliJ** e **Eclipse**, que são **Ambientes de Desenvolvimento Integrado** utilizados para codificação, compilação e execução.

3 ARQUITETURA E ESTRUTURA DO CÓDIGO

O projeto foi construído em um **único arquivo Java** (GerenciaBanco.Java), conforme a exigência do trabalho, contendo duas classes principais:

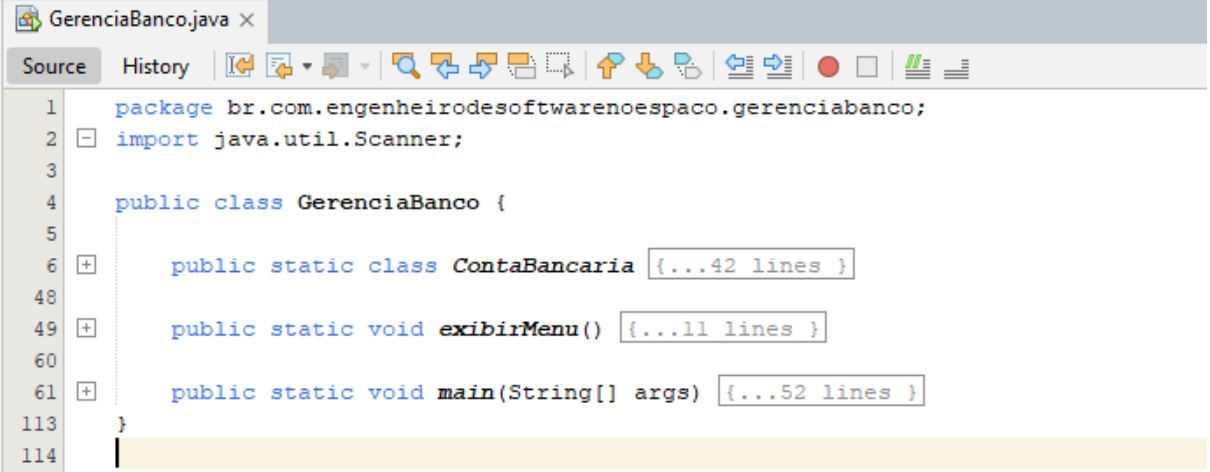


```
1 package br.com.engenheirodesoftwarenoespaco.gerenciabanco;
2 import java.util.Scanner;
3
4 public class GerenciaBanco {
5
6     public static class ContaBancaria {
7         private String nome;
8         private String sobrenome;
9         private String cpf;
10        private double saldo;
11
12        public ContaBancaria(String nome, String sobrenome, String cpf) {
13            this.nome = nome;
14            this.sobrenome = sobrenome;
15            this.cpf = cpf;
16            this.saldo = 0.0;
17        }
18
19        public void depositar(double valor) {
20            if (valor > 0) {
21                saldo += valor;
22                System.out.println("OK - Depósito de R$" + String.format("%.2f", valor) + " realizado com sucesso.");
23            } else {
24                System.out.println("X - Erro: O valor do depósito deve ser positivo.");
25            }
26        }
27
28        public void sacar(double valor) {
29            if (valor > 0 && valor <= saldo) {
30                saldo -= valor;
31                System.out.println("OK - Saque de R$" + String.format("%.2f", valor) + " realizado com sucesso.");
32            } else if (valor > saldo) {
33                System.out.println("X - Erro: Saldo insuficiente.");
34            } else {
35                System.out.println("X - Erro: O valor do saque deve ser positivo.");
36            }
37        }
38
39        public void consultarSaldo() {
40            System.out.println("\n--- INFORMAÇÕES DO CLIENTE ---");
41            System.out.println("Nome: " + nome + " " + sobrenome);
42            System.out.println("CPF: " + cpf);
43            System.out.println("Saldo Atual: R$" + String.format("%.2f", saldo));
44            System.out.println("-----\n");
45        }
46    }
47 }
```

Imagem do projeto base para ilustração do projeto – Fonte: Projeto Gerencia Banco em Java. Autor: Allas Maycon

3.1 A CLASSE PRINCIPAL (GERENCIABANCO)

É a classe GerenciaBanco, que contém o método main(String[] args) e é responsável pelo fluxo de execução do programa, inicializando o sistema, coletando dados do cliente e executando o loop de menu.



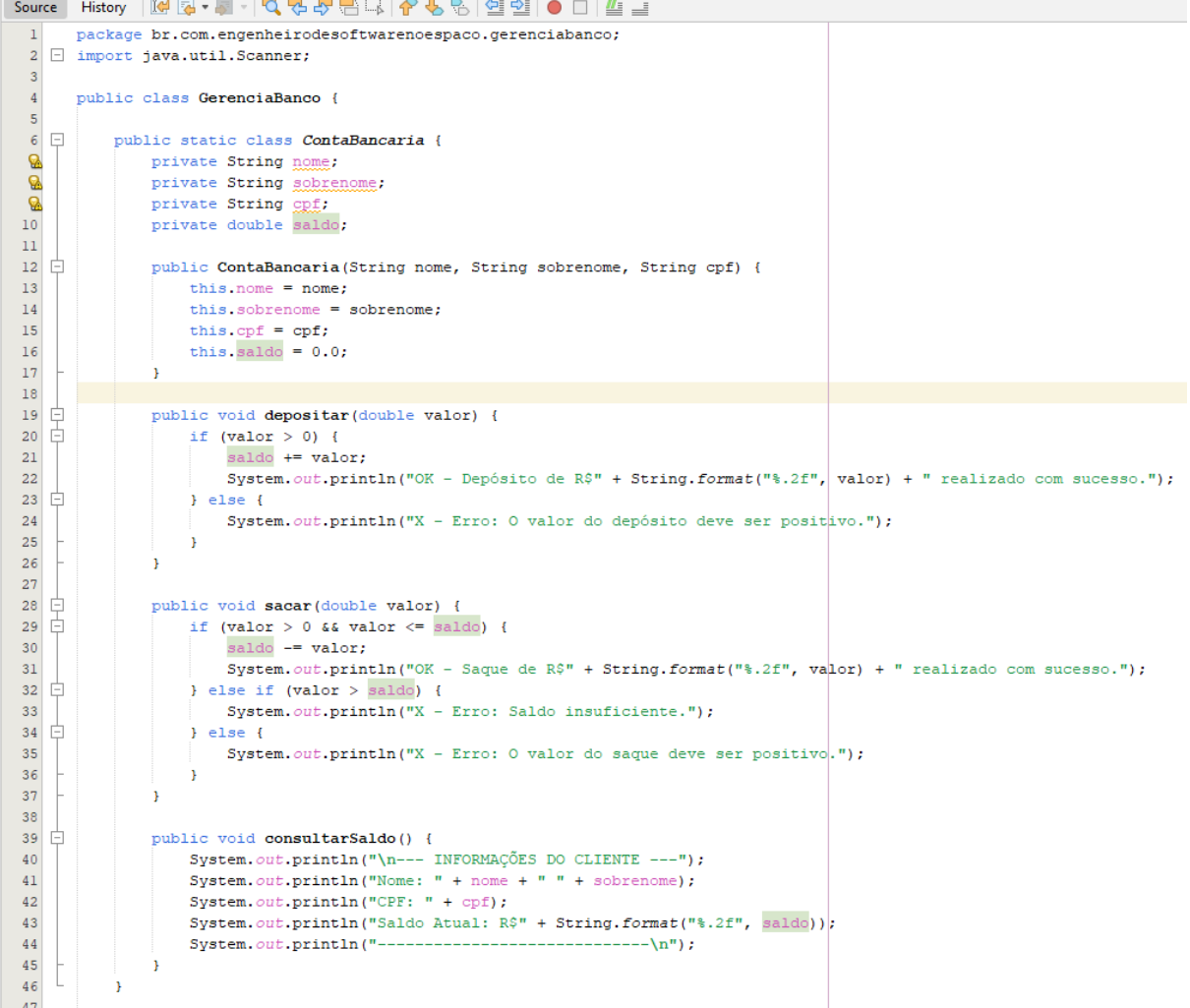
```
1 package br.com.engenheirodesoftwarenoespaco.gerenciabanco;
2 import java.util.Scanner;
3
4 public class GerenciaBanco {
5
6     public static class ContaBancaria {...42 lines }
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49     public static void exibirMenu() {...11 lines }
50
51
52
53
54
55
56
57
58
59
60
61     public static void main(String[] args) {...52 lines }
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113 }
114
```

Imagem do projeto base para ilustração do projeto – Fonte: Projeto Gerencia Banco em Java. Autor: Allas Maycon

Por motivos de criar primeiramente os requisitos do programa, optei por deixar o main pra fazer no final como finalização do projeto, assim facilitou apenas coletar os dados necessários para o funcionamento do programa. Normalmente crio as funcionalidades do programa e testo com dados fictícios para validação dos algoritmos, assim já inicio o processo de coleta de dados logo após no main.

3.2 A CLASSE DE MODELO (CONTABANCARIA)

É a classe ContaBancaria, definida como uma **Inner Class Estática** dentro da classe principal, representa o objeto do mundo real (a conta bancária).



```
1 package br.com.engenheirodesoftwarenoespaco.gerenciabanco;
2 import java.util.Scanner;
3
4 public class GerenciaBanco {
5
6     public static class ContaBancaria {
7         private String nome;
8         private String sobrenome;
9         private String cpf;
10        private double saldo;
11
12        public ContaBancaria(String nome, String sobrenome, String cpf) {
13            this.nome = nome;
14            this.sobrenome = sobrenome;
15            this.cpf = cpf;
16            this.saldo = 0.0;
17        }
18
19        public void depositar(double valor) {
20            if (valor > 0) {
21                saldo += valor;
22                System.out.println("OK - Depósito de R$" + String.format("%.2f", valor) + " realizado com sucesso.");
23            } else {
24                System.out.println("X - Erro: O valor do depósito deve ser positivo.");
25            }
26        }
27
28        public void sacar(double valor) {
29            if (valor > 0 && valor <= saldo) {
30                saldo -= valor;
31                System.out.println("OK - Saque de R$" + String.format("%.2f", valor) + " realizado com sucesso.");
32            } else if (valor > saldo) {
33                System.out.println("X - Erro: Saldo insuficiente.");
34            } else {
35                System.out.println("X - Erro: O valor do saque deve ser positivo.");
36            }
37        }
38
39        public void consultarSaldo() {
40            System.out.println("\n--- INFORMAÇÕES DO CLIENTE ---");
41            System.out.println("Nome: " + nome + " " + sobrenome);
42            System.out.println("CPF: " + cpf);
43            System.out.println("Saldo Atual: R$" + String.format("%.2f", saldo));
44            System.out.println("-----\n");
45        }
46    }
47 }
```

Imagem do projeto base para ilustração do projeto – Fonte: Projeto Gerencia Banco em Java. Autor: Allas Maycon

Vamos analisar este trecho para explicar:

```
public static class ContaBancaria {
    private String nome;
    private String sobrenome;
    private String cpf;
    private double saldo;

    // Construtor para inicializar o cliente
    public ContaBancaria(String nome, String sobrenome, String cpf) {
        this.nome = nome;
        this.sobrenome = sobrenome;
        this.cpf = cpf;
        this.saldo = 0.0; // Saldo inicial é zero
    }

    public void depositar(double valor) {
        if (valor > 0) {
            saldo += valor;
            System.out.println("OK - Depósito de R$" + String.format("%.2f",
valor) + " realizado com sucesso.");
        } else {
            System.out.println("X - Erro: O valor do depósito deve ser positivo.");
        }
    }

    public void sacar(double valor) {
        if (valor > 0 && valor <= saldo) {
            saldo -= valor;
            System.out.println("OK - Saque de R$" + String.format("%.2f", valor)
+ " realizado com sucesso.");
        } else if (valor > saldo) {
            System.out.println("X - Erro: Saldo insuficiente.");
        } else {

```

```

        System.out.println("X - Erro: O valor do saque deve ser positivo.");
    }
}

public void consultarSaldo() {
    System.out.println("\n--- INFORMAÇÕES DO CLIENTE ---");
    System.out.println("Nome: " + nome + " " + sobrenome);
    System.out.println("CPF: " + cpf);
    System.out.println("Saldo Atual: R$" + String.format("%.2f", saldo));
    System.out.println("-----\n");
}

```

Este acima é o trecho do código completo da classe ContaBancaria, declarei a a **Classe** como public static class ContaBancaria, para indicar que é uma classe pública e acessível por outras partes do código, isso também como estática, podendo assim usar a mesma dentro da classe principal sem precisar de criar uma instância para usa-la.

Temos os **Dados da Conta** private String nome; private String sobrenome; , private String cpf; , private double saldo; e sempre usando o modificador de encapsulamento private para ambos os dados para que nada mude esses valores diretamente e ter acesso apenas em alguns trechos necessários do código.

Temos o Construtor base da ContaBancaria:

```

public ContaBancaria(String nome, String sobrenome, String cpf) {
    this.nome = nome;
    this.sobrenome = sobrenome;
    this.cpf = cpf;
    this.saldo = 0.0;
}

```

Imagem do projeto base para ilustração do projeto – Fonte: Projeto Gerencia Banco em Java. Autor: Allas Maycon

Esté é um método especial que é chamado automaticamente apenas uma vez, quando criamos a conta por exemplo (ContaBancaria conta = new ContaBancaria(...) no main).

Está função recebe os dados essenciais (nome, sobrenome, cpf) e os usa para iniciar os atributos correspondentes do objeto. Garantimos também que toda conta comece com o saldo = 0.0.

Já o (this) é feito para diferenciar o atributo da classe (this.nome) do parâmetro no método (nome), usamos para apontar qual atributo da classe queremos atribuir os dados recebidos.

Os **Métodos** (Ações da Conta) depositar, sacar e consultarSaldo.

```
public void depositar(double valor) {  
    if (valor > 0) {  
        saldo += valor;  
        System.out.println("OK - Depósito de R$" + String.format("%.2f", valor) + " realizado com sucesso.");  
    } else {  
        System.out.println("X - Erro: O valor do depósito deve ser positivo.");  
    }  
}
```

Imagem do projeto base para ilustração do projeto – Fonte: Projeto Gerencia Banco em Java. Autor: Allas Maycon

Validamos a Regra de Negócio no bloco if (valor > 0) garantindo que o banco sóa ceite depósitos de valores positivos, protegendo a integridade dos dados e do sistema. A Operação de validação for positiva, a linha saldo += valor; atualiza o saldo da conta.

```
public void sacar(double valor) {  
    if (valor > 0 && valor <= saldo) {  
        saldo -= valor;  
        System.out.println("OK - Saque de R$" + String.format("%.2f", valor) + " realizado com sucesso.");  
    } else if (valor > saldo) {  
        System.out.println("X - Erro: Saldo insuficiente.");  
    } else {  
        System.out.println("X - Erro: O valor do saque deve ser positivo.");  
    }  
}
```

Imagem do projeto base para ilustração do projeto – Fonte: Projeto Gerencia Banco em Java. Autor: Allas Maycon

Validamos a Segurança dos dados neste método que contém duas importantes regras de negócio:

1. O valor do saque deve ser positivo (valor > 0).
2. O saldo na conta deve ser suficiente para cobrir o saque (valor <= saldo).

E a operado na linha `saldo -= valor`; só é executada se ambas as condições de segurança forem atendidas. Se houver erro, a mensagem será exibida.

```
public void consultarSaldo() {  
    System.out.println("\n--- INFORMAÇÕES DO CLIENTE ---");  
    System.out.println("Nome: " + nome + " " + sobrenome);  
    System.out.println("CPF: " + cpf);  
    System.out.println("Saldo Atual: R$" + String.format("%.2f", saldo));  
    System.out.println("-----\n");  
}
```

Imagem do projeto base para ilustração do projeto – Fonte: Projeto Gerencia Banco em Java. Autor: Allas Maycon

A função é um método simples de leitura (um **Getter** que já exibe o resultado formatado). Acessando os atributos internos do objeto e apresentando ao usuário de forma organizada, formatando os dados em casas decimais (`R$%.2f`) neste caso, duas casas decimais.

Temos a exibição do menu:

```
public static void exibirMenu() {  
    System.out.println("\n=====");  
    System.out.println("|          MENU PRINCIPAL          |");  
    System.out.println("=====");  
    System.out.println("1 - Consultar Saldo e Dados");  
    System.out.println("2 - Depósito");  
    System.out.println("3 - Saque");  
    System.out.println("0 - Sair");  
    System.out.println("=====");  
    System.out.print("Escolha uma opção: ");  
}
```

Imagem do projeto base para ilustração do projeto – Fonte: Projeto Gerencia Banco em Java. Autor: Allas Maycon

Onde o qual apresentamos o menu no console ao usuário para escolha de qual decisão tomar no uso do programa.

E para o final temos a nosso **main**:

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.println("--- BEM-VINDO AO GERENCIADOR BANCÁRIO ---");
    System.out.print("Informe seu Nome: ");
    String nome = scanner.nextLine();
    System.out.print("Informe seu Sobrenome: ");
    String sobrenome = scanner.nextLine();
    System.out.print("Informe seu CPF: ");
    String cpf = scanner.nextLine();
    System.out.println("-----");
    ContaBancaria conta = new ContaBancaria(nome, sobrenome, cpf);

    int opcao;
    do {
        exibirMenu();
        if (scanner.hasNextInt()) {
            opcao = scanner.nextInt();
        } else {
            System.out.println("X - Entrada inválida. Por favor, digite um número.");
            scanner.next();
            opcao = -1;
            continue;
        }

        switch (opcao) {
            case 1:
                conta.consultarSaldo();
                break;
            case 2:
                System.out.print("Digite o valor para depósito: R$");
                double valorDeposito = scanner.nextDouble();
                conta.depositar(valorDeposito);
                break;
            case 3:
                System.out.print("Digite o valor para saque: R$");
                double valorSaque = scanner.nextDouble();
                conta.sacar(valorSaque);
                break;
            case 0:
                System.out.println("\nObrigado por utilizar o Gerenciador Banco! Encerrando o sistema...");
                break;
            default:
                System.out.println("Opção inválida. Por favor, escolha uma opção do menu.");
        }
    } while (opcao != 0);
    scanner.close();
}
```

Imagem do projeto base para ilustração do projeto – Fonte: Projeto Gerencia Banco em Java. Autor: Allas Maycon

No **main** é a classe principal do programa como o nome mesmo diz, apresentamos a mensagem de BEM-VINDO, pedimos para que o usuário insira os dados necessários e implementamos o **Loop de Interatividade e Controle de Fluxo** do sistema. Usamos a **Estrutura** do...while e o tratamento de **Entrada** case, o tratamento de **Exceção** usando o if(scanner.hasNextInt()), e o **Encerramento** do case declarado como (case 0).

3.3 MODELAGEM (ATRIBUTOS E ENCAPSULAMENTO)

Para garantir o **Código Limpo (Clean Code)** e a integridade dos dados, foi aplicado o conceito de **Encapsulamento**:

Atributos: `private String nome;` , `private String sobrenome;` , `private String cpf;` , `private double saldo;` . O uso do modificador `private` restringe o acesso direto, protegendo os dados.

Getters: Métodos públicos (`getNome()` , `getSaldo()` , etc.) foram criados para permitir a leitura controlada dos atributos.

Imutabilidade: Não foram criados métodos `setters` para os dados de identificação (`nome`, `sobrenome`, `cpf`), pois estes não devem ser alterados após o cadastro.

3.4 MÉTODOS DE OPERAÇÃO

Os procedimentos bancários foram implementados como métodos públicos:

- **depositar(double valor):** Adiciona o valor ao saldo, incluindo uma **validação** para garantir que o valor seja positivo.

- **sacar(double valor):** Diminui o valor do saldo, incluindo **validação** para verificar se há saldo suficiente e se o valor é positivo.

- **consultarSaldo():** Exibe de forma formatada as informações pessoais do cliente e o saldo atual.

4 LÓGICA DE CONTROLE DE FLUXO

Conexão com os **Sistemas Distribuídos**, o método estático `exibirMenu()` é responsável por imprimir as opções disponíveis na tela, tornando o código do main mais limpo e focado na lógica de decisão.

Estrutura de Repetição e Decisão (Requisito 5), a aplicação é controlada por uma combinação de estruturas:

- **do... while**: É a estrutura principal que garante que o menu seja exibido e executado repetidamente. O loop só é encerrado quando a opção '**0 – Sair**' é escolhida pelo usuário.

- **switch...case**: Utilizado para tratar a opção escolhida pelo usuário no menu. Cada **case**: direciona a execução para o método da classe **ContaBancaria** correspondente (ex: **case 2** chama **conta.Depositar()**).

- **Tratamento de Exceção Básica**: Foi incluída uma validação (**scanner.hasNextInt()**) para garantir que a entrada do menu seja um número, melhorando a robustez da aplicação.

5 CONCLUSÃO

O projeto **Gerenciador Banco** mostra os fundamentos da **Programação Orientada a Objetos (POO) em Java**. A implementação, embora seja simples, reforça o entendimento sobre:

- A correta **modelagem de classes**, para representar entidades do mundo real.
- A importância do **Encapsulamento**, para a segurança e manutenibilidade do código.
- O uso eficiente de estruturas de controle (**do...while** e **switch...case**) para criar interfaces de console interativas.

Este projeto serve como uma base sólida para o desenvolvimento de sistemas mais complexos, utilizando as boas práticas de organização de código e estruturação de projetos.

REFERÊNCIAS

MARTIN, Robert C. Livro - **Código Limpo**: habilidades práticas do Agile Software. Rio de Janeiro: Altas Books, 2019.

Deitel, Harvey M. DEITEL, Paul J.. Livro - **Java**: como programar. 10 ed. São Paulo: Pearson Education do Brasil, 2016.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar – **UML:Guia do usuário**: 2. Ed.. Rio de Janeiro: Campus, 2005.

ATIVIDADE DE PORTFÓLIO/INTERDISCIPLINAR

As atividades possuem material exclusivo do portfólio que se encontra no GitHub para download.

https://github.com/allas-amk/portfolio_linguagem_orientada_a_objetos_java