PROGRAMAÇÃO ORIENTADA A OBJETOS EM JAVA
Projeto final
Professor: JHONNATA NOVAES PIRES

Alan Perdomo; Eduardo Diniz; Jhonathan William
202212004637; 202002465591; 202202136956

Início

O programa inicia na classe Main, onde executa a função "*exibirMenu"* que é responsável por imprimir uma lista de opções para o usuário escolher qual ação deseja executar e continua com o comando switch/case para verificar a opção escolhida, através do scanner, e com base nisso continuar o fluxo de execução do programa.

Classe Main.java:

```java
import java.util.Scanner;

import Classes.Conta;
import Classes.Pessoas;

public class Main {
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {

        int opcao;
        int opcao2;

        while (true) {
            try {
                exibirMenu();
                String input = scanner.nextLine();
                opcao = Integer.parseInt(input);
                System.out.println("====================================");

                switch (opcao) {
                    case 0:
                        System.out.println("Programa encerrado.");
                        scanner.close();
                        return;
```

```
                case 1:
                    System.out.println("\nCADASTRAR PESSOA\n");
                    Pessoas.cadastrarPessoa(scanner);
                    break;
                case 2:
                    System.out.println("\nCRIAR CONTA\n");
                    Conta.criarConta(scanner);
                    break;
                case 3:
                    System.out.println("Escolha uma opção:\n");
                    System.out.println("1 - Listar Pessoas");
                    System.out.println("2 - Listar Contas");
                    System.out.print("\nOpção: ");

                    input = scanner.nextLine();
                    opcao2 = Integer.parseInt(input);
                    System.out.println("====================================");

                    switch (opcao2) {
                        case 1:
                            Pessoas.listarPessoas();
                            break;
                        case 2:
                            Conta.listarContas();
                            break;
                        default:
                            System.out.println("\nOpção inválida. Tente novamente.");
                            break;
                    }
                    break;
                case 4:
                    System.out.println("Escolha uma opção:\n");
                    System.out.println("1 - Deletar Pessoa");
                    System.out.println("2 - Deletar Conta");
                    System.out.print("\nOpção: ");

                    input = scanner.nextLine();
                    opcao2 = Integer.parseInt(input);
                    System.out.println("====================================");

                    switch (opcao2) {
                        case 1:
```

```
                            System.out.println("\nDELETAR PESSOA\n");
                            Pessoas.deletarPessoa(scanner);
                            break;
                        case 2:
                            System.out.println("\nDELETAR CONTA\n");
                            Conta.deletarConta(scanner);
                            break;
                        default:
                            System.out.println("\nOpção inválida. Tente novamente.");
                            break;
                    }
                    break;
                case 5:
                    System.out.println("\nDEPOSITO\n");
                    Conta.realizarDeposito(scanner);
                    break;
                case 6:
                    System.out.println("\nSAQUE\n");
                    Conta.realizarSaque(scanner);
                    break;
                default:
                    System.out.println("Opção inválida. Tente novamente.\n");
                    break;
            }
        } catch (NumberFormatException e) {
            System.out.println("Opção inválida. Tente novamente.");
        } catch (Exception e) {
            System.out.println("Ocorreu um erro. Tente novamente.");
        }
    }
}

public static void exibirMenu() {
    System.out.println("\n=====================");
    System.out.println("----- Bem-vindo -----");
    System.out.println("=====================\n");
    System.out.println("Escolha uma opção:\n");
    System.out.println("1 - Cadastrar Pessoa");
    System.out.println("2 - Criar Conta");
    System.out.println("3 - Listar Pessoas ou Contas");
    System.out.println("4 - Deletar Pessoa ou Conta");
    System.out.println("5 - Depositar");
```

```
        System.out.println("6 - Sacar");
        System.out.println("0 - Sair");
        System.out.print("\nOpção: ");
    }

}
```

Pessoas

Classe Pessoas.java:

```
package Classes;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;

import Data.DbContext;

public class Pessoas {
    // Atributos
    private String nome;
    private String cpf;

    // Construtor
    public Pessoas(String nome, String cpf) {
        this.nome = nome;
        this.cpf = cpf;
    }

    // Métodos
    public void imprimirDados() {
        System.out.println("Nome: " + nome);
        System.out.println("CPF: " + cpf);
    }

    public String getNome() {
        return nome;
    }

    public String getCpf() {
```

```
        return cpf;
    }

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

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

    // Função para cadastrar pessoa
    public static void cadastrarPessoa(Scanner scanner) {
        System.out.print("Digite o nome da pessoa: ");
        String nome = scanner.nextLine();

        if (!validarNome(nome)) {
            System.out.println("Nome inválido. O nome deve conter apenas letras e
espaços.");
            return;
        }

        System.out.print("Digite o CPF da pessoa: ");
        String cpf = scanner.nextLine();

        if (!validarCPF(cpf)) {
            System.out.println("CPF inválido. O CPF deve conter 11 dígitos numéricos.");
            return;
        }

        DbContext database = new DbContext();

        // Cadastra a pessoa no banco de dados se o cpf informado for unico
        try {
            database.conectarBanco();
            boolean pessoaExistente = verificarPessoaExistente(database, cpf);
            if (pessoaExistente) {
                System.out.println("CPF já cadastrado. Não é possível cadastrar a mesma
pessoa novamente.");
            } else {
                boolean statusQuery = database.executarUpdateSql(
```

```
                    "INSERT INTO public.pessoas(nome, cpf) VALUES ('" + nome + "', '"
+ cpf + "')");
                if (statusQuery) {
                    System.out.println("-------------------------------------");
                    System.out.println("'" + nome + "' foi cadastrado(a)!");
                    System.out.println("-------------------------------------");
                }
            }
            database.desconectarBanco();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Função para verificar se o CPF informado ja esta cadastrado no banco de dados
    public static boolean verificarPessoaExistente(DbContext database, String cpf) throws
SQLException {
        String query = "SELECT COUNT(*) FROM public.pessoas WHERE cpf = '" + cpf + "'";
        ResultSet resultSet = database.executarQuerySql(query);

        if (resultSet.next()) {
            int count = resultSet.getInt(1);
            return count > 0;
        }

        return false;
    }

    // Verifica se o nome informado contem apenas letras e espaços, alem de nao
    // poder ser um nove em branco
    public static boolean validarNome(String nome) {
        // Verifica se o nome contém apenas letras e espaços
        return nome.matches("[a-zA-Z\\s]+") && !nome.trim().isEmpty();
    }

    // Verifica se o cpf é valido
    public static boolean validarCPF(String cpf) {
        // Remove caracteres não numéricos do CPF
        cpf = cpf.replaceAll("\\D+", "");

        // Verifica se o CPF possui 11 dígitos numéricos
        return cpf.matches("\\d{11}");
```

```
    }

    // Função para deletar pessoa com base em um cpf informado
    public static void deletarPessoa(Scanner scanner) {
        System.out.print("Digite o CPF da pessoa a ser deletada: ");
        String cpf = scanner.nextLine();

        Pessoas pessoa = buscarPessoaPorCPF(cpf);
        if (pessoa == null) {
            System.out.println("CPF não encontrado. Tente novamente.");
            return;
        }

        System.out.println(
                "\nTodas as Contas da pessoa serão deletadas. \nDeseja realmente deletar a
pessoa com CPF " + cpf
                        + " ? (S/N): ");
        String confirmacao = scanner.nextLine().toUpperCase();
        if (!confirmacao.equals("S")) {
            System.out.println("Operação cancelada.");
            return;
        }

        String nome = pessoa.getNome();

        DbContext database = new DbContext();

        try {
            database.conectarBanco();

            // Deleta as contas associadas ao CPF
            boolean statusContaQuery = database
                    .executarUpdateSql("DELETE FROM public.contas WHERE cpf = '" + cpf +
"'");
            if (statusContaQuery) {
                System.out.println("\n Todas as contas associadas ao CPF " + cpf + " foram
deletadas.");
            }

            // Deleta a pessoa com o CPF especificado
            boolean statusPessoaQuery = database
```

```
                    .executarUpdateSql("DELETE FROM public.pessoas WHERE cpf = '" + cpf +
"'");
            if (statusPessoaQuery) {
                System.out.println("O usuário " + nome + " com CPF " + cpf + " foi
deletado.");
            }

            database.desconectarBanco();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Função para buscar a pessoa com base no CPF informado
    public static Pessoas buscarPessoaPorCPF(String cpf) {
        DbContext database = new DbContext();

        try {
            database.conectarBanco();

            ResultSet resultSet = database.executarQuerySql("SELECT * FROM public.pessoas
WHERE cpf = '" + cpf + "'");
            if (resultSet.next()) {
                String nome = resultSet.getString("nome");
                Pessoas pessoa = new Pessoas(nome, cpf);
                return pessoa;
            }

            database.desconectarBanco();
        } catch (Exception e) {
            e.printStackTrace();
        }

        return null;
    }

    // Função para listar todas as pessoas cadastradas no banco de dados
    public static void listarPessoas() {
        DbContext database = new DbContext();

        try {
            database.conectarBanco();
```

```
            ResultSet resultSet = database.executarQuerySql("SELECT * FROM
public.pessoas");

            if (!resultSet.next()) {
                System.out.println("\nNenhuma pessoa cadastrada.");
            } else {
                System.out.println("\nLista de Pessoas Cadastradas:\n");
                do {
                    String nome = resultSet.getString("nome");
                    String cpf = resultSet.getString("cpf");
                    System.out.println("-------------------------------------");
                    System.out.println("Nome: " + nome + " | CPF: " + cpf);
                } while (resultSet.next());
                System.out.println("=====================================\n");
            }

            database.desconectarBanco();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

Caso a opção escolhida pelo usuário esteja relacionada a cadastrar/deletar/listar pessoas, serão utilizados métodos da classe Pessoas.

Cadastrar Pessoa

Para cadastrar uma nova pessoa é utilizado o método “*cadastrarPessoa*” onde o usuário deve informar um nome e um número de CPF, ambos serão validados, para garantir que o nome tenha apenas letras e espaços e o CPF tenha 11 caracteres numéricos, para isso são utilizadas as funções “*validarNome*” e “*validarCPF*”, após isso é feita a conexão com o banco de dados e uma verificação se o CPF já foi cadastrado utilizando a função “*verificarPessoaExistente*”, caso já tenha sido cadastrado o usuário é informado e é cancelada a operação de cadastro; caso ainda não tenha sido o nome e o CPF são registrados no banco de dados e é enviada uma mensagem confirmando o cadastro para o usuário, e é feita a desconexão com o banco de dados.

Deletar Pessoa

Para deletar uma pessoa é utilizado o método "*deletarPessoa*" onde o usuário deve informar o CPF da pessoa que deseja deletar, com o número do CPF informado é feita uma busca no banco de dados utilizando a função "*buscarPessoaPorCPF"* e é necessário a confirmação de que o usuário deseja deletar a pessoa com o CPF, pois caso continue, será feita a conexão com o banco de dados e TODAS às contas vinculadas a este CPF serão deletadas.

Listar Pessoas

Para listar às pessoas cadastradas é utilizado o método "*listarPessoas"* que realiza a conexão com o banco de dados e imprime o nome e o CPF de todas as pessoas cadastradas, caso não tenha nenhuma pessoa cadastrada no banco de dados essa informação é enviada para o usuário.

Contas

Classe Conta.java:

```java
package Classes;

import java.sql.ResultSet;
import java.util.Random;
import java.util.Scanner;

import Data.DbContext;

public abstract class Conta {
    private int numeroConta;
    private String cpfPessoa;
    private double saldo;

    public Conta(int numeroConta, String cpfPessoa, double saldo) {
        this.numeroConta = numeroConta;
        this.cpfPessoa = cpfPessoa;
        this.saldo = saldo;
    }

    public int getNumeroConta() {
        return numeroConta;
    }
```

```
    public String getCpfPessoa() {
        return cpfPessoa;
    }

    public double getSaldo() {
        return saldo;
    }

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

    public void depositar(double valor) {
        saldo += valor;
    }

    public boolean sacar(double valor) {
        if (valor <= saldo) {
            saldo -= valor;
            return true;
        } else {
            return false;
        }
    }

    // Função para criar uma conta
    public static void criarConta(Scanner scanner) {
        System.out.print("Digite o CPF da pessoa: ");
        String cpf = scanner.nextLine();

        DbContext database = new DbContext();

        try {
            database.conectarBanco();
            System.out.println();
            // Verificar se o CPF está cadastrado no banco de dados
            boolean pessoaExistente = Pessoas.verificarPessoaExistente(database, cpf);
            if (!pessoaExistente) {
                System.out.println("CPF não encontrado. Cadastre a pessoa antes de criar a
conta.");
                return;
```

```
            }

            System.out.println("Escolha o tipo de conta:\n");
            System.out.println("1 - Conta Corrente");
            System.out.println("2 - Conta Poupança");
            System.out.print("\nOpção: ");
            int opcao;

            try {
                opcao = scanner.nextInt();
                scanner.nextLine(); // Limpar o buffer do scanner
            } catch (Exception e) {
                System.out.println("Opção inválida. A conta não foi criada.");
                return;
            }

            // Gerar um número de conta único
            String numeroConta = gerarNumeroContaUnico();

            boolean statusQuery;
            switch (opcao) {
                case 1:
                    statusQuery = database.executarUpdateSql(
                            "INSERT INTO public.contas(numeroconta, cpf, saldo, tipo)
VALUES ('" + numeroConta + "', '"
                                    + cpf + "', 0, 'Corrente')");
                    if (statusQuery) {
                        System.out.println("\n---------------------------------");
                        System.out.println("CONTA CORRENTE criada com sucesso!");
                        System.out.println("---------------------------------");
                        System.out.println("CPF do Titular da conta: " + cpf);
                        System.out.println("Numer da conta: " + numeroConta);
                        System.out.println("---------------------------------");
                    }
                    break;
                case 2:
                    statusQuery = database.executarUpdateSql(
                            "INSERT INTO public.contas(numeroconta, cpf, saldo, tipo)
VALUES ('" + numeroConta + "', '"
                                    + cpf + "', 0, 'Poupança')");
                    if (statusQuery) {
                        System.out.println("\n---------------------------------");
```

```
                    System.out.println("CONTA POUPANÇA criada com sucesso!");
                    System.out.println("---------------------------------");
                    System.out.println("CPF do Titular da conta: " + cpf);
                    System.out.println("Numer da conta: " + numeroConta);
                    System.out.println("---------------------------------");
                }
                break;
            default:
                System.out.println("Opção inválida. A conta não foi criada.");
                break;
        }

        database.desconectarBanco();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

// Função para deletar uma conta
public static void deletarConta(Scanner scanner) {
    System.out.print("Digite o número da conta a ser deletada: ");
    int numeroConta;

    try {
        numeroConta = scanner.nextInt();
        scanner.nextLine();
    } catch (Exception e) {
        System.out.println("Número de conta inválido. Tente novamente.");
        return;
    }

    Conta conta = buscarContaPorNumero(numeroConta);
    if (conta == null) {
        System.out.println("Conta não encontrada. Tente novamente.");
        return;
    }

    String cpf = conta.getCpfPessoa();
    Pessoas pessoa = Pessoas.buscarPessoaPorCPF(cpf);
    if (pessoa == null) {
        System.out.println("Dados da pessoa não encontrados. Tente novamente.");
        return;
```

```
        }

        String nome = pessoa.getNome();

        System.out.println("\n-------------------------------------");
        System.out.println("Nome: " + nome);
        System.out.println("CPF: " + cpf);
        System.out.println("Deseja realmente deletar a conta número " + numeroConta + "?
(S/N)");
        String confirmacao = scanner.nextLine().toUpperCase();
        if (!confirmacao.equals("S")) {
            System.out.println("Operação cancelada.");
            return;
        }

        DbContext database = new DbContext();

        try {
            database.conectarBanco();

            boolean statusQuery = database.executarUpdateSql(
                    "DELETE FROM public.contas WHERE numeroconta = " + numeroConta);
            if (statusQuery) {
                System.out.println("\n-------------------------------------");
                System.out.println("\nConta número " + numeroConta + " foi deletada.");
            }

            database.desconectarBanco();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Função para listar as contas
    public static void listarContas() {
        DbContext database = new DbContext();

        try {
            database.conectarBanco();

            ResultSet resultSet = database.executarQuerySql("SELECT * FROM public.contas
ORDER BY cpf");
```

```
            if (!resultSet.next()) {
                System.out.println("\nNenhuma conta cadastrada.");
                System.out.println("====================================");
            } else {
                System.out.println("\nLista de Contas Cadastradas:");

                String cpfAnterior = "";
                String nomeAnterior = "";

                do {
                    String numeroConta = resultSet.getString("numeroconta");
                    String cpfAtual = resultSet.getString("cpf");
                    String nomeAtual = "";

                    // Verifica o nome da pessoa no banco de dados com base no CPF
informado
                    ResultSet resultSetPessoa = database
                            .executarQuerySql("SELECT nome FROM public.pessoas WHERE cpf =
'" + cpfAtual + "'");
                    if (resultSetPessoa.next()) {
                        nomeAtual = resultSetPessoa.getString("nome");
                    }
                    resultSetPessoa.close();

                    if (!cpfAtual.equals(cpfAnterior)) {
                        cpfAnterior = cpfAtual;
                        nomeAnterior = nomeAtual;
                        System.out.println("\n====================================");
                        System.out.println("CPF: " + cpfAtual + " | Nome: " + nomeAtual);
                        System.out.println("====================================");
                    } else if (!nomeAtual.equals(nomeAnterior)) {
                        nomeAnterior = nomeAtual;
                        System.out.println("Nome: " + nomeAtual);
                    }

                    double saldo = resultSet.getDouble("saldo");
                    String tipoConta = resultSet.getString("tipo");

                    System.out.println("Número da Conta: " + numeroConta);
                    System.out.println("Saldo: R$ " + String.format("%.2f", saldo));
                    System.out.println("Tipo de Conta: " + tipoConta);
```

```
                    System.out.println("-----------------------------------");
                } while (resultSet.next());
            }

            resultSet.close();
            database.desconectarBanco();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Função para fazer deposito na conta
    public static void realizarDeposito(Scanner scanner) {
        System.out.print("Digite o número da conta: ");
        int numeroConta;

        try {
            numeroConta = scanner.nextInt();
            scanner.nextLine();
        } catch (Exception e) {
            System.out.println("Número de conta inválido. Tente novamente.");
            return;
        }

        DbContext database = new DbContext();

        try {
            database.conectarBanco();

            ResultSet resultSet = database
                    .executarQuerySql("SELECT * FROM public.contas WHERE numeroconta = '"
+ numeroConta + "'");

            if (!resultSet.next()) {
                System.out.println("Conta não encontrada. Tente novamente.");
                return;
            }

            // Busca as informações da conta com base no numero de conta informado
            String cpf = resultSet.getString("cpf");
            String tipoConta = resultSet.getString("tipo");
            double saldo = resultSet.getDouble("saldo");
```

```
            resultSet.close();

            // Verifica se é uma conta corrente ou conta poupança
            Conta conta;
            if (tipoConta.equalsIgnoreCase("Corrente")) {
                conta = new ContaCorrente(numeroConta, cpf, saldo);
            } else if (tipoConta.equalsIgnoreCase("Poupança")) {
                conta = new ContaPoupanca(numeroConta, cpf, saldo);
            } else {
                System.out.println("Tipo de conta inválido. Tente novamente.");
                return;
            }

            System.out.print("Digite o valor a ser depositado: R$");
            double valor;

            try {
                String inputValor = scanner.nextLine();

                // Verificar se o valor é numérico
                valor = Double.parseDouble(inputValor);

                if (valor < 0) {
                    System.out.println("O valor do depósito não pode ser negativo. Tente
novamente.");
                    return;
                }
            } catch (NumberFormatException e) {
                System.out.println("Valor inválido. Tente novamente.");
                return;
            }

            conta.depositar(valor);

            // Atualizar o saldo da conta no banco de dados
            boolean statusQuery = database.executarUpdateSql("UPDATE public.contas SET
saldo = " + conta.getSaldo()
                    + " WHERE numeroconta = '" + numeroConta + "'");
            if (statusQuery) {
                System.out.println("Depósito de R$" + valor + " realizado. Novo saldo: R$
"
```

```
                        + String.format("%.2f", conta.getSaldo()));
            }

            database.desconectarBanco();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Função para fazer saque na conta
    public static void realizarSaque(Scanner scanner) {
        System.out.print("Digite o número da conta: ");
        int numeroConta;
        // Veficifa a entrada do usuário
        try {
            numeroConta = scanner.nextInt();
            scanner.nextLine();
        } catch (Exception e) {
            System.out.println("Número de conta inválido. Tente novamente.");
            return;
        }

        DbContext database = new DbContext();

        try {
            database.conectarBanco();

            ResultSet resultSet = database
                    .executarQuerySql("SELECT * FROM public.contas WHERE numeroconta = '"
+ numeroConta + "'");

            if (!resultSet.next()) {
                System.out.println("\n====================================");
                System.out.println("Conta não encontrada. Tente novamente.");
                System.out.println("====================================\n");
                return;
            }

            String cpf = resultSet.getString("cpf");
            String tipoConta = resultSet.getString("tipo");
            double saldo = resultSet.getDouble("saldo");
```

```
            resultSet.close();

            Conta conta;
            if (tipoConta.equalsIgnoreCase("Corrente")) {
                conta = new ContaCorrente(numeroConta, cpf, saldo);
            } else if (tipoConta.equalsIgnoreCase("Poupança")) {
                conta = new ContaPoupanca(numeroConta, cpf, saldo);
            } else {
                System.out.println("Tipo de conta inválido. Tente novamente.");
                return;
            }

            System.out.print("Digite o valor a ser sacado: R$");
            double valor;

            try {
                String inputValor = scanner.nextLine();

                // Verificar se o valor é numérico
                valor = Double.parseDouble(inputValor);

                if (valor < 0) {
                    System.out.println("O valor do saque não pode ser negativo. Tente
novamente.");
                    return;
                }
            } catch (NumberFormatException e) {
                System.out.println("Valor inválido. Tente novamente.");
                return;
            }

            if (conta instanceof ContaCorrente) {
                ContaCorrente contaCorrente = (ContaCorrente) conta;
                if (!contaCorrente.sacar(valor)) {
                    System.out.println("Saldo insuficiente para realizar o saque.");
                    return;
                }
            } else {
                if (!conta.sacar(valor)) {
                    System.out.println("Saldo insuficiente para realizar o saque.");
                    return;
                }
```

```
            }

            // Atualizar o saldo da conta no banco de dados
            boolean statusQuery = database.executarUpdateSql("UPDATE public.contas SET
saldo = " + conta.getSaldo()
                    + " WHERE numeroconta = '" + numeroConta + "'");
            if (statusQuery) {
                System.out.println("\n====================================");
                System.out.println("Saque de R$" + valor + " realizado. Novo saldo: R$ "
                        + String.format("%.2f", conta.getSaldo()));
                System.out.println("====================================\n");
            }

            database.desconectarBanco();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Função para gerar numero de conta
    public static String gerarNumeroContaUnico() {
        // Gere um número de conta aleatório com 8 dígitos
        Random random = new Random();
        int numeroConta = random.nextInt(90000000) + 10000000;

        // Verifica se o número de conta já existe no banco de dados
        DbContext database = new DbContext();
        try {
            database.conectarBanco();
            ResultSet resultSet = database
                    .executarQuerySql("SELECT * FROM public.contas WHERE numeroconta = '"
+ numeroConta + "'");

            // Se o número de conta já existe no banco de dados um novo numero de conta é
            // gerado ate que seja um numero unico
            while (resultSet.next()) {
                numeroConta = random.nextInt(90000000) + 10000000;
                resultSet = database
                        .executarQuerySql("SELECT * FROM public.contas WHERE numeroconta =
'" + numeroConta + "'");
            }
```

```
            resultSet.close();

            database.desconectarBanco();
        } catch (Exception e) {
            e.printStackTrace();
        }

        return String.valueOf(numeroConta);
    }

    // Função para Buscar a conta com base no numer de conta informado
    public static Conta buscarContaPorNumero(int numeroConta) {
        DbContext database = new DbContext();

        try {
            database.conectarBanco();

            ResultSet resultSet = database.executarQuerySql(
                    "SELECT * FROM public.contas WHERE numeroconta = " + numeroConta);

            if (resultSet.next()) {
                String cpf = resultSet.getString("cpf");
                double saldo = resultSet.getDouble("saldo");
                String tipo = resultSet.getString("tipo");

                Conta conta;
                if (tipo.equals("Corrente")) {
                    conta = new ContaCorrente(numeroConta, cpf, saldo);
                } else if (tipo.equals("Poupança")) {
                    conta = new ContaPoupanca(numeroConta, cpf, saldo);
                } else {
                    return null;
                }

                database.desconectarBanco();
                return conta;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
```

```
}
```

Classe ContaCorrente.java:

```
package Classes;

public class ContaCorrente extends Conta {
    private static final double LIMITE_CHEQUE_ESPECIAL = 500.00;

    public ContaCorrente(int numeroConta, String cpfPessoa, double saldo) {
        super(numeroConta, cpfPessoa, saldo);
    }

    @Override
    public void depositar(double valor) {
        setSaldo(getSaldo() + valor);
    }

    @Override
    public boolean sacar(double valor) {
        double saldoTotal = getSaldo() + LIMITE_CHEQUE_ESPECIAL;
        if (saldoTotal >= valor) {
            setSaldo(getSaldo() - valor);
            System.out.printf("Saque de %.2f realizado. Novo saldo: R$%.2f\n", valor,
getSaldo());
            return true;
        } else {
            System.out.println("Saldo insuficiente para saque.");
            return false;
        }
    }
}
```

Classe ContaPoupanca.java:

```
package Classes;

public class ContaPoupanca extends Conta {
    private double taxaJuros;

    public ContaPoupanca(int numeroConta, String cpfPessoa, double saldo) {
        super(numeroConta, cpfPessoa, saldo);
```

```
        this.taxaJuros = 0.005; // 0.005% (0.005/100)
    }

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

    public double getTaxaJuros() {
        return taxaJuros;
    }

    @Override
    // Função para fazer deposito na conta poupança
    public void depositar(double valor) {
        super.depositar(valor);
        aplicarJuros();
    }

    // Função par aplicar o juros no momento do deposito
    private void aplicarJuros() {
        double juros = getSaldo() * taxaJuros;
        setSaldo(getSaldo() + juros);
    }
}
```

Caso a opção escolhida pelo usuário esteja relacionada a criar/deletar/listar contas, serão utilizados métodos da classe Conta, para saque/depósito serão utilizados métodos das classes ContaCorrente e ContaPoupanca relacionados aos respectivos tipos de conta em que será realizada a operação.

Criar Conta

Para criar uma conta é utilizado o método *"criarConta"* onde o usuário deve informar o CPF ao qual a conta estará vinculada, este CPF é verificado no banco de dados a partir da função *"verificarPessoaExistente"* presente na classe *Pessoas,* caso o CPF esteja cadastrado em Pessoas, o usuário deve informar o tipo de conta que deseja criar, Conta Corrente ou Conta Poupança, após escolher o tipo de conta, um número de conta único é gerado, a partir da função *"gerarNumeroContaUnico".* Com o CPF, tipo de conta e número da conta, é executado o comando SQL para cadastrar a nova conta no banco de dados.

Deletar Conta

Para deletar uma conta é utilizado o método *“deletarConta”* onde o usuário deve informar o número da conta que deseja deletar, com o número da conta é executada a função “buscarContaPorNumero” que recebe as informações da conta e com o CPF obtém as informações da pessoa vinculada, utilizando os metodos *“getCpfPessoa”* da classe conta e *“buscaPessoPorCPF”* da classe Pessoas. Os Nome e Cpf da pessoa vinculada a conta informada exibido e é necessária a confirmação do usuário para continuar o processo de deletar a conta, caso confirmado, o banco de dados é acessado e a conta é deletada.

Listar Contas

Para listar contas é utilizado o método *“listarContas”* que acessa o banco de dados, seleciona todas as contas e imprime todas as contas de forma agrupada por pessoas vinculadas, exibindo o número da conta, o tipo de conta e o saldo de cada conta.

Depositar

Para realizar depósitos é utilizado o método *“realizarDeposito”* onde o usuário deve informar o número da conta em que deseja realizar o depósito, o número da conta é verificado no banco de dados e o CPF, saldo e tipo de conta são obtidos. O usuário deve informar o valor que deseja depositar, esse valor é verificado para não ser um número negativo, e com base no tipo de conta o depósito é feito ou seguindo o método *“depositar”* da classe ContaCorrente, onde é utilizado o método *“getSaldo”* e o valor informado é somado ao saldo, de forma simples, ou o método “depositar” da classe ContaPoupanca que também soma o valor informado ao saldo anterior mas utiliza a função *“aplicarJuros”* que adiciona uma taxa de 0.005% ao valor do saldo. Os dados são salvos no banco de dados e é informado a confirmação do depósito.

Sacar

Para realizar um saque é utilizado o método *“realizarSaque”* onde o usuário deve informar o número da conta que deseja realizar o saque, o número informado é verificado e os dados das contas são obtidos pelo programa. O usuário deve informar o valor do saque, e este valor é verificado para que não seja um valor nulo ou negativo. O tipo de conta é definido e o método *“sacar”* é usado, no caso de conta poupança o *valor do saldo* - *valor do saque* não pode ser **negativo**, e no caso de conta corrente o *valor do saldo* + *Limite do cheque especial* - valor do saque não pode ser **negativo**, pois a conta corrente permite que o saldo chegue a -R$500. Caso seja possível realizar o saque, o saldo é atualizado no banco de dados é informada a confirmação do saque para o usuário e a conexão com o banco de dados é interrompida, voltando para o menu inicial.