



UNIVERSIDADE FEDERAL DE VIÇOSA - *CAMPUS FLORESTAL*

Gabriel Vitor da Fonseca Miranda - mat.3857

Pedro Augusto Maia Silva - mat.378

Trabalho Prático
Programação Orientada a Objetos
Pair Bank

Trabalho prático da disciplina
Programação Orientada a Objetos CCF -
313, do curso de Ciência da Computação
da Universidade Federal de Viçosa -
Campus Florestal.

Professor: Fabrício Silva

Florestal

2022

SUMÁRIO

1. Introdução.....	3
2. Desenvolvimento.....	3
3. Implementação.....	3
4. Encapsulamento.....	3
4.1 Modelo.....	4
4.2 Visao.....	4
4.3 Controle.....	4
4.4 Persistência Dao.....	5
5. Herança.....	6
5.1 Classe Pessoa.....	6
6. Polimorfismo.....	7
5.6 Classe Conta.....	7
7. Execução.....	8
8. Conclusão.....	8
8.1 Resultados.....	8
8.2 Considerações finais.....	13
5. Referências.....	14

1. Introdução

Neste projeto de sistema de banco "Pair Bank", foi utilizada uma classe do tipo usuário, onde este usuário teria informações úteis para contas comumente usadas em banco, aliado a isso, este usuário usaria seu CPF e sua senha para fazer login no sistema. É lícito ressaltar que terão usuários que além de possuírem uma conta como cliente teriam também acesso administrativo. Dessa forma, o usuário teria alguns métodos como fazer: pix, transferência, depositar, pagar, sacar, solicitar empréstimo. Portanto, o sistema de banco, terá como objetivo fornecer as diversas funcionalidades que um banco possui para melhor compreensão de como estas operações ocorrem em bancos reais.

2. Desenvolvimento

Para a implementação do trabalho proposto, foi utilizada a linguagem de programação Java, juntamente com a ferramenta Git para gerenciar o controle das versões do código. Além disso, com intuito de manter a organização do código e facilitar o processo de análise dos algoritmos, foram criadas várias classe que ficaram muito bem encapsuladas. Neste projeto foi usado o modelo MVC para um melhor encapsulamento.

3. Implementação

A seguir será apresentado e explicado detalhadamente o funcionamento de cada uma das principais funcionalidades implementadas no trabalho prático. Para melhor visualização, estes foram organizados em subtópicos.

4. Encapsulamento

O encapsulamento é um conceito da Programação Orientada a Objetos onde o estado de objetos (as variáveis da classe) e seus comportamentos (os métodos da classe) são agrupados em conjuntos segundo o seu grau de relação.

Assim sendo, o propósito do encapsulamento é o de organizar os dados que sejam relacionados, agrupando-os (encapsulando-os) em objetos (classes), reduzindo as colisões de nomes de variáveis (dado que variáveis com o mesmo nome estarão em namespaces distintos) e, da mesma forma, reunindo métodos relacionados às suas propriedades (ou variáveis de classe). Este padrão ajuda a manter um programa com centenas ou milhares de linhas de código mais legível e fácil de trabalhar e manter.

Neste trabalho prático foi usado o modelo MVC (modelo, visão e controle) para se fazer a implementação do sistema de banco.

4.1 Modelo

No modelo foram criadas as seguintes classes: Agência, Cliente, Conta, ContaCorrente, ContaPoupança, ExtratoCliente, Extrato, Gerente, Pessoa. Estas classes que estão bem definidas e encapsuladas foram usadas para a formação do sistema de banco “PairBank”.

4.2 Visao

O package “visão” possui as classes: ‘VisaoCliente’, ‘VisaoContaCorrente’, ‘VisaoGerente’, ‘VisaoPessoa’, ‘VisaoTela’. Todas essas classes vão administrar as funções do controle e do banco de dados, de forma a mostrar o funcionamento e execução do algoritmo.

```
=====
1 - Cadastro
2 - Login Cliente
3 - Login Gerente
4 - Sair
=====
Opcao:
```

Figura 1. Menu inicial no terminal.

Este modo de execução é apenas demonstrativo, visto que foi implementado uma interface gráfica para a execução do algoritmo. Este modo tem como função a execução de testes e usado como parâmetro para a construção da interface final.

4.3 Controle

O package “controle” possui as seguintes classes: ControleAgência, ControleCadastro, ControleCliente, ControleContaCorrente, ControleContaPoupança, ControleExtrato, ControleGerente, ControlePessoa. Esses controles têm como funcionalidade administrar todas as funcionalidades do código e determinar se as entradas são válidas ou não. Para exemplificar, temos o seguinte controle:

```
public class ControleCadastro {

    private static final Logger LOGGER = Logger.getLogger("controleCadastro");
    private popUPmensagem mensagem;
    public ControleCadastro() { mensagem = new popUPmensagem(); }
}
```

Figura 2. ControleCadastro.

```
public boolean dataEhValida(int dia,int mes,int ano){
    if(dia > 31 || dia <= 0){
        return false;
    }
    else {
        if (mes > 12 || mes < 0) {
            return false;
        } else {
            Date dataHoraAtual = new Date();
            String anoAtual = new SimpleDateFormat( pattern: "yyyy").format(dataHoraAtual);
            return ano < Integer.parseInt(anoAtual) && ano > 0;
        }
    }
}

public boolean oValorEhInteiro(String valor){
    try {
        Integer.parseInt(valor);
        return true;
    }
    catch (Exception error){
        return false;
    }
}
```

Figura 3. dataEhValida.

O controle em questão, tem como funcionalidade determinar se a data passada é válida, se o valor inserido é um valor inteiro e também é responsável por determinar se uma senha e um CPF são válidos. Dessa forma, fica a cargo do modelo apenas estruturar os dados, que serão inseridos após passados pelo controle.

4.4 Persistência Dao

Neste trabalho prático também foi possível armazenar as informações que são usadas de entradas pelos usuários via banco de dados. O banco de dados usado foi o MySQL Server. Para se fazer a conexão com o banco de dados primeiro criamos o banco de dados no

MySQL, cada uma das classes foi representada por uma tabela para assim poder recuperar informações mesmo depois que o usuário encerrar o programa.

Para encapsular melhor cada requisição ao banco de dados criamos as seguinte classes no pacote “Dao”: AgenciaDao, ClienteDao, ContaCorrenteDao, ContaPoupançaDao, ExtratoClienteDao, ExtratoDao e GerenteDao. Cada uma dessas classe faz conexão ao banco de dados, todos os dados que são atualizados, informados ou retirados pelo o usuário são atualizados pelo banco de dados.

Outrossim, voltando para o assunto da conexão foi criada um classe que faz essa conexão com o banco de dados, então todas as classe que faram requisições ao banco de dados recebem a classe conexão que é inicializada nos seus construtores.

5. Herança

Herança é um princípio usado em programação para reaproveitar estruturas e códigos já existentes com intuito de evitar repetição de funções e estruturas já construídas. Dessa forma, permite que em linguagens orientadas a objetos, classes já pré-existentes, ou construídas no projetos, sejam possíveis de se reutilizar em um mesmo projeto sem reescrever seu código .

5.1 Classe Pessoa

No projeto em questão foi construída a classe pessoa que é responsável por armazenar informações que uma pessoa comumente retém ao se cadastrar em um banco. Dessa forma, essa classe pessoa possuindo os atributos de “nome, cpf, dataNascimento, salario” , permite que as classes “cliente” e “gerente” herdem a classe “Pessoa” e tenham as mesmas características da classe herdada, podendo assim realizar operações que estão atreladas a classe pessoa, como podemos ver a seguir na Figura 4:

```

public double getSalario() { return salario; }
public void setNome(String nome){this.nome = nome;}
public int getDiaNascimento(){return this.dataNascimento.getDia();}
public int getMesNascimento(){return this.dataNascimento.getMes();}
public int getAnoNascimento(){return this.dataNascimento.getAno();}
public int getIdade(){
    Calendar cal = Calendar.getInstance();
    int anoAtual = cal.get(Calendar.YEAR);
    int mesAtual = cal.get(Calendar.MONTH);
    int diaAtual = cal.get(Calendar.DAY_OF_MONTH);
    if((mesAtual == this.dataNascimento.getMes() && diaAtual >= this.dataNascimento.getDia()) ||
        return anoAtual - this.dataNascimento.getAno();
    }
    else {
        return anoAtual - this.dataNascimento.getAno() - 1;
    }
}
public String toString(){
    return "\nNome:"+getNome()+"\n"+
        "\nIdade:"+getIdade()+"\n"+
        "\nCPF:"+getCpf()+"\n"+
        "\nData de Nascimento:"+getDataNascimento()+"\n";
}

```

Figura 4. Classe Pessoa.

```

public class Gerente extends Pessoa

```

Figura 5. Classe Gerente herdando de Pessoa.

```

public class Cliente extends Pessoa

```

Figura 6. Classe Cliente herdando de Pessoa.

6. Polimorfismo

Apesar de aplicar todos os conceitos de orientação a objetos, neste projeto de banco foi utilizado o polimorfismo. Neste caso foi usado a Sobreposição de métodos (override) que é um conceito do polimorfismo que nos permite reescrever um método, ou seja, podemos reescrever nas classes filhas métodos criados inicialmente na classe pai, os métodos que serão sobrepostos, diferentemente dos sobrecarregados, devem possuir o mesmo nome, tipo de retorno e quantidade de parâmetros do método inicial, porém o mesmo será implementado com especificações da classe atual, podendo adicionar um algo a mais ou não.

6.1 Classe Conta

A sobreposição de métodos foi usada nas classes filhas de *Conta*, no caso *ContaCorrente* e *ContaPoupança*, já que as mesmas fazem operações semelhantes que são,

sacar, depositar, transferir. Dessa forma, foi criada uma classe do tipo abstrato *Conta*, e métodos abstratos nesta classe. Diante disso, as classe *ContaCorrente* e *ContaPoupanca* herdam esses métodos de *Conta* e fazem uma sobreposição de métodos.

7. Execução

Para se fazer a execução deste programa você precisa ter o MySQL Server, isto para fazer a conexão com o banco de dados. É importante ressaltar que o banco de dados deste projeto está anexando no repositório do git, na pasta ScriptSQL.

Foi usada a ferramenta MySQL Workbench para as inserções das tabelas e operações que seriam necessárias no projeto, isto no pacote Dao. Depois que as tabelas forem criadas na sua ferramenta, basta ir no código e ir no pacote conexão e alterar a senha da conexão para a sua senha de usuário do MySQL Workbench: Na figura abaixo onde está em preto você deve colocar a sua senha.

```
public class Conexao {
    Statement statement;
    Connection connection;
    public Connection getConnection() {
        try {
            connection = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/PairBank", user: "root", password: ██████████ );
            return connection;
        } catch (Exception e) {
            System.err.println("Falha ao conectar ao banco de dados!!"+e.getMessage());
            return null;
        }
    }
}
```

Figura 7. Conexão.

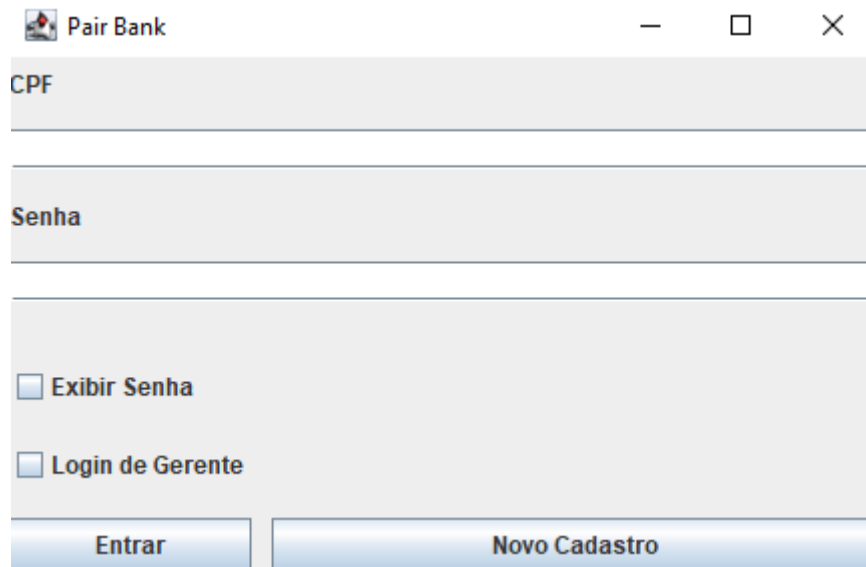
Quando colocar a senha certa basta executar o script SQL que se encontra no repositório do git, e usar o programa.

8. Conclusão

A seguir serão apresentados, respectivamente, os resultados encontrados a partir da execução das funções descritas na aba Implementação e as considerações finais a respeito deste trabalho prático.

8.1 Resultados

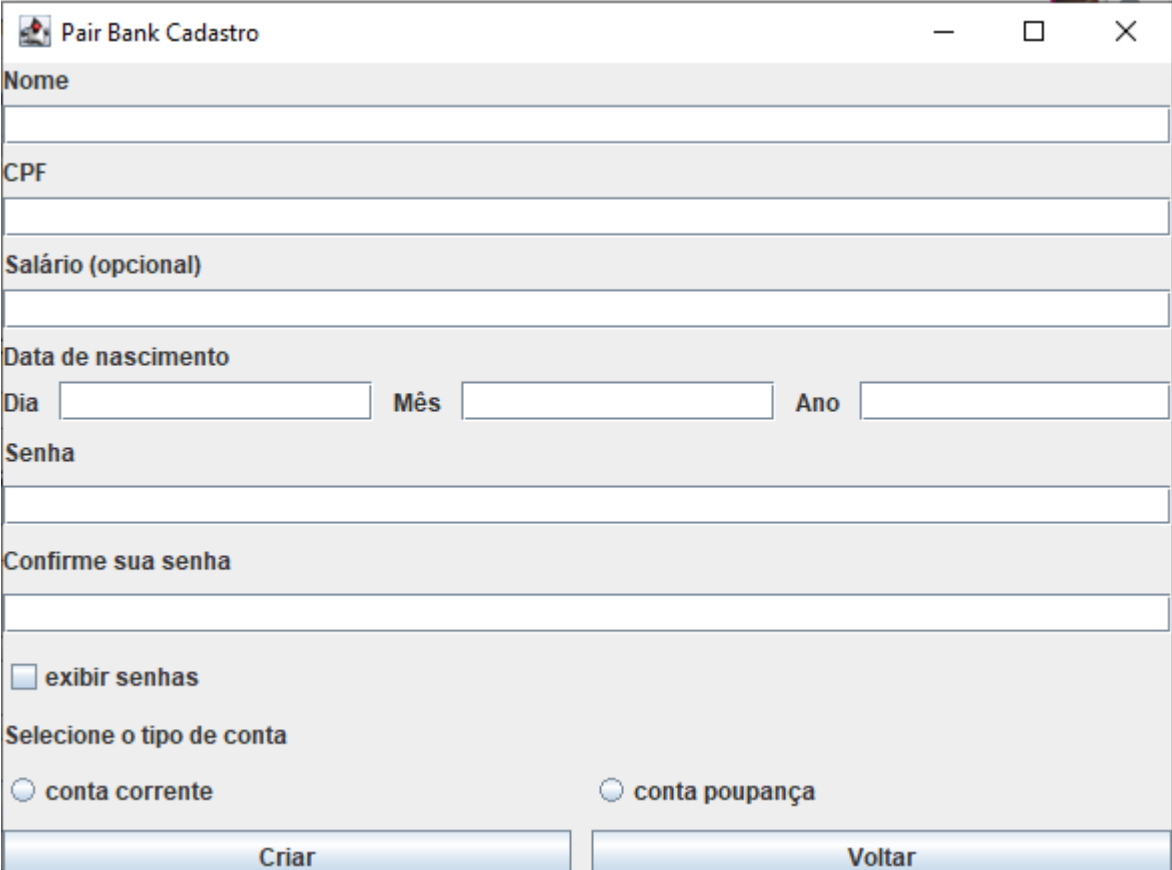
Como podemos observar, a interface implementada permite que um usuário, seja cliente ou gerente, faça login e consiga realizar suas operações bancárias. É exigido apenas o CPF e a senha do usuário para que o login seja efetuado.



The image shows a window titled "Pair Bank" with standard window controls (minimize, maximize, close). The interface contains two text input fields: the first is labeled "CPF" and the second is labeled "Senha". Below the "Senha" field, there are two checkboxes: "Exibir Senha" and "Login de Gerente". At the bottom of the window, there are two buttons: "Entrar" on the left and "Novo Cadastro" on the right.

Figura 8. Janela inicial da interface.

Ao clicar no botão “Novo Cadastro”, o usuário tem a possibilidade de criar uma conta, caso o mesmo não possua cadastro no Banco Pair Bank. Vale ressaltar que o cadastro depende do CPF do usuário, dessa forma, não permitindo criar duas contas bancárias com mesmo CPF.



Pair Bank Cadastro

Nome

CPF

Salário (opcional)

Data de nascimento

Dia Mês Ano

Senha

Confirme sua senha

☐ exibir senhas

Selecione o tipo de conta

☐ conta corrente ☐ conta poupança

Criar Voltar

Figura 9. Tela de cadastro.

Na tela de cadastro o usuário tem como opção digitar o salário para posteriormente obter um empréstimo, selecionar se sua conta vai ser uma conta poupança ou uma conta corrente.

Terminado o cadastro e os dados forem válidos, o usuário tem a possibilidade de efetuar o login e começar a usar os serviços do banco.



Figura 10. Menu Cliente.

Como podemos ver na tela acima, o usuário tem como opção ver o saldo (a string do saldo aparece na tela como na imagem acima), tem também a possibilidade de realizar transferências, fazer pix, alterar as configurações da conta e ver o seu extrato que são as principais funções do banco.



Extrato		
Descrição	Data	Valor
Saque	02/03/2022	50
Deposito	02/03/2022	20
Deposito	23/03/2022	200.0
Saque	23/03/2022	200.0
Pix	23/03/2022	200.0
Deposito	23/03/2022	200.0
Voltar		

Figura 11. Extrato.

Como podemos ver na imagem acima, o usuário tem como analisar todas as movimentações realizadas em sua conta, bem como a data de tal movimentação.



Figura 12. Menu Gerente.

Na Figura 12, podemos analisar as funcionalidades existentes para um gerente, o qual tem a possibilidade de ver todas as agências existentes, verificar o saldo de um cliente e listar todos os clientes da agência para a qual ele gerencia.

8.2 Considerações finais

Neste trabalho prático foi explorado um tema de grande importância para os estudos, principalmente na área da Ciência da Computação que foi o estudo sobre Programação Orientada a Objetos. Programação orientada a objetos (POO, ou OOP segundo as suas siglas em inglês) é um paradigma de programação baseado no conceito de "objetos", que podem conter dados na forma de campos, também conhecidos como atributos, e códigos, na forma de procedimentos, também conhecidos como métodos. Uma característica de objetos é que um procedimento de objeto pode acessar, e geralmente modificar, os campos de dados do objeto com o qual eles estão associados (objetos possuem uma noção de "this" (este) ou "self" (próprio)).

Em POO, programas de computadores são projetados por meio da composição de objetos que interagem com outros. Há uma diversidade significativa de linguagens de POO, mas as mais populares são aquelas baseadas em classes, significando que objetos são instâncias de classes, que, normalmente, também determinam seu tipo.

Outrossim, o objetivo principal deste trabalho foi criar um sistema para banco com tudo o que foi aprendido durante as aulas de Programação Orientada a Objetos, isto utilizando conceitos de encapsulamento, herança e polimorfismo. Ademais também foram utilizados conceitos de banco de dados, que foram aplicados no sistema de banco.

Portanto, esta abordagem mais prática da implementação de um programa usando os conceitos de POO, melhorou ainda mais os conhecimentos adquiridos em sala de aula, pois fez com que conceitos dos exemplos de POO que foram apresentados em sala fossem utilizados para a implementação dessa estrutura o “PairBank”.

9. Referências

- [1] Encapsulamento em Java: Primeiros passos, DEVMEDIA. 2014. Disponível em: [Link](#). Acesso em: 20 de Março de 2022.
- [2] Sobrecarga e sobreposição da métodos em orientação a objetos. DEVMEDIA. 2015. Disponível em: [Link](#). Acesso em: 20 de Março de 2022.
- [3] Programação orientada a objetos. Wikipedia, 2021. Disponível em: [Link](#). Acesso em: 21 de Março de 2022.
- [4] IntelliJ IDEA 2021: How to use GUI Designer. BoostMyTool. 2021. Disponível em: [Link](#). Acesso em: 14 de Março de 2022.
- [5] Conceitos e Exemplos - Herança: Programação Orientada a Objetos -Parte 1. DEVMEDIA. 2020. Disponível em: [Link](#). Acesso em: 23 de Março de 2022.