Trabalho Prático 1: Banco UNESP

Disciplina: Programação Orientada a Objetos

{*Entregar até o dia 17/04}

Prof. Dr. Lucas C. Ribas

Sumário

1 Banco UNESP

Neste projeto, você será responsável por desenvolver um sistema de banco em Java que permita a criação e manipulação de diversas classes, incluindo DisplayBanco, Banco, Agência e Conta.

O sistema de banco será composto de uma classe Banco que possuirá várias classes Agência. Por sua vez, cada objeto da classe Agencia possuirá várias classes Conta associadas. As Contas serão usadas para armazenar informações sobre os clientes do banco, incluindo seu saldo, nome, endereço, número de conta, CPF e outras informações relevantes. Para implementar este projeto, você precisará desenvolver, **no mínimo**, as classes DisplayConta, Agencia, Banco e Contas em Java.

Para desenvolver esse trabalho, você deve usar os diversos conceitos visto em aula tanto de Orientação a Objetos como Java.

E o deadline? o que eu posso e não posso fazer? e o meu pagamento? vai lá na Seção 5 que eu te explico.

2 Descrição

O programa deve conter no **mínimo** as classes listada no diagrama UML, que por sua vez devem conter os campos e métodos descritos em cada uma delas. Outros métodos, campos e classes podem ser implementados, bem como outras funcionalidades.

Inicialmente, o método `main` do seu programa deve setar 3 String com os caminhos dos arquivos `banco.txt`, `agencias.txt` e `contas.txt`. Esses arquivos irão conter as seguintes informações:

- `banco.txt`: uma única linha com os campos da classe Banco.
- `agencias.txt`: várias linhas, em que cada linha representa as informações de uma agência.

- `contas.txt`: várias linhas em que cada linha representa uma conta do banco.

Para mais informações sobre como acessar e usar essses arquivos, leia a Seção 4.

Após isso, na `main`, o seu código deve criar um objeto do tipo `DisplayBanco` passando para o construtor as 3 string com os caminhos dos arquivos de dados.

- A classe `DisplayBanco` deve ter um único construtor (ou seja, só pode ser instanciada passando as Strings com os caminhos dos arquivos de dados) e um único método público chamado `login`. O construtor da classe `DisplayBanco` irá instanciar o objeto do tipo `Banco` usando as informações no arquivo (`banco.txt`) e atribuí-lo ao campo privado `meuBanco`. Depois, ainda no construtor, irá criar as agências em `meuBanco` e por último criar as contas de cada agência. Tudo isso usando as informações contidas nos arquivos de dados que vocês receberam.

- Após criar um objeto da classe `DisplayBanco` na função principal (`main`), é necessário invocar o método `login` para que o usuário possa entrar no sistema bancário. O método `login` tem como objetivo mostrar uma mensagem de boas-vindas na tela do usuário, solicitar o número da agência, o número da conta e a senha do usuário e, em seguida, verificar se as informações fornecidas estão corretas. Para realizar essa verificação, o método `login` utiliza um método da classe `Banco` chamado `logarCliente`, que recebe como argumentos o número da agência, o número da conta e a senha. Esse método realiza uma busca nos registros do banco para encontrar a agência e a conta correspondentes aos números informados e, em seguida, verifica se a senha está correta. Se as informações estiverem corretas, o método `logarCliente` atribui à variável `contaLogada` na classe `Banco` a referência do objeto da conta logada (essa referência será retornada pelo método `buscarConta` da instância da `Agencia`). Essa variável é utilizada em outras operações bancárias para acessar diretamente a conta que está sendo manipulada.

- Caso os dados informados estejam incorretas (por exemplo, não exista a conta ou senha incorreta) uma mensagem deve ser mostrada na tela informado o erro e, em seguida, disponibilizado novamente a leitura das informações para o usuário. Caso estejam corretos, um método privado chamado `telaUsuario` será invocado a partir do método `login`.

- `telaUsuario` irá mostrar para o usuário as opções de operações que podem ser realizada usando sua conta: **1- Consultar Saldo; 2- Depósito; 3- Saque; 4- Transferência; 5- Pix (chave será o CPF) 6- Sair.**

- Para cada uma das opções acima terá um método privado dentro de `DisplayBanco` que irá chamar os métodos da classe Banco para realizar as operações e também mostrar para o usuário os retornos. Esses métodos estão no diagrama UML.

- Quando o usuário selecionar a opção **Sair**, o método `operacaoSair` será invocado a partir de `telaUsuario`. Esse método será responsável por chamar o método `deslogarConta` do objeto em `meuBanco`. O método `deslogarConta` dentro da classe `Banco` irá setar como `null` o campo `contaLogada`, ou seja, a refência será perdida. Após isso, os outros métodos da classe Banco não conseguirão mais acessar saldo, transferências e demais operações através do campo `contaLogada`, deixando de funcionar.

- Uma vez selecionada a opção Sair, a tela será limpa e o usuário terá novamente a tela inicial de login que é mostrada pelo método `login`.

3 Diagrama UML

O diagrama UML apresentado na Figura 1 descreve as classes, atributos e métodos do sistema bancário. Embora as classes apresentadas sejam obrigatórias, outras classes podem ser criadas conforme necessário. Da mesma forma, embora os métodos e atributos listados sejam obrigatórios, pode ser necessário adicionar

outros (será!). Vale ressaltar que os métodos de atributos `get` e `set` foram omitidos por simplicidade, mas devem ser implementados sempre que necessário.

A Figura representa um sistema bancário, em que um usuário pode fazer login em sua conta através de uma interface, o DisplayBanco (classe), e realizar operações bancárias, como saques, depósitos, transferências, entre outros.

A classe Banco é a classe principal deste sistema, que possui atributos como número, nome, CNPJ, endereço, lista de agências e a conta atualmente logada. Possui métodos para realizar o login de um cliente, realizar saques, verificar o saldo, realizar depósitos, cadastrar agências e contas, buscar uma agência, realizar transferências e PIX, e deslogar uma conta. Já a classe Agência é composta pelo código, nome, endereço e lista de contas. Tem métodos para cadastrar uma nova conta e buscar uma conta existente.

Na classe Conta os atributos são: número, saldo, nome, endereço, CPF, data de nascimento e senha. Tem métodos para depositar, sacar, comparar e alterar a senha da conta. Por fim, a classe DisplayBanco é a interface do usuário que permite realizar as operações bancárias. Possui atributos como meuBanco (que é uma instância da classe Banco) e métodos para login, tela de usuário, operações de depósito, saque, PIX, transferência, saldo e sair.

As associações entre as classes indicam a relação entre elas. Banco possui uma lista de agências, Agência possui uma lista de contas e DisplayBanco é associada a um único objeto Banco. As multiplicidades indicam o número mínimo e máximo de instâncias que podem ser associadas, onde '0..*' representa "zero ou muitos"e 1 representa "apenas um".

4 Banco de Dados

Após implementar todas as classes e funcionalidades, você poderá testar o seu programa usando os dados fornecidos em 3 arquivos:

- `banco.txt`: esse arquivo possui 1 linha com os campos do banco, nessa ordem, separados pelo caractere "`|`": Nome, Numero, CNPJ, Endereço.
- `agencias.txt`: esse arquivo possui 5 linhas com os campos de cada agência (total de 5), nessa ordem, separados pelo caractere "`|`": Nome, Código ou Número (0 a 4), Endereço.
- `contas.txt`: esse arquivo possui 300 contas, onde cada linha representa uma conta com os seguintes campos, nessa ordem, separados pelo caractere "`|`": Nome, Data Nascimento, Endereço, CPF, Saldo, Número da agência, Número da Conta, senha.

Note que é possível ligar as contas com as agências por meio do número/código da agência que está presente em ambos os arquivos. Por exemplo, olhando para os dados da Ana Sophia, é possível ver que sua conta pertence à agência 2.

```
1 Ana Sophia Fernandes|25/01/1991|Esplanada Thiago Rezende, 60,Nunes
    Grande-BA|71920164383|39538.95|2|62475|231439
```

Para ler os arquivos de dados, use o seguinte o código abaixo. Note que, na linha 18 do código, a linha que foi lida do arquivo é quebrada em campos usando "#"como o separador. Portanto, o vetor de String `campos` terá todos os campos nas suas posições, começando pelo nome na posição 0 até a senha na última posição. Outra observação é que os campos que são numéricos precisam ser convertidos de String para double (ou int e long). Para isso, é só usar os métodos `valueof` das classes dos tipos primitivos, por exemplo: `int valorConvertido = Integer.valueOf(str);`

```
1
2 // Cria um objeto File com o caminho do arquivo
3 File arquivo = new File("/.../contas.txt");
```

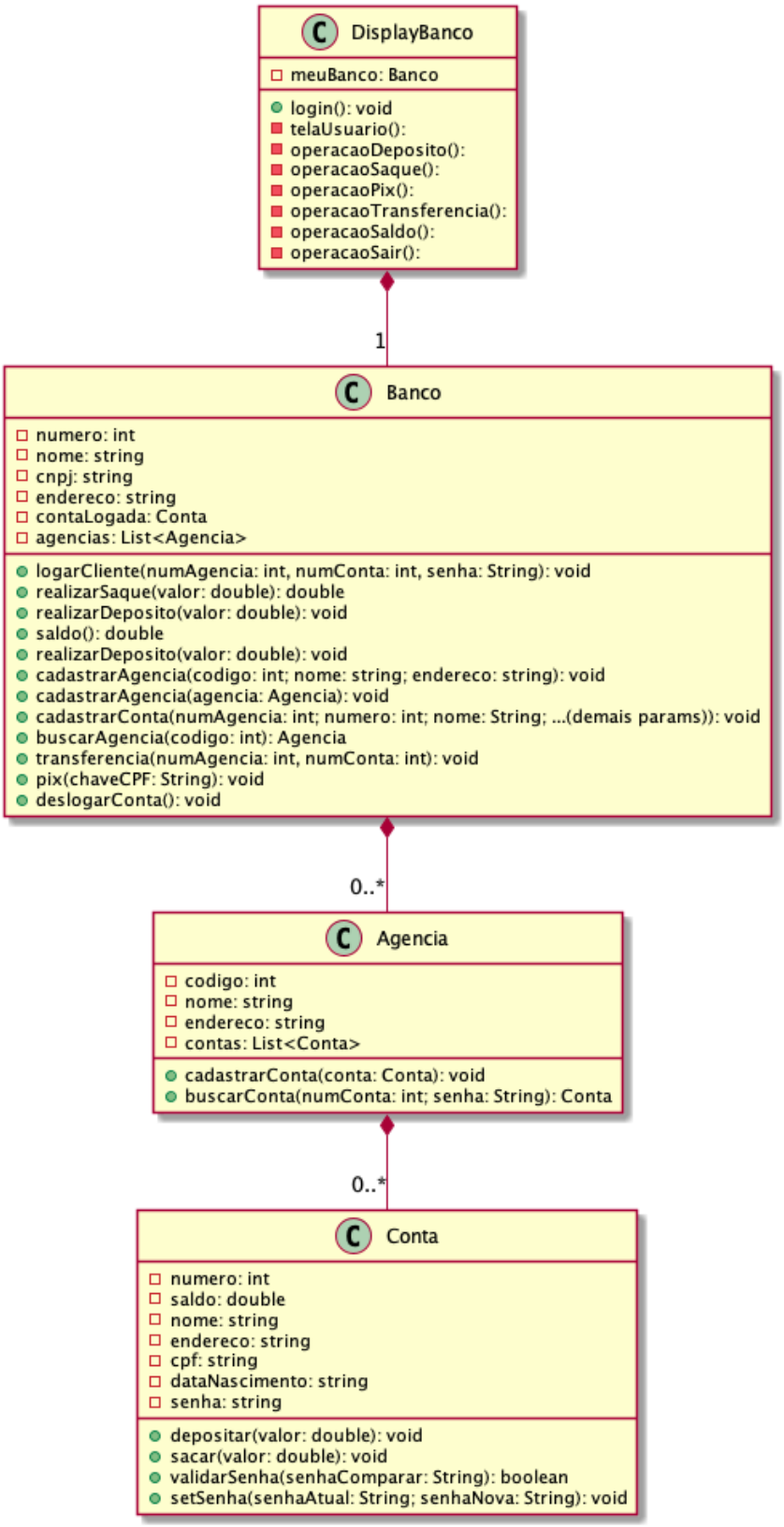

Figura 1: Diagrama UML do projeto de sistema do banco UNESP.

```
4
5  // Cria um objeto Scanner para ler o arquivo
6  Scanner scanner = null;
7  try {
8      scanner = new Scanner(arquivo);
9  } catch (FileNotFoundException ex) {
10     Logger.getLogger(Disciplinapoo.class.getName()).log(Level.SEVERE, null,
           ex);
11 }
12
13 // Lê cada linha do arquivo
14 while (scanner.hasNextLine()) {
15     String linha = scanner.nextLine();
16
17     // Divide a linha em campos separados por #
18     String[] campos = linha.split( "#");
19
20     // Imprime o conteúdo de cada campo (pode usar para testar se a leitura
           está correta, nesse caso ele imprime com | como separador)
21     for (String campo : campos) {
22         System.out.print(campo + " | ");
23     }
24     // Pula uma linha
25     System.out.println();
26 }
```

5 Avaliação

- Este trabalho é **individual**. Isso significa que qualquer tipo de plágio será penalizado com nota **ZERO**.
- Posteriormente à entrega do trabalho, poderei solicitar que alguns de vocês expliquem o código. Essa entrevista também poderá influenciar na nota final.
- Seu pagamento será em moedas do tipo Nota :). A nota atribuída ao trabalho será de 0 a 10 e fará parte da média de trabalhos realizados ao longo do semestre. O peso dessa média foi definido no início da disciplina.
- Implementações extras, como a classe Extrato, e a utilização de boas práticas de programação, como documentação JavaDoc, contarão como ponto extra.
- A entrega deve ser feita via classroom. Você deverá anexar todo o projeto em formato .zip.
- Deadline: **17/04/2023**. Não terá adiamentos e não serão aceitos trabalhos após o prazo de entrega.

6 Dúvidas

Qualquer dúvida ou dificuldade, não hesite em me procurar. O mais importante é que você aprenda.

Meu e-mail é `lucas.ribas@unesp.br` e meu gabinete é o 12 no DCCE (envie um e-mail antes para checar a disponibilidade).