UFRPE UAST Unidade Acadêmica de Serra Talhada - PE Desde 2006

BSi Bacharelado em Sistemas de Informação

MPOO

Site: https://sigs.ufrpe.br/sigaa/ava/index.jsf

Disciplina: Modelagem e Programação Orientada a Objetos (MPOO)

Profº: Richarlyson D'Emery

LISTA DE EXERCÍCIOS XI

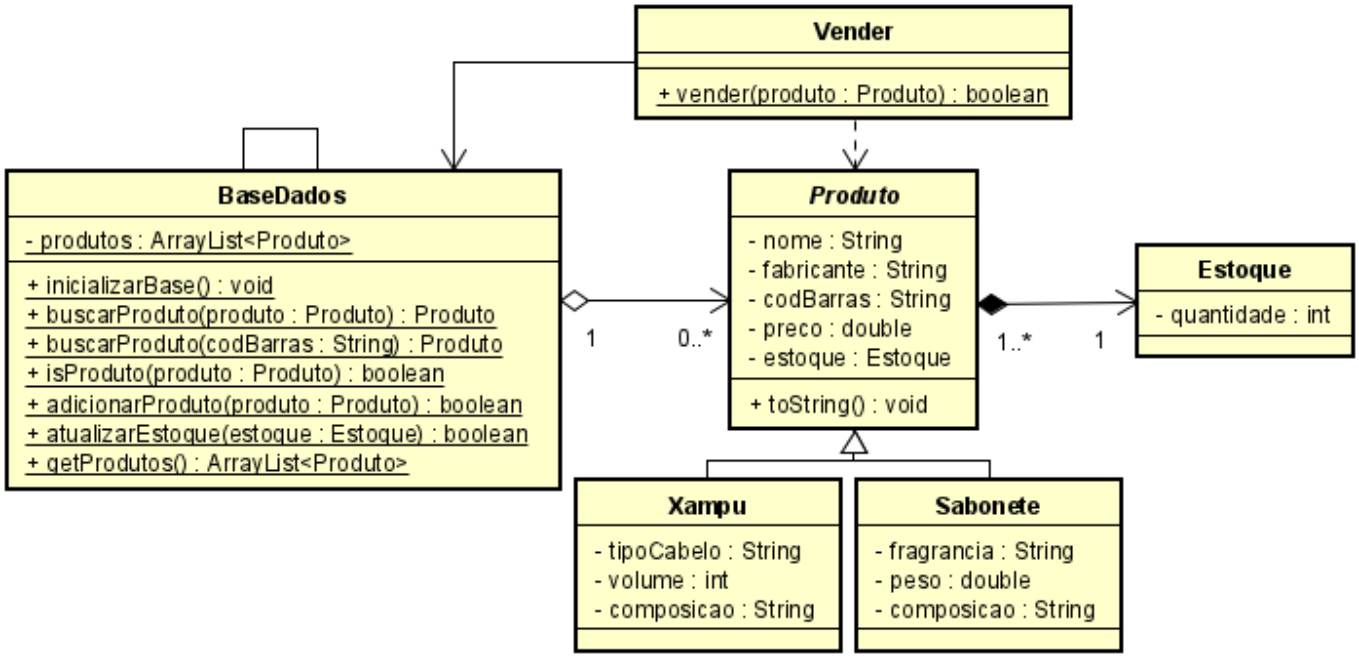
Leia atentamente as instruções gerais:

- No Eclipse crie um novo **projeto chamado br.edu.mpoo.listaXI.SeuNomeSobrenome**, o qual deverá ter **pastas de pacotes** para cada questão: questao1, questao2, e assim sucessivamente, contendo todas as respostas da lista.
- A lista envolve questões práticas, então deverão ser desenvolvidos os códigos-fonte (projeto completo). Em caso de diagramas, você poderá salvar o arquivo também na pasta correspondente do projeto.

Mão na Massa!

Você, aluno de MPOO, está experienciando situações-problemas do universo de desenvolvimento de software e começará a ser desafiado a solucionar problemas a partir de conhecimentos de Programação e Orientação a Objetos.

1) Observe o diagrama de classes abaixo e as regras de negócio definidas:



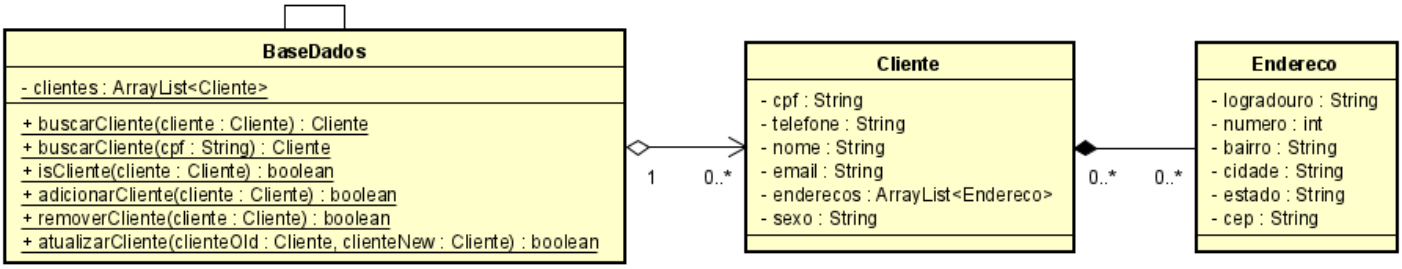
São regras de negócios:

- RN01 – um produto é identificado pelo codBarras;
- RN02 – um produto só poderá ser cadastrado uma única vez;
- RN03 – a quantidade de produto poder ser vista pelo atributo quantidade;
- RN04 – atualizarEstoque atualiza a quantidade de produtos disponíveis; e
- RN05 – um produto quando vendido tem seu estoque atualizado.

A partir dessas características, implemente o sistema em Java. Demonstre em uma aplicação:

- A criação de diversos produtos;
- A atualização de estoque dos produtos criados; w
- A venda de alguns produtos.

2) Um contratante solicitou a empresa MPOOSoftware LTDA um sistema de cadastro de clientes. O Scrum Master de MPOOSoftware LTDA solicitou a um de seus programadores (de codinome *mustela putórius furo* – "O Furão") que resolvesse essa demanda. Para isso apresentou o diagrama de classes abaixo produzido pela equipe de modelagem.



Antes de responder, analise as seguintes regras de negócios:

- RN01 – um cliente é identificado pelo seu cpf;
- RN02 – um cliente só poderá ser cadastrado uma única vez;
- RN03 – um cliente poderá ter diversos endereços, desde que contenham todos os dados válidos;
- RN04 – um cliente quando removido do sistema tem seus endereços removidos;
- RN05 – um cliente não poderá ter um mesmo endereço repetido; e
- RN06 – diversos clientes podem ter um mesmo endereço.

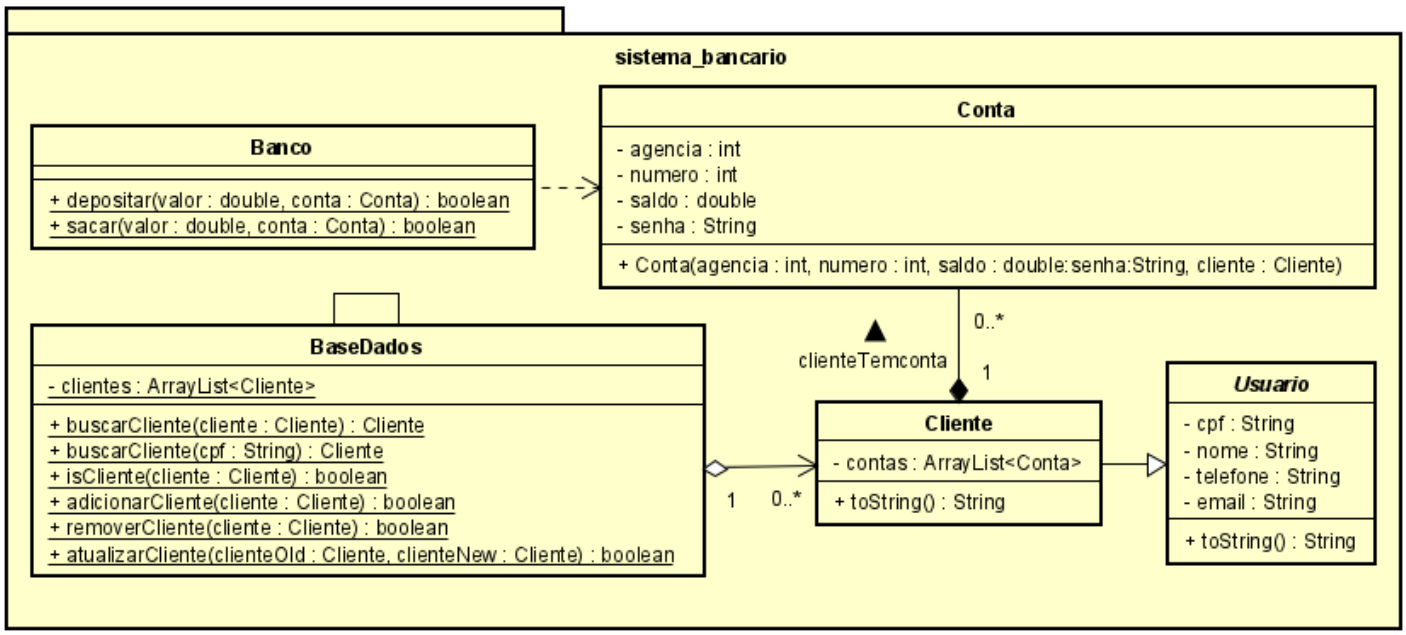
Ilustre em uma aplicação:

(i) o cadastro de dois clientes em que o primeiro cliente possui um endereço e o segundo dois endereços.
(ii) São endereços:
- Rua dos Cactos, n. 100, Cactolândia, Serra Talhada-PE, CEP 56970-000
- Av. Gregório Ferraz Nogueira, n. 20, José Tomé de Souza Ramos, Serra Talhada-PE, CEP 56909-535

Desafio

3) **(Desafio)** Após a entrega da primeira versão do sistema, o contratante solicitou a **inclusão de uma nova funcionalidade: "*o sistema deveria ter uma base de endereços independente de seus clientes*"**. O Scrum Master de MPOOSoftware LTDA solicitou para "O Furão" que aproveitasse a primeira versão do sistema de maneira a economizar tempo de programação. Entretanto "O Furão" não soube atender a essa demanda o que levou a empresa de desenvolvimento de software a promover um *hackathon* para a contratação de um novo funcionário. Mostre que você é um exímio programador e **apresente uma solução contendo um diagrama de classes e a codificação Java** para garantir essa vaga!

4) Implemente em Java a codificação para o sistema representado no diagrama de classes e abaixo:



Antes de responder, analise as seguintes descrições:

- Cliente e Conta representam um caso de composição;

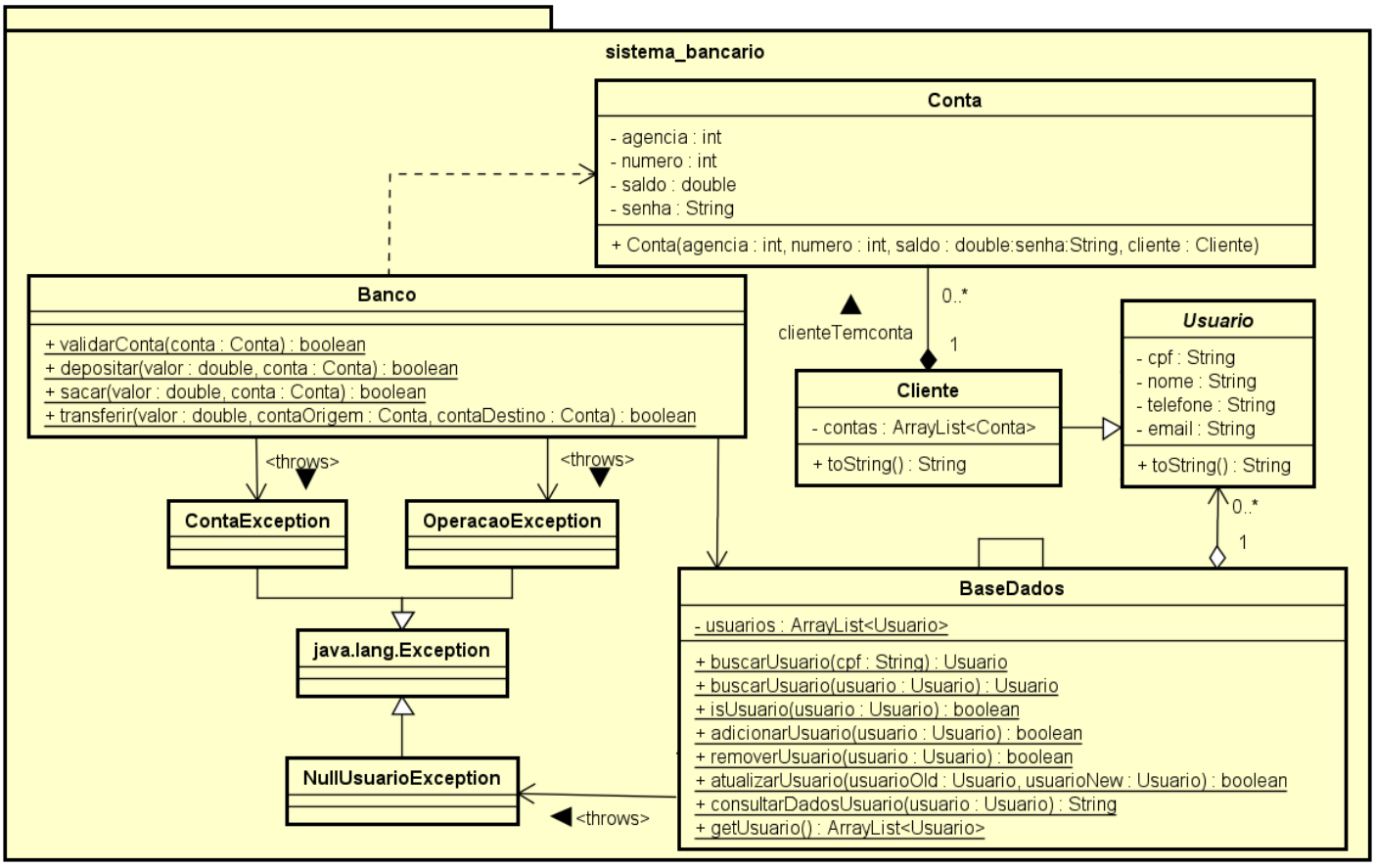
- As operações no saldo de uma conta são dadas pelos comportamentos de sacar e depositar. O saque só deve ser realizado se o valor do saque for igual ou superior ao valor do saldo;
- Falhas de saque ou depósito devem exibir em console as mensagens, respectivamente: “Erro ao efetuar o saque” ou “Erro ao efetuar o depósito”;
- A consulta dos dados de um usuário deve utilizar o método toString(); e
- Na classe BaseDados, os clientes deverão ser manipulado pelos métodos definidos.
    - Um cliente só deve ser adicionado se não existir na base. Use como critério de existência o cpf do cliente.
    - A verificação da existência de um cliente é dada por `boolean isCliente (Cliente cliente) {}`. Utilize o cpf como critério de existência.
    - Observe as diversas maneiras para buscar um cliente: ou pelo cpf ou por uma instância válida de cliente.
- As contas do sistema bancário devem ser *auto-increment*.

Crie uma App, em que:

- A base possui um cliente com duas contas e outro cliente com apenas uma conta;
- Exibe os dados dos clientes;
- Realiza o saque de uma conta e deposita a quantia em outra conta; e
- Exibe em console os valores antes e depois de cada operação.

5) No problema anterior, o projeto UML não prevê algumas situações de “exceção”, como, por exemplo, o fato de uma conta não existir nas operações de depósito e/ou saque; além da possibilidade de uso de “polimorfismo de objetos”, em que a base de dados passa a ter a relação com Usuario ao invés de Cliente e que o banco possa realizar a transferência de uma conta para outra. Sendo assim, a partir do diagrama repensando, faça a devida análise e proceda com as devidas modificações no projeto. (Utilize a estratégia de copiar a pasta de pacotes questao4 para a pasta questao5 de maneira a reaproveitar a codificação da questão anterior)

Problema 1: O diagrama de classes abaixo apresenta as dependência entre classes e tratamento de exceção.



Algumas descrições do sistema:

- Cliente e Conta representam um caso de composição.

- Toda operação sobre Conta deverá validar se conta é válida.
- As operações no saldo de uma conta são dadas pelos comportamentos de sacar e depositar. O saque só deve ser realizado se o valor do saque for igual ou superior ao valor do saldo.
- As operações de transferência utilizam as operações de saque e depósito.
- Falhas de depósito ou saque devem levantar a exceção OperacaoException, que tem como tratamento a exibição de um JOptionPane com a mensagem: “Erro ao efetuar o saque” ou “Erro ao efetuar o depósito”.
- A consulta dos dados de um usuário deve utilizar o método toString(), exceto da exibição do dado sigiloso: senha.
- Na classe BaseDados, os usuários deverão ser manipulado pelos métodos definidos.
    - Um usuário só deve ser adicionado se não existir na base. Use como critério de existência o cpf do usuário.
    - A verificação da existência de um usuário é dada por `boolean isUsuario(Usuario usuario) {}`. Utilize o cpf como critério de existência.
    - Observe as diversas maneiras para buscar um usuário: ou pelo cpf do usuário ou por um usuário. Um usuário é identificado pelo seu cpf.
    - O método isUsuario de BaseDados deve levantar a exceção NullUsuarioException caso um usuário não pertença a base. Como tratamento deve-se exibir um JOptionPane com a mensagem: "Cliente não existe" se o usuário for do tipo Cliente.
- O método da classe Banco `public static boolean validarConta(Conta conta) throws ContaException{}` deve levantar a Exceção ContaException caso a conta informada não seja uma conta válida. Esse método deverá ser utilizado ao se tentar realizar uma transferência. Faça o devido uso do bloco **try** - **catch** na chamada do método e **throw** e **throws** para o tratamento da Exceção. No tratamento da validação de uma Conta, caso uma conta não seja válida, exiba em JOptionPane a mensagem: "Conta Origem Inválida" ou "Conta Destino Inválida".
- As contas do banco devem ser *auto-increment*.

Problema 2: Crie uma App, na qual:

- Possui os clientes:
    - José, cpf 111.111.111-11.
        - Conta agencia: 1, numero 1, saldo R$ 0.0, senha: 123.
        - Conta agencia: 1, numero 2, saldo R$ 100.0, senha: jose123.
    - Maria, cpf 222.222.222-22, conta agencia: 1, numero 3, saldo R$ 1000.0, senha: Maria222.
- Possui uma transferência de R$100 de Maria para José.
- Exibe em console os valores antes e depois da transferência.
- Ilustra todas as exceções. Antes de cada comando, adicione um `//comentário` explicando que Exceção está sendo realizada.