

	INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ Curso: ADS Disciplina: Programação Orientada a Objetos Professor: Ely
--	---

Exercício 05

Atualize a implementação das classes apresentadas em sala de acordo com as seguintes abaixo.

1) Classe Cliente

Crie uma classe Cliente com os seguintes atributos:

- id: Identificador único do cliente (número);
- nome: Nome completo do cliente (string);
- cpf: CPF único do cliente (string);
- dataNascimento: Data de nascimento do cliente (Date);
- contas: Array de contas associadas ao cliente.

2) Classe Conta

Atualize a classe Conta para incluir os seguintes atributos:

- id: Identificador único da conta (número);
- cliente: Cliente associado à conta;
- dataDeAbertura: data em que a conta foi criada.

3) Classe Banco

Atualize a classe Banco para incluir os seguintes atributos e métodos:

- clientes: Array de objetos do tipo Cliente, além do array de contas.

Além dos métodos já criados, crie também

a) Adicionar cliente

- Método: inserirCliente(cliente: Cliente): void

- Insere o cliente passado por parâmetro no array de clientes.

b) Consultar cliente pelo CPF

- Método: consultarCliente(cpf: string): Cliente
- Retorne o cliente correspondente ao CPF;

c) Associar um cliente a uma conta

- Método: associarContaCliente(numeroConta: string, cpfCliente: string): void
- Procure o cliente e a conta com os dados fornecidos e associe-os, respeitando considerando que o cliente não pode ter a mesma conta adicionada mais de uma vez.

d) Listar contas de um cliente

- Método: listarContasCliente(cpf: string): Conta[]
- Retorne todas as contas associadas ao cliente cujo CPF foi informado.

e) Totalizar saldo por cliente

- Método: totalizarSaldoCliente(cpf: string): number
- Calcule e retorne o saldo total de todas as contas de um cliente. f)

Incluir um cliente

- Método: inserirCliente(cliente: Cliente): void
- Adicione o cliente ao array de clientes, respeitando as seguintes regras:
- Não permitir que um cliente com o mesmo id ou cpf seja cadastrado mais de uma vez.

g) Alterar o método de incluir uma conta

- Método: inserirConta(conta: Conta): void
- Adicione a conta ao array de contas, não permitindo que uma conta com o mesmo id ou numero seja criada.

4) Regras de Negócio

- a. Cada cliente pode ter várias contas, mas uma mesma conta não pode

ser associada mais de uma vez ao mesmo cliente.

b. Cada conta só pode ser associada a um único cliente.

c. O sistema deve impedir duplicações:

i. Nenhum cliente pode ter o mesmo id ou cpf de outro cliente.

ii. Nenhuma conta pode ter o mesmo id ou numero de outra conta.

5) Realize testes em todos os métodos da classe Banco.

R: O código que obedece às especificações acima se encontra no arquivo externo (banco_atualizado.ts).

6) Questionamentos:

a. Você concorda que o banco faz o cadastro de duas entidades e ainda faz regras de negócios?

R: Não concordo. Pois a classe Banco do código já possui muitas responsabilidades, como armazenar listas de Cliente e Conta, realizar as operações CRUD e aplicar as regras de negócio. Dessa forma, eu creio que ter uma classe que faz tantas coisas diferentes torna difícil as funções de administração e filtragem de erros.

b. Não seria adequado o banco ter uma class CadastroDeClientes e CadastroDeContas e algumas regras de validação serem feitas no banco e deixar os métodos de consulta e inclusão os mais simples possíveis?

R: Sim, essa abordagem seria mais adequada para garantir um bom design orientada a objetos. Dessa forma a classe Banco atuaria como uma espécie de serviço que usaria as instâncias de CadastroDeClientes e CadastroDeContas para realizar operações de negócio mais complexas como associarContaCliente.

c. O método associar cliente a uma conta deveria estar em que classe? Banco, CadastroDeContas ou CadastroDeClientes?

R: O método associarContaCliente deveria estar na classe Banco. Pois a classe Banco é a única que deve ter conhecimento e acesso aos dois cadastros para realizar essa orquestração.