UNIVERSIDADE FEDERAL DE LAVRAS

# Práticas de Programação Orientada a Objetos

GAC106 – 2022-1

## Exercícios de Conta Bancária

Versão: 2022-06-10

*Julio César Alves* *DAC - ICET* *https://julio.dac.ufla.br/*

É hora de colocarmos a **mão na massa** mais uma vez.

Lembre-se que essa é uma **disciplina de programação** e, logo, **essencialmente prática**. Aproveite então a oportunidade e aprofunde seus conhecimentos.

Vamos criar nosso programa completo do zero e reforçar, principalmente, os seguintes conceitos:

- Entrada e saída padrão.
- Construtores.
- Encapsulamento (visibilidade).
- Atributos e métodos estáticos.

# Passo 1 – Conta Bancária

Esse e os próximos exercícios se referem às operações de uma conta bancária. Nesse primeiro, crie uma classe *Conta*, que atenda às necessidades abaixo. Lembre-se de usar adequadamente a visibilidade (*public/private*) dos atributos e métodos.

- As contas devem possuir um saldo e um limite.
- Deve existir um método para consultar o saldo.
- Devem existir métodos para saque e para depósito de valores na conta.
- A conta pode ter saldo negativo no máximo até o limite estabelecido.
- A classe *Conta* não deve ter nenhuma interação com o usuário (ou seja, dentro dela não deve existir exibição de mensagens, nem leitura de dados).

Crie também uma outra classe que represente um Banco. Nessa classe, crie um atributo para guardar uma conta, implemente um menu com as opções abaixo e implemente métodos separados para cada opção.

- Criar Conta, Consultar Saldo, Depositar, Sacar e Sair.

Por fim, crie uma classe que tem o método *main*. O método *main* deve apenas criar um objeto da classe que representa o banco e disparar a execução do menu.

Teste sua implementação!

Dica: Lembre-se de que você já implementou um menu ao seguir os slides *PPOO 1.4 - Praticando OO com Java*.

# Passo 2 – Titular da Conta

A classe *Conta* agora deve passar a tratar dados do seu titular. Para isso, crie uma nova classe *Cliente* que tem:

- Atributos nome e CPF.
- E métodos para consultar o nome e o CPF do cliente.
- Lembrando que a classe *Cliente* também não deve ter nenhuma interação com o usuário.

Em seguida, altere a classe *Conta* para que ela tenha um atributo do tipo *Cliente*.

- Trate o construtor da classe *Conta* de forma que ele receba um objeto cliente por parâmetro.

Por fim, altere a classe *Banco* para que:

- Na criação de contas, obtenha do usuário o nome e o CPF do cliente, criando o objeto correspondente e utilizando-o para criar a conta.
- Na opção de consultar saldo, seja exibido o nome do titular da conta junto com o saldo (*cuidado com o encapsulamento!*).

Testes suas alterações!

# Passo 3 – Construtores

Altere a classe *Conta* de forma que ela passe a ter dois construtores.

- Os dois construtores devem receber o cliente e o valor do limite da conta.
- E apenas um deles deve receber o valor do saldo inicial da conta. Portanto, no outro construtor o saldo deve ser inicializado com zero.

Altere então a classe do banco para que existam duas opções diferentes de criação de contas.

- Uma que pede para o usuário o saldo inicial e outra que não pede.
- Cada opção deve chamar o construtor correspondente.

Como o operador *this* pode ser utilizado em um dos construtores?

# Passo 4 – Numeração das contas

Você deve criar um atributo na classe *Conta* para guardar o número da conta e deve existir um método para consultar esse atributo.

- Podemos tratar a numeração das contas de forma que ela seja automática (sem precisar ser informada pelo usuário). Para isso, crie um atributo estático na classe *Conta* para guardar o número da última conta criada (inicialmente ele deve ter o valor 1000).
  - Repare que são dois atributos diferentes: um é o número da conta que é diferente para cada conta, e o outro é um atributo estático que indica o número da última conta criada.
- No construtor* da classe *Conta* este atributo deve ser incrementado, e o seu valor deve ser usado como número da conta que está sendo criada.

Na classe que trata o banco, crie mais um atributo para que ela passe então a tratar duas contas.

- Na opção de menu de criar contas, trate para que o usuário forneça os dados das duas contas.
- Após a criação de cada conta, o programa deve exibir o número gerado para cada conta de forma que o usuário saiba o número que deve usar nas demais operações.

Lembre-se também de alterar as demais opções de menu pedindo ao usuário o número da conta sobre a qual ele quer realizar uma operação (consultar saldo, saque e depósito).

*Obs: uma boa prática de programação é evitar a replicação (repetição de código). Portanto, altere apenas um construtor e garanta que ele será chamado pelo outro.

# Passo 5 – Transferência entre Contas

Agora que conseguimos criar contas diferentes, podemos fazer transferências entre elas.

Crie um método na classe *Conta* para realizar a transferência entre contas.

- Ele deve receber por parâmetro o valor a ser transferido e o objeto da conta de destino.
- O método não deve alterar diretamente o saldo da conta e sim chamar os métodos de saque e depósito apropriadamente.
- Lembre-se que a conta não deve permitir a transferência se o valor solicitado estiver além do limite da conta de origem.

Na classe que representa o banco, crie uma opção de menu para transferência entre contas.

- O usuário deverá escolher qual será a conta de origem e qual será a conta de destino (de acordo com os números das contas).

Teste sua implementação!

# [Opcional] Passo 6 – Rendimento na conta

Suponha que nossa conta é especial e possui um rendimento periódico. Para tratá-lo faça o seguinte:

- Na classe *Conta*, crie um atributo estático para guardar o valor da taxa de rendimento (percentual).
- Crie um método *render* que aplica a taxa de rendimento ao saldo da conta. Tal método deve ser estático ou não? Por que?
- Crie uma opção de menu na classe que representa o banco para que o usuário possa fazer a conta render.

# [Opcional] Passo 7 – Alterando rendimento

Crie um método na classe *Conta* para alterar a taxa de rendimento da conta (e crie uma opção de menu para o usuário informar a nova taxa).

O método deve ser estático ou não? Por que?

Bacana! Ao chegar até aqui, exercitamos diversos conceitos importantes da disciplina.