Laboratório 2 - Simulação de Sistema Bancário

Universidade Federal do Rio Grande do Norte (UFRN)
Instituto Metrópole Digital (IMD)
Disciplina: Linguagem de Programação 2
Valor: 5,0 pontos

UFRN

Descrição do Problema

Um banco digital em ascensão, o (escolha o nome do seu banco boe), precisa de um sistema de software para gerenciar as contas de seus clientes. O sistema deve ser capaz de criar diferentes tipos de contas, processar transações financeiras básicas (depósito, saque, transferência) e calcular tributos específicos.

Para modelar este sistema, você aplicará conceitos fundamentais de Programação Orientada a Objetos. A ideia central é criar uma estrutura flexível que permita ao banco, no futuro, adicionar novos tipos de contas (como Conta Salário ou Conta Investimento) com o mínimo de alteração no código existente. Para isso, utilizaremos **Herança** para modelar os tipos de conta, **Interfaces** para definir contratos de comportamento (como ser tributável) e **Polimorfismo** para manipular as diferentes contas de forma unificada.

Requisitos Funcionais

O sistema deverá ser executado via console e apresentar um menu principal com as seguintes opções:

1. **Criar Conta:**
    - O sistema deve perguntar o nome do cliente.
    - O usuário deverá escolher o tipo de conta a ser criada: **Conta Corrente** ou **Conta Poupança**.
    - Um número de conta deve ser gerado automaticamente de forma sequencial (101, 102, 103, ...).
    - A conta criada deve ser armazenada em uma lista de contas.
2. **Realizar Depósito:**
    - Solicitar o número da conta e o valor a ser depositado.
    - Encontrar a conta na lista e adicionar o valor ao saldo. Exibir uma mensagem de sucesso.
3. **Realizar Saque:**
    - Solicitar o número da conta e o valor a ser sacado.
    - Verificar se há saldo suficiente.
        - Para **Conta Corrente**, o saque possui uma taxa de 5% sobre o valor sacado. O saldo final não pode ficar negativo.
        - Para **Conta Poupança**, o saque não possui taxa.
    - Se o saque for possível, decrementar o valor do saldo. Caso contrário, exibir uma mensagem de saldo insuficiente.
4. **Realizar Transferência:**
    - Solicitar o número da conta de **origem**, o número da conta de **destino** e o **valor**.
    - Aplicar a mesma lógica e taxa do saque na conta de origem e, se a operação for bem-sucedida, depositar o valor na conta de destino.
5. **Listar Contas:**
    - Exibir uma lista com todas as contas cadastradas.
    - Para cada conta, mostrar: Número, Nome do Cliente, Saldo e o Tipo da Conta (Corrente ou Poupança).
6. **Calcular Total de Tributos:**
    - O sistema deve percorrer todas as contas e calcular o valor total dos tributos a serem recolhidos.
    - Apenas a **Conta Corrente** é tributável. O tributo corresponde a **1% do saldo** atual da conta.
    - Ao final, exibir o valor total consolidado dos tributos.
7. **Sair:**
    - Encerra a execução do programa.

Requisitos Técnicos Obrigatórios

Para a implementação, você **deve** utilizar os seguintes conceitos:

- **Linguagem:** Java.
- **Orientação a Objetos:**
  - **Classe Abstrata (Conta):** Crie uma classe base abstrata Conta com os atributos e métodos comuns a todas as contas (ex: numero, cliente, saldo, depositar(), sacar(), transferir()). Os métodos de saque e transferência devem ser abstratos para forçar a implementação nas classes filhas.
  - **Herança:** Crie as classes ContaCorrente e ContaPoupanca que **herdam** da classe Conta.
  - **Polimorfismo:** Utilize uma única lista (ArrayList<Conta>) para armazenar objetos de ContaCorrente e ContaPoupanca. As chamadas de métodos (como sacar()) devem se comportar de maneira diferente dependendo do tipo real do objeto na lista.
  - **Interface (ITributavel):** Crie uma interface chamada ITributavel contendo um método calculaTributos(). A classe ContaCorrente deve **implementar** esta interface.
- **Estrutura de Dados:** Utilizar ArrayList para armazenar a lista de contas.
- **Controle de Fluxo:** O menu principal deve ser implementado com switch-case e um laço while.
- **Interface:** A interação com o usuário deve ser realizada via console (Scanner).

Instruções de Entrega

1. Crie um novo repositório no GitHub para desenvolver a atividade. Mantenha um bom histórico de commits.
2. Para a entrega, submeta o link do seu repositório do GitHub ou um arquivo .zip contendo os arquivos .java do projeto no SIGAA.

Ajudinha do Prof:

Para a funcionalidade de "Calcular Total de Tributos", você precisará percorrer a lista de contas e verificar quais delas são tributáveis. O operador instanceof é perfeito para isso! Ele permite checar se um objeto é uma instância de uma determinada classe ou se implementa uma interface.

Veja um exemplo de como você pode estruturar essa lógica:

```
// Supondo que você tenha uma lista de contas: ArrayList<Conta> listaDeContas;
double totalTributos = 0.0;

// Percorre a lista polimorficamente
for (Conta conta : listaDeContas) {
    // Verifica se a conta da iteração atual implementa a interface ITributavel
    if (conta instanceof ITributavel) {
        // Se sim, fazemos um "cast" para poder chamar o método da interface
        ITributavel contaTributavel = (ITributavel) conta;
        totalTributos += contaTributavel.calculaTributos();
    }
}

System.out.println("\n========================================");
System.out.println("Total de tributos a recolher: R$ " + totalTributos);
System.out.println("========================================");
```

Bom trabalho!