# PROVA 04

Universidade Católica de Pernambuco
Professor: Augusto César Oliveira
Disciplina: Programação III
Aluno(a): _____ data: ___/___/_____

**Observações**:
***** A prova deve ser feita em **dupla** ou **individual** e deve ser apresentada em sala de aula em até 15 minutos na **data agendada no Teams**.

* Os códigos devem ser enviados pelo Teams até às 23:59 do dia que antecede a data da apresentação.

**Questão 01**. Um pacote chamado **Banco** deve ser implementado utilizando uma IDE de sua preferência. Dentro deste pacote, as seguintes classes devem ser criadas (peso 0,5):

1. Cliente
2. Conta;
3. ContaPoupanca;
4. ContaCorrente;
5. TesteOperacoes.

**Questão 02**. A classe **Cliente** deve possuir como atributos privados peso (0,5):

- nome;
- endereço;
- profissão.

**Questão 03**. A classe **Cliente** deve possuir como métodos (peso 0,5):

- **getters** e **setters** para cada atributo;
- **exibirAtributos()**: mostra na tela todos os atributos do cliente.

**Questão 04**. A classe **Conta** é uma **superclasse** que deve possuir os seguintes atributos privados (peso 0,5):

- número da agência;
- número da conta;
- saldo;
- cliente: objeto do tipo **Cliente**.

**Questão 5**. A classe **Conta** possui os seguintes métodos (peso 1,0):

1. **getters** e **setters** para cada atributo;
2. **deposito()**: recebe uma quantia de dinheiro e salva no saldo da conta;
3. **saque()**: remove uma quantia de dinheiro e retira do saldo da conta;
4. **transferencia()**: recebe como parâmetro um objeto do tipo conta; transfere uma quantia do saldo da conta atual para o saldo da conta que foi recebida como parâmetro. Garanta que exista saldo suficiente para isso.
5. **exibirSaldo()**: apresenta uma mensagem de texto informando o nome do usuário e o quanto ele possui na conta.

**Questão 06**. A classe **ContaPoupanca** é uma subclasse de **Conta** e tem o seguinte atributo (peso 0,5):

- **taxaRendimento**: valor fixo de 0.05.

**Questão 07**. A classe **ContaPoupanca** tem o seguinte método (peso 1,0):

a. **simularOperacao()**: esse método deve receber como parâmetro um número inteiro, representando a quantidade de meses desejados para simular o rendimento da conta poupança. O método deve retornar o rendimento do saldo atual de acordo com a **taxaRendimento** da conta. O rendimento ocorre a cada mês a partir da seguinte fórmula: **saldo + (taxaRendimento * saldo)**.

**Questão 08**. A classe **ContaCorrente** é uma subclasse de **Conta** e tem o seguinte atributo (peso 0,5):

- taxaManutencao: valor fixo de 50 reais.

**Questão 09**. A classe **ContaCorrente** tem o seguinte método (peso 1,0):

a. **simularOperacao()**: esse método deve receber como parâmetro um número inteiro, representando a quantidade de meses desejados para simular o valor de custo das operações da conta corrente. O método retorna o saldo final após o decrescimento da **taxaManutencao** mensal.

**Questão 10**. A classe **TesteOperacoes** possui os seguintes atributos (peso 0,5):

- listaClientes: uma lista que deve armazenar todos os clientes criados;
- listaContas: uma lista que deve armazenar todas as contas criadas.

**Questão 11**. A classe **TesteOperacoes** possui os seguintes métodos:

1. **criarConta()** (peso 1,0):

    a. Esse método deve receber as strings: nome, endereço e profissão do cliente por meio do método Scanner. Em seguida, essas informações devem instanciar um objeto do tipo **Cliente** e adicioná-lo no atributo **listaClientes**.

    b. Receba uma string: **tipoConta** por meio do método Scanner, que deve informar qual o tipo de conta que o usuário deseja criar (poupança ou corrente). Crie um objeto de acordo com a classe especificada.

    c. Para preencher o objeto, receba dois números inteiros (**número da agência** e **conta)** e um número flutuante: **saldo**, através do método Scanner. Adicione o objeto no atributo **listaContas**.

    d. Use o **tratamento de exceções** para evitar que o usuário informe números de agência e conta como string, bem como para evitar que ele insira o valor do saldo menor do que zero.

2. **realizarOperacoes()** (peso 1,0):

    a. Esse método deve receber quatro parâmetros: o número da agência e conta do **cliente que deseja enviar o dinheiro**, bem como o número da agência e conta do **cliente que vai receber o dinheiro**.

    b. Através desses parâmetros, realize duas buscas no atributo **listaContas** para encontrar a conta do cliente que deseja enviar o dinheiro e a conta do cliente que vai receber o

dinheiro. Dica: utilize um for para percorrer a lista e encontre o objeto através das funções do arraylist.

c. Use o método **transferencia()** existente na classe **Conta** para realizar a movimentação do valor.

3. **exibirSaldo()** (peso 1,0):

   a. Esse método deve receber como parâmetro os números de agência e conta do usuário que terá o saldo consultado. Realize isso utilizando o método Scanner.

   b. Percorra a lista de contas, encontre a conta pesquisada e a salve em um objeto.

   c. Receba através do método Scanner a quantidade de meses que vão ser simulados para demonstrar o saldo. Em seguida, utilizando polimorfismo, execute o método **exibirSaldo** da classe **Conta**.

4. **apresentarMenu()** (peso 0,5):

   a. Esse método deve apresentar três mensagens: i) Criar conta; ii) Realizar operações; e iii) Exibir saldo;

   b. De acordo com o valor recebido pelo usuário através do método Scanner, o método direciona para cada ação ao seu respectivo método.

   c. Utilize um **while** para executar o método **apresentarMenu** até que o usuário deseje parar as operações (obs: ao final de cada operação essa pergunta pode ser feita ao usuário).