# Trabalho Final: Instituição Financeira.

# Agradeço a Prof. Elaine pelo enunciado.

Perguntas:

1- Implementar usando a linguagem Java o sistema descrito a partir do conjunto de requisitos a seguir.

Situação problema: Sistema para uma Instituição Financeira

1. Para a instituição financeira é muito importante manter um cadastro de todas as contas que foram criadas na instituição. No momento de criação da conta é importante associar uma senha a ela. Essa senha será solicitada antes da execução de qualquer transação.
2. Para cada conta criada é importante saber se a conta está ativa ou já foi desativada (ou seja, o cliente encerrou a conta). Isso ajudará a filtrar as pesquisas. Ainda que um cliente encerre uma conta no banco, é importante manter o registro da conta na base de dados, mas com um indicativo de que ela não está ativa.
3. As contas da instituição podem pertencer às seguintes categorias: corrente, poupança e salário.
4. Para as contas da categoria corrente, as seguintes informações devem ser armazenadas: nro da conta, saldo atual, data de abertura, data da última movimentação, limite do cheque especial e valor da taxa administrativa.
5. Para as contas da categoria poupança as seguintes informações devem ser armazenadas: nro da conta, saldo atual, data de abertura, data da última movimentação, rendimento do mês atual.
6. Para as contas da categoria salário as seguintes informações devem ser armazenadas: nro da conta, saldo atual, data de abertura, data da última movimentação, limite para saque e limite para transferência.
7. Os clientes da instituição devem ser registrados com as seguintes informações: CPF, nome, endereço completo, estado civil, escolaridade, data de nascimento.
8. É importante também saber em qual agência o cliente foi cadastrado.
9. As agências bancárias devem ser previamente cadastradas, sendo que cada agência possui um número, um nome fictício e um endereço, sendo que os campos cidade, estado e bairro são campos frequentemente usados para se realizar buscas por agências.
10. Cada agência possui um gerente, que é um funcionário nomeado para tal tarefa. Cada gerente pode gerenciar apenas uma agência.
11. Os gerentes são funcionários da empresa que precisam ter armazenados alguns atributos específicos: data de ingresso na carreira de gerente, agência que ele gerencia e se possui curso de formação básico em gerência.
12. Para os funcionários da agência é preciso armazenar as seguintes informações: CPF, nome completo, nro da carteira de trabalho, RG, data de nascimento, endereço, sexo, estado civil, cargo na empresa (existem diferentes cargos além do gerente), salário e ano de ingresso.
13. Toda conta bancária deve ter no mínimo um cliente associado. No entanto, uma conta pode ser conjunta. Nesse caso, é possível ter dois clientes associados à mesma conta. Um mesmo cliente pode ter diferentes contas bancárias.
14. Toda conta bancária está ligada a uma agência bancária, ou seja, a agência na qual ela foi criada.
15. Uma vez que um cliente tenha uma conta bancária ele pode efetuar diferentes movimentações financeiras, conhecidas como transações bancárias. Cada transação bancária está relacionada a uma conta bancária. Além disso, é importante armazenar a data em que a transação foi realizada, o valor da transação e o canal onde foi feito a transação (internet banking, caixa eletrônico ou caixa físico). A realização de uma transação pode impactar no saldo da conta.
16. Cada transação possui um tipo. Os tipos de transação atualmente disponíveis são: saque, depósito, consultar saldo e efetuar pagamento. Em cada uma dessas transações é importante armazenar apenas o valor da transação. Com o objetivo de simplificar o trabalho não vamos armazenar os atributos específicos de cada tipo de transação, embora esse seja um requisito interessante em problemas reais.
17. Implemente em Java, usando técnicas de encapsulamento em **todo** o **Sistema de Instituição Financeira** (gets and sets).
18. Para validação de CPF, implemente o algoritmo real de validação de CPF (que é encontrado facilmente na Internet, implementado nas mais diversas linguagens).

1. Considerando o uso de construtores:
2. Crie um construtor para a classe cliente que receba como parâmetro o seu nome e o seu CPF.
3. Crie um construtor *default* (sem parâmetros) explicitamente para a classe cliente.
4. Crie um construtor para a classe Transação de forma que esta receba como parâmetro a conta e a data. Desta forma, obriga-se a associação da transação com a conta, que já foi previamente criada no sistema.
5. Crie construtores para cada uma das classes do problema de forma a atribuir valor aos atributos da classe com os parâmetros recebidos. Toda classe deve ter pelo menos um construtor criado por você.
6. Considerando os casos de herança, implemente todas as situações de herança descritas nos requisitos. Lembre-se que existe um relacionamento de herança entre a classe conta (superclasse) e as subclasses que indicam cada tipo de conta bancária.
7. Lembre-se também de implementar a herança entre o funcionário e o gerente
8. Uma herança possível de ser implementada é a que representa as pessoas do sistema e tem como subclasses cliente e funcionários. Implemente também essa herança.
9. Implemente um atributo estático para armazenar a comissão recebida por um gerente para gerenciar uma agência.
10. Como este atributo estático é encapsulado (*private*), crie os métodos get e set para manipulá-lo. Os métodos devem ser estáticos.
11. Implemente um método calcular Salario específico para funcionários, calculado por meio do seu salário base mais um adicional de 10% caso ele tenha mais que 15 anos de empresa.
12. Implemente um método calcular Salario específico para gerentes, calculado pela soma do salário de um funcionário comum acrescido pela comissão recebida por gerenciar uma agência bancária
13. Implemente a realização de uma transação, como uma das possibilidades: sacar um valor, depositar um valor, consultar saldo e efetuar pagamento. Lembre-se, só é possível sacar e efetuar pagamento se existir saldo suficiente em conta. O depósito deve receber um valor maior que zero.

# Classes Abstratas

* 1. Implemente a classe Pessoa como abstrata.
  2. Invente um requisito no problema que justifique a criação de uma classe ou um método abstrato e sua implementação nas subclasses. Não esqueça de descrever o requisito.

# Exceptions & Interfaces

* 1. Implemente pelo menos uma nova exceção e utilize em seu código.
  2. Crie uma interface e use em pelo menos duas classes não relacionadas.

1. Persistência: fazer persistência dos dados principais. Não escrever os dados no formato texto.
2. Apresentar diagrama UML (opcional).
3. Fazer uma interface (pode ser textual) para interação dos dados (cadastrar cliente, contas, realizar operações bancárias, e outras operações que permita ao usuário do programa acesso completo as funcionalidades implementadas.