Exercício Complementar de Orientação a Objetos em Java

Objetivo: Criar a modelagem de uma conta bancária utilizando todos os conceitos da orientação a objetos aprendidos na apostila estudada Orientação a Objetos em Java (K19). Ao final do exercício deverá haver um pequeno sistema apenas com a modelagem que testará os conceitos estudados. O sistema terá Banco, Agência, Conta Corrente, Poupança, Conta Premium, Cliente, Cliente Premium, Gerente e Caixa. Estes são os objetos principais, será necessário mais classes para aplicar os conceitos.

1. Criar um objeto Banco, deve ser possível serializar(S) este objeto. Ele deve ser capaz de armazenar um código de 3 posições do tipo int e um nome do tipo String, o código, mesmo que seja informado um número maior deve-se armazenar apenas 3 posições. O objeto deve ser sempre criado. Seus valores não podem ser alterados depois de ter sido criado, mas podem ser lidos. O objeto deve ser capaz de ser clonado(C) e o código definirá que dois objetos são iguais(E[codigo]).
2. Criar um objeto Agência(S, C, E[banco, digito, numero]). Deve ser capaz de armazenar um número com 4 posições do tipo int, dígito com 2 posições do tipo char, nome do tipo String e um relacionamento de

agregação com banco. O objeto deve ser sempre criado. Seus valores não podem ser alterados depois de ter sido criado, mas podem ser lidos. Os campos número e digito mesmo que informado mais posições só podem armazenar o seu limite.

3. Conta Corrente, Poupança e Conta Premium serão objetos semelhantes e da mesma classificação. Eu nunca poderei criar uma conta que não seja de nenhuma dessas especificadas, porém a Conta deve ser capaz de ser clonada. Os atributos em comum serão número com 7 posições do tipo int, dígito com 2 posições do tipo char, saldo dever ser um valor decimal (BigDecimal é mais fácil de fazer cálculos), um atributo que identifique um Status (Ativo ou Inativo, utilize enumerations aninhado que tenha um código do tipo char com uma posição e que seja possível distinguí-lo pelo código) e um relacionamento de agregação com agência. A conta também deve ser capaz de armazenar um agendamento de depósito, aconselho a usar Map de String e BigDecimal. Com exceção do agendamento os outros atributos só podem ser lidos, o agendamento não pode ser alterado e nem lido por qualquer outra classe em hipótese alguma. A comparação da conta deve ser feita pelos atributos agência, dígito e número. Em Conta o depósito só pode ser feito por um Cliente, deve receber como parâmetro o valor a ser depositado e retornar o saldo. Caso o dia em que o depósito esteja sendo feito não seja um dia útil, a operação não deve ser feita, ele deve ser agendada para ser feita em uma data posterior (utilizar a classe Calendar facilitará). O saque só pode ser feito por um cliente, deve receber o valor a ser sacado e retornar o saldo. Antes de fazer o saque deve verificar se tem saldo suficiente para sacar, caso não tenha deve ser interrompido o processo. O Gerente deve ser capaz de ativar e inativar uma conta assim como conferir os depósitos agendados que consiste em verificar se o dia atual é maior ou igual ao dia do agendamento, caso seja deve ser removido do

agendamento e deve ser feito o depósito, a conferência só pode ser feita se o dia atual for um dia útil.

4. Conta Corrente deve haver um limite de cheque especial que deverá iniciar em 5.000,00 e somente o gerente pode alterar o saldo de cheque especial que por padrão deverá ser 5.000,00. Ao ler o saldo de uma Conta Corrente, deve retornar o saldo do cheque especial somado ao saldo da conta. Ao fazer um saque de uma Conta Corrente, caso não tenha saldo suficiente para ser feito o saque, o saque deve ser feito do cheque especial que deve certificar-se de que haja limite no cheque especial para fazer o saque, exceto no caso do Cliente ser um Cliente Premium, neste caso o cheque especial pode ficar negativo. Após fazer o saque deve ser retornado o saldo atual. Ao fazer um depósito na Conta Corrente deve-se verificar se o cheque especial está abaixo do limite, caso esteja deve ser feito o depósito no cheque especial, até que ele volte a ter o saldo no seu limite permitido. Quando atingir o limite a diferença deve ser depositada no saldo da conta mesmo.
5. Poupança deve ter um atribudo rendimento que deve iniciar em 0,00. Sempre que for feito um depósito em Poupança deverá ser somado ao rendimento 2% do valor depositado e o valor a ser depositado será o valor acrescido de 2%.
6. Gerente e Caixa são funcionários. Os Funcionários(S, E[cpf]) devem ter cpf e nome, ambos uma String, que depois de serem criados não podem ser alterados apenas lidos. Funcionário nunca poderá ser instanciado. Tanto Gerente como Caixa devem fazer parte de um setor financeiro onde possa imprimir um extrato da movimentação de uma conta independente de saber qual desses dois funcionários se tenha uma instância. A Movimentação(S, C) deve apresentar a data do movimento, conta, Tipo (Débito ou Crédito, utilize enumerations aninhado que tenha um código do tipo char com uma posição e que seja possível distingui-lo pelo código), valor movimentado, saldo da

conta e saldo do cheque especial (apenas para gerente). Ela não precisa implementar uma regra de negócio, os dados serão preenchidos através de uma simulação a uma consulta em banco de dados, que pode ser uma classe com objetos e métodos que não precisam ser instanciados e a classe não pode ser extendida para garantir que não seja modificado o processo feito nela. O Gerente altera o limite do cheque especial, inativa uma conta, ativa uma conta e confere depósitos agendados.

7. Cliente(S, E[cpf]) deve ter um cpf e nome do tipo String e uma lista de contas. Esses dados não podem ser alterados após ter sido criado o objeto, mas podem ser lidos. O cliente pode adicionar ou remover uma conta exceto uma Conta Premium, apenas um Cliente Premium pode fazê-lo. Pode retornar uma conta a partir do número da conta. Pode sacar de uma conta a partir do número da conta e depositar também, ambos deve retornar o saldo.
8. Conta Premium só pode ser criada por Cliente Premium.
9. Cliente Premium deve ser capaz de adicionar uma Conta Premium a partir de uma Conta Corrente.
10. Crie um teste unitário para testar a movimentação nas contas.