Prof. Rita Suzana Pitangueira Maciel

EXERCÍCIO DE FIXAÇÃO Nº 1

Criar uma classe em Java que represente a conta corrente de um cliente em um banco. A classe deve possuir o nome de *ContaCorrente*. Ela deve possuir os atributos número da conta, dois dígitos verificadores, e número da agencia. Possui também um atributo que identifica o grau da conta (0 - comum, 1 - especial, 2 - super e 3 - vip), um atributo que representa seu saldo, outro que representa o limite da conta (para contas com grau da conta > 0) e um atributo booleano que diz se ela está ativa ou não. Essa classe deve ser capaz de implementar as seguintes operações através dos seus métodos:

- Método construtor: deve receber como parâmetros o número da conta (sem dígitos verificadores), número da agencia, grau da conta, limite, além do depósito inicial que deve alimentar o atributo saldo. O grau da conta deve obedecer aos valores permitidos (0 a 3). Deve ser informado também como parâmetro qual o grau da conta. Para contas comuns (grau 0), implica que o depósito inicial tenha que ser maior do que 100,00. Para contas grau 1, depósito mínimo permitido de 200,00; grau 2 possui depósito mínimo de 500,00 e grau 3 possui depósito mínimo de 1.000,00.. Para valores de limites, os valores tem que ser sempre positivos e cliente comum (grau 0) deve possuir obrigatoriamente limite zero. .Se todas as condições estipuladas forem obedecidas, a conta é criada e marcada como ativa. Caso contrário, deve ser marcada como inativa e uma mensagem simples deve ser emitida dizendo qual(is) a(s) condição(ões) que não foi(ram) obedecida(s). Caso a conta seja classificada como ativa, o método construtor deve chamar os métodos para cálculo dos dígitos verificadores. Existem dois métodos: um para o primeiro dígito verificador e outro para o segundo dígito verificador.

- *int calc1DigV()*: Método para cálculo do primeiro dígito verificador. O primeiro dígito da conta multiplica-se por 1, o segundo por 2, o terceiro por 3 e assim consecutivamente até passar por todos os dígitos, para em seguida fazer a soma de todos estes valores resultantes. Depois deve subtrair dessa soma os números "noves" até que se tenha um único dígito. Exemplo: Conta 4520. Soma = 4x1 + 5x2 + 2x3 + 0x4 = 20. Subtraindo 9, fica 11 (2 dígitos). Subtraindo 9 novamente resulta em 2 (1 dígito). Então, para a conta 4520 o primeiro dígito verificador é 2. ATENÇÃO: para a codificação deste método utilize apenas *while* como estrutura de repetição, quando necessária a utilização de uma.

- *int calc2DigV()*: Método para cálculo do segundo dígito verificador. O último dígito da conta, multiplica-se por 1, o penúltimo por 2, o antepenúltimo por 3 e assim consecutivamente até chegar ao primeiro dígito que deve ser multiplicado pelo quantidade de dígitos que compõe a conta. Depois deve-se fazer a soma de todos eles. Depois deve subtrair da soma os números "noves" até que se tenha um único dígito. Exemplo: Conta 4520. Soma = 4x4 + 5x3 + 2x2 + 0x1 = 35. Subtraindo 9, fica 26 (2 dígitos). Subtraindo 9 novamente resulta em 17 (ainda 2 dígitos). Subtraindo 9 mais uma vez resulta em 8 (1 dígito). Então, para a conta 4520 o segundo dígito verificador é 8. ATENÇÃO: para a codificação deste método utilize apenas *for* como estrutura de repetição, quando necessária a utilização de uma.

- *boolean lancOp(double valorOp, char tipoOp)*: Método de lançamento de operações. Recebe como parâmetros o valor da operação e também o tipo de operação (‘D’ para débito e ‘C’ para crédito, ambos em maiúsculos). Qualquer outro código para tipo de operação o lançamento deve ser rejeitado e uma mensagem de erro deve ser emitida. Em caso de operação de débito, não é permitida operação que ocasione o saldo ultrapassar o valor do limite em número negativo, em caso do cliente não ter saldo suficiente. Por exemplo, o cliente tem um saldo negativo de 100,00 e quer debitar 200,00 possuindo um limite de 500,00. Sendo assim, o cliente ficará com o saldo negativo de 300,00 e não ultrapassaria seu limite. Nesse caso a operação seria aceita. Mas se um cliente possui saldo de 10,00 e quer fazer uma operação de débito de 200,00, tendo o seu limite estipulado em 100,00, a operação seria rejeitada, pois o cliente ficaria com um saldo negativo de 190, acima do seu limite de 100,00. Caso a operação seja aceita, o saldo deve ser calculado e atualizado no mesmo instante, e o método deve retornar verdadeiro. Caso a operação seja rejeitada, o saldo não sofre nenhuma alteração, o método deve retornar como falso e emitir uma mensagem simples explicando o motivo da rejeição.

- *void setLimite(double novoLimite)*: Método para alterar limite. Possui como parâmetro o novo valor do limite. Lembrando que os valores de limite tem que ser sempre positivos e cliente comum (grau 0) deve possuir obrigatoriamente limite zero. Além disso, o novo limite deve garantir que o saldo do cliente não esteja ultrapassando o valor do novo limite em caso de saldo negativo. Caso essas condições não sejam obedecidas, a alteração do limite não poderá ser realizada e uma mensagem simples de erro deve ser emitida.

- *void setGrau(int novoGrau)*: Método para alterar grau. Possui como parâmetro o novo grau do cliente. Só serão permitidos valores dentro dos permitidos (0 a 3). Além disso, em caso de alteração para grau 0, deve-se alterar também o limite para zero, porém sendo necessário antes verificar se o saldo do cliente é válido (maior ou igual a zero), pois cliente comum (grau 0) não possui limite. Caso essas condições não sejam obedecidas, a alteração do grau não poderá ser realizada e uma mensagem simples de erro deve ser emitida.

Faça também métodos que permitam consultar/retornar os valores de todos os seus atributos. São eles: *getNumConta(), getNumAgencia(), getDigVer1(), getDigVer2(), getGrau(), getSaldo(), getLimite(), isAtiva()*, todos ales apenas com o propósito de retornar os valores dos respectivos atributos.

ATENÇÃO: importante que o nome e assinatura de todos os métodos sejam obedecidos, assim como o nome da classe.