COM220 – Trabalho 07

Sistema para Controle de Transações Bancárias

Parte 1 [valor: **60** pts]
Deseja-se implementar um sistema de controle bancário que permita registrar as transações de **depósitos** e **saques** efetuadas em cada conta, além de **transferências** de valores realizadas entre duas contas (conta débito e conta crédito). Neste sistema, essas 3 **transações** sempre possuem uma data e um valor, que pode ser de débito ou de crédito.

Para permitir registrar o histórico das transações realizadas, cada uma das transações suportadas foi modelada como uma classe (*Saque*, *Deposito* e *Transferencia*). Como existem atributos e operações comuns a essas 3 classes, foi definida a super classe *Transacao*, conforme ilustra o modelo de dados anexo. A classe *Conta*, por sua vez, agrega por meio de um objeto ArrayList chamado **transacoes**, todas as transações realizadas. Percebe-se, portanto, que o ArrayList explora o conceito de polimorfismo para poder conter objetos diferentes.

As operações da classe *Conta* permitem criar um objeto conta e agregar o número de transações desejado por meio dos métodos *adicionaDeposito()*, *adicionaSaque()* e *adicionaTransf()*. Note que o método *adicionaTransf()* requer o parâmetro *pContaFav*, que é a conta que vai receber a transferência, ou seja, é a conta crédito ou conta favorecida (Fav). Assim, o objeto corrente é a conta débito e a conta crédito é representada pelo objeto *pContaFav*. Note, portanto, que quando é feita uma transferência da conta 1 para conta 2, deve-se criar dois objetos do tipo Transferencia: um para conter o débito na conta 1 (pTipo = “D”) e outro para conter o crédito na conta 2 (pTipo = “C”).

Outro ponto importante é que as operações de Saque e Transferência requerem o uso de uma senha válida. Além disso, só deve ser possível realizar um saque ou uma transferência caso haja saldo na conta (saldo real + limite). É por isso que os métodos *adiconaSaque*() e *adicionaTransf*() são booleanos. Caso haja problema de validação da senha, ou o saldo seja insuficiente, a operação deve falhar e o método retorna *false*. O método *CalculaSaldo*() da Classe Conta deve varrer o ArrayList de transacoes adicionando as operações de crédito e subtraindo as operações de débito. Ao final, este método retorna o saldo da conta. Note que o saldo deve levar em conta o limite, ou seja, o valor do limite sempre é computado como se fosse uma operação de crédito.

Implemente as classes do modelo de dados fornecido seguindo à risca as especificações do modelo e a descrição do sistema fornecida. Ao final use o código a seguir para testar sua implementação.

```
public class TestaAplic {
    public static void main(String args[]){
        Conta c1 = new Conta(1234, "Jose da Silva", 1000, "senha1");
        c1.adicionaDeposito(5000, new Date(), "Antonio Maia");
        if(!c1.adicionaSaque(2000, new Date(), "senha1"))
            System.out.println("Não foi possível realizar o saque no valor de 2000");
        if(!c1.adicionaSaque(1000, new Date(), "senha-errada"))   // deve falhar
            System.out.println("Não foi possível realizar o saque no valor de 1000");
        Conta c2 = new Conta(4321, "Joao de Deus", 1000, "senha2");
        //-------
        c2.adicionaDeposito(3000, new Date(), "Maria da Cruz");
        if(!c2.adicionaSaque(1500, new Date(), "senha2"))
            System.out.println("Não foi possível realizar o saque no valor de 1500");
        if(!c2.adicionaTransf(5000, new Date(), "senha2", c1))    // deve falhar
            System.out.println("Não foi possível realizar a transf no valor de 5000");
        if(!c2.adicionaTransf(800, new Date(), "senha2", c1))
            System.out.println("Não foi possível realizar a transf no valor de 800");
        //------
        System.out.println("---------");
        System.out.println("Saldo de c1 = " + c1.calculaSaldo());  // deve imprimir 4800
        System.out.println("Saldo de c2 = " + c2.calculaSaldo());  // deve imprimir 1700
    }
}
```

Parte 2 [valor: **40** pts]
Utilizando o modelo MVC, implemente as classes limiteConta e controleConta, a fim de prover as seguintes operações:

1. [valor: 10 pts] Criar contas
2. [valor: 12 pts] Realizar transações nas contas (saques, depósitos e transferências)
3. [valor: 18 pts] Imprimir o extrato de uma conta, dado o seu nro. O extrato deve conter o saldo da conta.

É permitido acrescentar novos métodos nas classes do modelo, se necessário.

Importante: Criar um projeto para cada parte da prova.