Atividade continuada parte 2 – inserção do dao genérico nos repositórios específicos

ATENÇÃO: Os pacotes das classes a serem criadas, e mais alguns detalhes estão BEM CLAROS nos testes automatizados.

Classes a serem implementadas ou alteradas	O que ela faz
RepositorioGeral e repositórios específicos	RepositorioGeral é a mãe dos repositórios específicos. É abstrata, tem um método abstrato Class<?> getClasseEntidade() e um atributo dao, do tipo DAOSerializadorObjetos. Por herança, os repositórios específicos podem usar este atributo dao para realizar as operações de inclusão, exclusão, alteração e busca. Os repositórios específicos devem implementar o método abstrato getClasseEntidade(), retornado o class da entidade (exemplo: Transacao.class). O construtor desta classe deve ser vazio, e inicializar o atributo dao com uma instância de DAOSerializadorObjetos. Como o construtor de DAOSerializadorObjetos recebe um Class, a chamada dele deve passar o retorno do método getClasseEntidade().
Entidade, Transacao, Ativo, EntidadeOperadora	Entidade é uma casse abstrata, e tem um método abstrato String getIdUnico(). Deve ser a mãe de Ativo, Transacao e EntidadeOperadora. Estas classes devem implementar tal método, retornando cada uma o seu id único. O id único de Ativo e de EntidadeOperadora é o identificador. O id único de Transacao é a concatenação de: Id único da entidade de crédito + "_" + Id único da entidade de débito + "_" + Id único da ação OU id único de título da dívida + "_" + Data e hora da operação formatada como yyyymmddhhmmss

Atividade continuada parte 3 – relatórios

Esta parte prevê a implementação de dois relatórios, cujos resultados são dois arrays de transações, retornados por dois métodos de uma nova classe a ser criada: RelatorioTransacaoBroker. Os nomes dos dois métodos estão bem claros nos testes automatizados t006 e t007 da classe TesteRelatorios. Estes arrays são bem simples de serem gerados, pois são todas as transações gravadas (ou seja, o retorno de uma busca geral), só que ordenados por nome da entidade de crédito e por data hora operação (descubram olhando os testes t006 e t007 se a ordem é crescente ou decrescente, é só olhar as transações previamente gravadas pelos testes e o resultado esperado dos testes).

Para realizar estas duas ordenações, a proposta é criar um esquema próprio de ordenação de arrays, em vez de usar o esquema já pronto do JAVA que ordena listas. Para isso, devem ser criadas algumas classes e interfaces.

ATENÇÃO: Os pacotes das classes a serem criadas, e mais alguns detalhes estão BEM CLAROS nos testes automatizados.

Classes e interfaces a serem implementadas	O que ela faz
Comparavel	É uma interface similar a Comparable do JAVA Core. Tem um único método que prevê a comparação do Comparavel atual com outro recebido como parâmetro. int comparar(Comparavel c). Se o elemento atual for maior que o parâmetro, o método deve retornar um valor maior que zero, se for menor, o método deve retornar um valor menor que zero, e se for igual, zero.
Comparador	É uma interface similar a Comparator do JAVA Core. Tem um único método que prevê a comparação de dois Comparavel recebidos como parâmetro. int comparar(Comparavel c1, Comparavel c2). Se o primeiro parâmetro for maior que o segundo parâmetro, o método deve retornar um valor maior que zero, se for menor, o método deve retornar um valor menor que zero, e se for igual, zero.
Ordenador	É uma classe que implementa o algoritmo de ordenação de arrays de comparáveis. Tem dois métodos estáticos: public static void ordenar(Comparavel[] ents, Comparador comp) public static void ordenar(Comparavel[] comps) O primeiro ordena o array de entrada perguntando ao comparador se dois elementos são maiores, menores ou iguais. O segundo ordena o array de entrada perguntando a um elemento se ele é maior, menor ou igual a outro elemento.
ComparadorTransacao PorNomeCredora	Herda de ComparadorPadrao. Deve implementar Comparador, e o critério de comparação que ela deve usar é o de nome da entidade credora.
Transacao	Deve implementar Comparavel, e o critério de comparação que ela deve usar é o de data e hora da operação.
RelatorioTransacaoBroker	Tem os dois métodos mencionados acima, que devem ler do repositório de transações todas as transações e ordenar o retorno desta leitura por nome da entidade credora e por data hora da operação, retornando os respectivos arrays ordenados.