package br.edu.cesarschool.next.oo.negocio;

import java.util.Arrays;

import java.util.Collection;

import java.util.Collections;

import java.util.List;

import br.edu.cesarschool.next.oo.dao.DAOContaCorrente;

import br.edu.cesarschool.next.oo.entidade.ContaCorrente;

import br.edu.cesarschool.next.oo.entidade.ContaPoupanca;

/\*

\* Métodos:

Construtor padrão.

String incluir(ContaCorrente conta)

Deve validar os dados da conta corrente recebida e, se algum dado estiver inválido, retornar

mensagem pertinente. As seguintes validações devem ser feitas:

• numero: dever ser diferente de nulo e de branco, e ter tamanho mínimo de 5

caracteres e máximo de 8 caracteres.

• saldo: deve ser maior ou igual a zero.

• nomeCorrentista: dever ser diferente de nulo e de branco, e ter tamanho máximo de

60 caracteres.

• Se a conta corrente recebida for uma conta poupança, validar o percentual de bônus

que deve ser maior ou igual a zero.

Se os dados estiverem válidos, incluir a conta corrente no mecanismo de persistência usando

o DAO. O retorno false do incluir do DAO indica que a chave da conta corrente (o seu

número) já existe. Se isto ocorrer, retornar mensagem pertinente.

Caso a inclusão seja realizada com sucesso, retornar null.

\*

\*

\*

\*

\*

\*/

public class MediatorContaCorrente {

private DAOContaCorrente daoContaCorrente = new DAOContaCorrente();

public MediatorContaCorrente(){

}

public MediatorContaCorrente(DAOContaCorrente daoContaCorrente) {

this.daoContaCorrente = daoContaCorrente;

}

public boolean stringNulaOuVazia(String dado){

if(dado == null || dado.trim().equals("")){

return false;

} else{

return true;

}

}

public String incluir(ContaCorrente conta) {

if (conta == null) {

return "Conta não encontrada";

} else if (stringNulaOuVazia(conta.getNumero())) {

return "Conta nula ou vazia";

} else if (conta.getNumero().length()<5 || conta.getNumero().length()>8) {

return "Numero da Conta menor que 5 e maior que 8";

} else if (conta.getSaldo() < 0) {

return "Saldo Negativo ";

} else if(stringNulaOuVazia(conta.getNomeCorrentista())) {

return "Nome nulo ou vazio";

}else if(conta.getNomeCorrentista().length() > 60){

return "Nome muito grande " ;

}else if(conta instanceof ContaCorrente){

ContaPoupanca contaP = (ContaPoupanca)conta;

if(contaP.getPercentualBonus()<0){

return "Conta Poupança Sem percentual de Bonus ";

} else {

return null;

}

} else {

boolean ret = daoContaCorrente.incluir(conta);

if(!ret){

return "Conta Já Existente!!";

} else {

return null;

}

}

}

/\*

\* String creditar(double valor, String numero)

Deve validar os dados recebidos. se algum dado estiver inválido, retornar mensagem

pertinente. As seguintes validações devem ser feitas:

• valor tem que ser maior ou igual a zero.

• numero tem que ser diferente de nulo e de branco.

O método deve buscar no DAO a conta corrente por número. Se ela não existir, retornar

mensagem pertinente. Se ela existir, creditar, usando o método da conta corrente, valor na

conta, alterar, no DAO, a conta, e retornar null.

\*/

public String creditar(double valor, String numero){

if(valor < 0){

return "Valor Invalido!!";

} else if (stringNulaOuVazia(numero)){

return "Numero Não Informado!!";

} else {

ContaCorrente conta = daoContaCorrente.buscar(numero);

if(conta == null){

return "Conta Não encontrada!!!";

} else {

conta.creditar(valor);

daoContaCorrente.alterar(conta);

return null;

}

}

}

/\*

\* String debitar(double valor, String numero)

Deve validar os dados recebidos. se algum dado estiver inválido, retornar mensagem

pertinente. As seguintes validações devem ser feitas:

• valor tem que ser maior ou igual a zero.

• numero tem que ser diferente de nulo e de branco.

O método deve buscar no DAO a conta corrente por número. Se ela não existir, retornar

mensagem pertinente. Se ela existir, verificar se o saldo atual é menor que valor, se for,

retornar mensagem pertinente. Caso contrário, debitar, usando o método da conta corrente,

valor na conta, alterar, no DAO, a conta, e retornar null.

\*/

public String debitar(double valor, String numero){

if(valor < 0){

return "Valor Invalido!!";

} else if (stringNulaOuVazia(numero)){

return "Numero Da conta Invalido!!!";

} else {

ContaCorrente conta = daoContaCorrente.buscar(numero);

if(conta == null){

return "Conta não Encontrada";

} else {

if(conta.getSaldo() < valor){

return "Saldo Insuficiente";

} else{

conta.debitar(valor);

daoContaCorrente.alterar(conta);

return null;

}

}

}

}

/\*

\* ContaCorrente buscar(String numero)

Se numero for nulo ou branco, retornar null. Caso contrário, realizar busca por número no

DAO e retornar o retorno desta busca.

\*/

public ContaCorrente buscar(String numero){

if(stringNulaOuVazia(numero)){

return null;

} else {

return daoContaCorrente.buscar(numero);

}

}

/\*

\* List<ContaCorrente> gerarRelatorioGeral()

Implementar tal método seguindo o modelo implementado no exemplo de Produto. A lista

de contas correntes retornada deve estar ordenada por saldo, em ordem crescente. Para

isto, usar o ComparadorContaCorrenteSaldo, já implementado.

\*/

public List<ContaCorrente> gerarRelatorioGeral(){

ComparadorContaCorrenteSaldo comparador = new ComparadorContaCorrenteSaldo()

ContaCorrente[] contas = daoContaCorrente.buscarTodos();

List<ContaCorrente> listaConta = Arrays.asList(contas);

Collections.sort(listaConta, comparador);

return listaConta;

}

}