Projeto e Construção de Sistemas

Lista de Exercícios 5 - Exceções

Exercício 1:

Sejam os seguintes elementos:

```
public interface ProdutoDAO {
      void armazenar(Produto p) throws DAOException;
      Produto recuperar(String codigoProduto) throws DAOException;
}

public class Exemplo {
      public void salvarProduto(Produto p, ProdutoDAO dao) {
            dao.armazenar(p);
      }
}
```

ProdutoDAO é uma interface que define duas operações:

- a operação **armazenar** que é responsável por armazenar um objeto **Produto** recebido como parâmetro.
- A operação **recuperar** que é responsável por recuperar um objeto **Produto** cujo código seja igual ao parâmetro codigoProduto.

Parte 1:

Implemente uma classe (ProdutoDAOMemory) que implementa as operações definidas na interface ProdutoDAO de forma que os produtos fique armazenados em um array de produtos com capacidade máxima de 1000 produtos.

- Nesta implementação, o armazenamento dos produtos deve ser feito em um array com capacidade máxima de 1000 produtos. Caso não exista um produto com o código do produto recebido como parâmetro, a operação deverá armazenar a referência ao produto na próxima entrada vaga do array. Caso já exista um produto com o código do produto recebido como parâmetro, a operação deverá atualizar a referência na posição onde esse produto com mesmo código foi encontrada. Caso não haja mais posições livre no array, essa operação arremessa uma exceção do tipo **DAOException**.
- Na operação recuperar, a implementação deve percorrer o array em busca de um produto com o mesmo código passado como parâmetro. Se não for encontrado nenhum produto correspondente a esse código, a operação deve arremessar uma exceção do tipo **DAOException**.

Parte 2:

A classe **Exemplo** possui uma operação **salvarProduto,** que recebe um **Produto p** e um **ProdutoDAO** dao, e solicita o armazenamento do produto **p** ao objeto **dao**. O código acima não está correto devido a um problema referente ao tratamento de exceções. Para consertá-lo, considere os cenários descritos a seguir:

a) Suponha que a operação **salvarProduto,** no caso em que a operação **armazenar** arremesse uma exceção, não efetue nenhum tratamento local dessa exceção e simplesmente repasse essa exceção para a operação que a chamou (para o chamador da operação **salvarProduto**, portanto). Reescreva o código da classe Exemplo para essa situação. Escreva um pequeno programa (operação main) que chame a operação **salvarProduto** nesta situação.

b) Agora suponha que a operação **salvarProduto,** caso a operação **armazenar** arremesse uma exceção, imprima no console uma mensagem de erro ("erro ao tentar armazenar o produto") e repasse essa mesma exceção para a operação que a chamou (para o chamador da operação **salvarProduto**, portanto). Reescreva o código da classe Exemplo para essa situação. Escreva um pequeno programa (operação main) que chame a operação **salvarProduto** nesta situação.

c) Agora suponha que a operação **salvarProduto,** caso a operação **armazenar** arremesse uma exceção, apenas imprima no console uma mensagem de erro ("erro ao tentar armazenar o produto") sem repassar essa exceção para o seu chamador (para chamador da operação **salvarProduto**, portanto). Reescreva o código da classe Exemplo para essa situação. Escreva um pequeno programa (operação main) que chame a operação **salvarProduto** nesta situação.

d) Agora suponha que a operação **armazenar** possa arremessar dois tipos de exceção: DAOException e SQLException. Suponha também que a operação **salvarProduto,** caso a operação **armazenar** arremesse uma exceção DAOException, apenas imprima no console uma mensagem de erro ("erro ao tentar armazenar o produto") sem repassar essa exceção para o seu chamador (para o chamador da operação **salvarProduto**, portanto). Caso a operação **armazenar** arremesse uma exceção SQLException, a operação **salvarProduto** imprime no console uma mensagem de erro ("erro no acesso ao banco de dados") e repassa essa exceção para a operação que a chamou (para o chamador da operação **salvarProduto**, portanto). Reescreva o código da classe Exemplo para essa situação. Escreva um pequeno programa (operação main) que chame a operação **salvarProduto** nesta situação.

Exercício 2:

Passo 1: Construir as classes Conta e BancoAppException

Classe Conta:
Atributos:

- numero (string)
- saldo (double)

Operações:

- Construtor com os parâmetros numero da conta e saldo inicial.
- Operação: debitar (parâmetro: valor) -> debita valor no saldo da conta.
- Operação: creditar (parâmetro: valor) -> acumula valor do saldo da conta.

Considere as seguintes regras:

a) debitar:
    - valor a debitar tem que ser um número positivo maior que zero.
    - conta tem que ter saldo suficiente para o débito, isto é, a conta tem que ter um valor maior ou igual a zero após o débito.
    - Existe um limite para o valor a debitar que é de 2.000,00. O limite é por operação de débito (suponha que não exista limite diário).

b) creditar:
    - valor a creditar tem que ser um número positivo maior que zero.

Em caso de falha nessas regras, a implementação da operação (debitar ou creditar) deve arremessar uma exceção do tipo *BancoAppException* (que deve ser criada por você, estendendo a classe *Exception* já existente no Java) com uma mensagem de erro correspondente ao problema.

ATENÇÃO: Defina as classes Conta e BancoAppException no pacote banco.negocio

Passo 2: Construir a classe BancoUI (interface com o usuário)

Faça uma classe BancoUI que corresponderá a uma interface com usuário rudimentar via console que deverá se comportar da seguinte forma:

A classe BancoUI deverá instanciar o objeto Conta que será manipulado durante a execução do programa. Antes de instanciar o objeto Conta, porém, BancoUI deverá perguntar para o usuário o número e o saldo inicial que serão passados para o construtor do objeto Conta.

A operação do programa será baseada em um menu textual em console (não se preocupem com coisas do tipo limpar a tela, e etc), com as seguintes opções:
1 – creditar na conta
2 – debitar da conta
3 – consultar saldo
4 – finalizar o programa

Se o usuário entrar a opção 1, o programa deverá perguntar o valor a ser creditado. O usuário entrará o valor e a classe BancoUI chamará diretamente a operação creditar com o valor passado. Caso haja uma exceção, o programa deverá exibir a mensagem de erro correspondente. Caso contrário o programa deverá apresentar a mensagem "Operação bem sucedida".

Se o usuário entrar a opção 2, o programa deverá perguntar o valor a ser creditado. O usuário entrará o valor e a classe BancoUI chamará diretamente a operação creditar com o valor passado. Caso haja uma exceção, o programa deverá exibir a mensagem de erro correspondente. Caso contrário, o programa deverá apresentar a mensagem "Operação bem sucedida".

Se o usuário entrar a opção 3, o programa deverá apresentar o valor atual do saldo da conta.

Após executar qualquer destas três opções, o programa deverá apresentar novamente o menu de opções.

ATENÇÃO: Defina a classe BancoUI no pacote banco.ui.

Passo 3: Construir a classe BancoApp (classe com main do programa)

Crie uma terceira classe BancoApp que conterá apenas a operação main e a instanciação e execução de BancoUI.

ATENÇÃO: Defina a classe BancoApp no pacote banco.app.

OBS:
1) Os valores a debitar ou a creditar serão capturados da forma acima como um String e deverão ser convertidos para Double através da chamada:

```
Double.parseDouble(stringEntradoPeloUsuario);
```

Atenção que se o string entrado pelo usuário não for um número, a operação parseDouble arremessará a exceção *NumberFormatException*, que deverá ser tratada pelo programa (ex: apresentando mensagem de número inválido).