| | |
|---|---|
| UFABC | Universidade Federal do ABC<br>Disciplina: Programação Orientada a Objetos<br>2Q-2019 |

## Atividade 7: Tratamento de exceções

Obs.: Para entrega: a pasta src com os arquivos .java de cada um dos projetos (os exercícios relacionados podem estar no mesmo projeto). Renomeie cada pasta de modo a identificar o exercício correspondente. Submeta uma pasta compactada na atividade do Tidia.

## Exercício 1

Vamos modificar o exercício 1 da aula passada (figuras planas) para incluir o tratamento de exceções. Crie classes de exceções para tratar erros que podem ser gerados na validação de figuras:

1) Verificar se é um triângulo válido (exceção do tipo checked). Escreva a classe **TrianguloInvalidoException**.
2) Informar valores negativos para os lados em qualquer figura (exceção do tipo unchecked). Escreva a classe **ValorNegativoException**.

Modifique os métodos para lançar as exceções específicas (usando throw) e faça a captura e tratamento das exceções no programa principal (try-catch).

## Exercício 2 (Prof. Paulo Pisani)

O professor ABC escreveu a classe **PilhaSimples** (código disponível no site da disciplina), mas ele esqueceu de verificar algumas situações: empilhar quando o vetor está cheio e desempilhar quando a pilha está vazia.
Um aluno de POO sugeriu implementar essa verificação usando **exceções unchecked**. Para isso, haverá uma classe abstrata **PilhaException** e duas subclasses: **PilhaCheiaException** e **PilhaVaziaException**.

A classe **PilhaCheiaException** deve armazenar o limite da pilha e qual foi o elemento não empilhado.
O professor desafiou o aluno a fazer essas verificações sem modificar o código da classe **PilhaSimples**. Para isso, outro aluno sugeriu criar uma classe chamada **PilhaAprimorada**, que estende **PilhaSimples.**
Utilize a classe **TesteEstruturas** como programa principal (disponível na própria atividade).

## Exercício 3 (Prof. Paulo Pisani)

O diretor de uma empresa percebeu que seus funcionários sempre usavam pilhas e, após o seu uso, esqueciam essas pilhas com elementos empilhados;
Cansado disso, ele pediu para um aluno da disciplina de POO criar uma classe **PilhaRecurso**, que estende **PilhaAprimorada** e pode ser usada com o **try with resources**:

```
try (PilhaRecurso pilha = new PilhaRecurso(5)) {
      pilha.empilha("Teste1");
      pilha.empilha("Teste2");
      System.out.println(pilha.desempilha());
}
```

## Exercício 4

Neste exercício vamos trabalhar com leitura e escrita em arquivo (o código está disponível na própria atividade). O arquivo de dados chama-se "funcionarios.txt" e contém os dados de funcionários (código do funcionário, nome e salário) separados por um ponto-e-vírgula (abra o arquivo para visualizar o seu formato). Crie um novo projeto e copie este arquivo na pasta raiz do projeto.

Temos as classes **Funcionario** (para representar um funcionário) e a classe **LeitorEscritorAquivo** que contém métodos para a leitura e escrita de dados no arquivo. O método **leDados** obtém cada linha (String) do arquivo, separa os campos (código, nome e salário) e em seguida cria um objeto do tipo funcionario com esses valores, armazenando-o em um vetor. O método **salva** recebe um objeto funcionario e escreve os valores de seus atributos no arquivo.

No programa principal, temos um vetor de funcionários que é preenchido na primeira linha do programa (método **carregaDados**).
O programa oferece algumas operações para cadastro e remoção de funcionários, mas está incompleto (a opção para remover funcionário não está implementada).
Inicialmente, teste o programa como está para imprimir os dados dos funcionários e cadastrar novos funcionários (veja as modificações que ocorrem no arquivo ao cadastrar).

Complemente o programa com as seguintes funcionalidades:

1. Crie um método **busca(int id)** que recebe o id de um funcionário e retorna o índice em que o funcionário está posicionado no vetor (caso esteja no vetor) ou, -1, caso contrário.

2. Vamos acrescentar algumas classes de exceções para tratar os seguintes problemas:

- O vetor de funcionários está cheio, não permite cadastrar um novo funcionário. Escreva a classe de exceção **LimiteAtingidoException**.
- O arquivo de funcionários está vazio, não permite carregar dados no vetor (simule esta situação removendo todos os dados do arquivo "funcionarios.txt" e salvando-o). Escreva a classe de exceção **NaoHaFuncionariosCadastradosException**. Esta exceção pode ser lançada também ao tentar imprimir o vetor vazio.
- Um funcionário não está no vetor (método busca do item 1). Escreva a classe **FuncionarioNaoCadastradoException** com o atributo inteiro id para especificar na mensagem de erro o código (id) do funcionário solicitado.

   Modifique os métodos para lançar as repectivas exceções e faça o tratamento das exceções na chamada dos métodos.

3. Tratar o caso em que o usuário digita uma opção inválida no menu inicial, por exemplo um texto não numérico (dica: capture e trate a exceção **NumerFormatException**).
4. Implementar o método para remover um funcionário. Para isso, é necessário remover o funcionário do vetor (e se necessário fazer um ajuste para mover os elementos para preencher o espaço vazio) e, também é preciso remover o funcionário do arquivo.
   Sugestão: como não há um método para apagar uma linha específica de um arquivo, remova o arquivo (veja o exemplo abaixo) e reescreva os dados do vetor atualizado no arquivo.

```
File file = new File(arquivo);
file.delete();
```

Obs.: O método de remoção deve lançar uma exceção caso o funcionário não esteja cadastrado no vetor.