Programação Orientada a Objetos

Prof. Paulo Henrique Pisani

Prof. Saul de Castro Leite

http://professor.ufabc.edu.br/~paulo.pisani/

UFABC

julho/2018

Exercício 1 (a) - Contas

20 min

Sugestão de Ricardo Drudi

- Ricardo escreveu o código de um programa para gerenciar uma conta corrente (código disponível no site da disciplina);
- Implemente a classe Conta (utilizada no main):
  - O método sacar pode lançar uma exceção SaldoInsuficienteException, caso o saldo disponível para saque seja inferior ao valor a ser sacado;
  - **Saldo para saque = saldo + limite**;
  - A exceção SaldoInsuficienteException armazena o saldo disponível para saque (**saldo + limite**);
  - A classe desta exceção, que é do tipo **checked**, também precisa ser implementada.

UFABC

Exercício 1 (a) - Contas

Sugestão de Ricardo Drudi

- A saída do programa deverá ser:

```
Pessoa teste tem 80.0
Pessoa teste tem -100.0
O saldo da conta eh insuficiente!
O saldo para saque eh 0.0
```

- **Não modifique o arquivo PrincipalConta.java**

UFABC

Exercício 1 (b) - Contas

Sugestão de Ricardo Drudi

- Versão (b) do exercício está disponível no site da disciplina.

UFABC

Exercício 2

25 min

- 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.

UFABC

Exercício 2 (continuação)

- 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 no site da disciplina).

UFABC

Exercício 3

20 min

- O professor escreveu a classe FilaSimples (código disponível no site da disciplina) e também esqueceu de colocar as verificações de fila cheia/vazia;
- **Implemente as verificações na fila de forma similar a pilha**: estendendo a classe FilaSimples e criando uma hierarquia de exceções.

UFABC

Exercício 4

10 min

- 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 resourses*:

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

UFABC

Exercício 5

5 min

- O gerente percebeu o mesmo problema observado pelo diretor no exercício anterior, mas desta vez com o uso da fila.
- Implemente a classe FilaRecurso seguindo o mesmo princípio.

UFABC

Exercício 6

Para fazer uma expedição, Vikings embarcam em seus navios da seguinte forma: primeiro entra o Timoneiro, responsável por guiar o barco; em seguida, 18 remadores, e por último o líder da expedição. Ao desembarcar, o processo é ao contrário, iniciando com o líder e terminando com o timoneiro. Em cada processo de embarcar e desembarcar, cada Viking grita sua posição no navio seguido por “ARRGH”. Implemente a estrutura de classes contendo uma classe abstrata viking e subclasses timoneiro, remador e líder. Implemente uma classe navio que permita que os vikings embarquem e desembarquem seguindo o protocolo descrito acima. Use a estrutura de dados PilhaAprimorada.

UFABC

Exercício 7

Use a PilhaAprimorada do exercício 2 para implementar o jogo Torre de Hanoi. Este jogo é composto por 3 pilhas. A primeira pilha contém um número n de discos. O objetivo do jogo é mover todos os discos da pilha inicial para outra pilha qualquer seguindo as seguintes regras:

1. Somente um disco pode ser movido entre as pilhas por vez;
2. Somente o disco do topo pode ser movido;
3. Os discos menores devem estar sempre encima dos discos maiores.

Você deve implementar uma classe Disco, que contém um atributo com seu tamanho. O programa principal deve exibir o estado das pilhas e solicitar o jogador sua movimentação usando o STDIN.

Desafio: Implemente um algoritmo recursivo para solucionar o problema automaticamente.

UFABC