Universidade Federal de Minas Gerais
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
Disciplina: Algoritmos e Estruturas de Dados II (DCC004)
Professor: Renato Martins (renato.martins AT dcc.ufmg.br)
https://www.dcc.ufmg.br/~renato.martins/courses/DCC004
2° Semestre de 2018

# Lista 3 – Refatoração, Programação Defensiva, Tratamento de Exceções e Testes

Data de publicação: 20/11/2018
**Data de Entrega: 03/12/2018** (via email)

**Importante:** A linguagem para implementação dos exercícios deve ser C++. Crie um projeto para cada exercício (quando apropriado), que deve conter um programa principal e o seu respectivo Makefile. O seu Makefile deve permitir a compilação, execução de testes e execução do programa principal respectivamente com os comandos `make`, `make testes` e `make run`. A pontuação de cada item é indicada antes da questão, totalizando NP = 10 pontos.

1. **[1 pt]** Localize os erros na seguinte classe e explique como corrigi-los:

```
class Exemplo {
  public:
    Exemplo( int y = 10 ) : dado( y ) {}
    int getDadoIncrementado() const {
      return dado++;
    }
    static int getContador() {
      return contador;
    }
  private:
    int dado;
    static int contador;
};
```

2. **[1 pt]** Modifique o código abaixo para que o mesmo lance uma exceção quando acontecer uma divisão por zero.

```
#include <iostream>
using namespace std;

double divisao(int a, int b) {
    return (double) a / b;
}

int main() {
   double d = divisao(4, 0);
   cout << d << endl;
}
```

**3.** [**1 pt**] Escreve o código para lançar um objeto de exceção `EntradaInvalida` se o número recebido pela função fatorial for maior do que 20. Você deve definir a classe e implementar o código necessário.

```
#include <iostream>
using namespace std;

class EntradaInvalida {
};

int fatorial(int numero){
   int fat = 1;
   int n = 1;
   while (++n <= numero)
     fat *= n;
   return fat;
}

int main(){
  cout << fatorial(10) << endl;
}
```

**4.** [**1 pt**] Modifique o programa para que ele tenha outra exceção chamada `EntradaReprovadoExcept`
Se o usuário fornecer uma entrada menor do que 60, o programa deve exibir uma mensagem que diz que notas que reprovam não são permitidas, e então continuar a aceitar notas. A frequência deve ser revisada de forma que somente quatro sejam exibidas.

```
#include <iostream>

using namespace std;

class EntradaNegativaException {
  public:
    EntradaNegativaException() {
      cout << "Fim da entrada de dados.\n";
    }
};

int main() {
  int nova_nota, indice, limite_1, limite_2;
  int freq[10] = {0};
  try {
    while (true) {
      cout << "Digite uma nota ou -1 para terminar: ";
      cin >> nova_nota;
      if (nova_nota < 0){
        throw EntradaNegativaException();
      }
      indice = nova_nota / 10;
      try {
        if (indice > 9)
          throw nova_nota;
        freq[indice]++;
      }
```

```
28        catch (int nota) {
29          if (nota == 100)
30            freq[9]++;
31          else
32            cout << "Erro: " << nota << " fora do limite!\n";
33        }
34      }
35    }
36    catch (EntradaNegativaException e) {
37      cout << "Frequencia dos limites\n";
38      for (indice = 0; indice < 10; indice++) {
39        limite_1 = 10 *indice;
40        limite_2 = limite_1 + 9;
41        if (indice == 9)
42          limite_2 = 100;
43        cout << limite_1 << " " << limite_2 << " " << freq[indice] <<
              endl;
44      }
45    }
46  }
```

**5.** [**1 pt**] O código abaixo ordena uma lista de retas pela inclinação. Veja o comportamento caso alguma das retas tenha inclinação infinita, e observe que a exceção se propaga através de várias chamadas de funções aninhadas. Reescreva esse código *sem exceções* para que ele tenha o mesmo comportamento. Tente preservar o máximo possível da estrutura do código.

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <math.h>
5
6  class Ponto {
7    public:
8      Ponto(double x_, double y_) : x(x_),y(y_) {}
9      double x,y;
10 };
11
12 class Reta {
13   public:
14     Reta(const Ponto &a_, const Ponto &b_) : a(a_),b(b_) {}
15     Ponto a,b;
16 };
17
18 double computa_inclinacao(const Ponto &a, const Ponto &b) throw(int
       ) {
19   double eixo_y = b.y - a.y;
20   double eixo_x = b.x - a.x;
21   double epsilon = 0.00001;
22   if (fabs(eixo_x) < epsilon) throw -1;
23   return eixo_y / eixo_x;
24 }
25
26 double inclinacao(const Reta &reta) {
27   return computa_inclinacao(reta.a,reta.b);
```

```
28 }
29
30 bool inclinacao_maior(const Reta &reta1, const Reta &reta2) {
31   double inclinacao_r1 = inclinacao(reta1);
32   double inclinacao_r2 = inclinacao(reta2);
33   return inclinacao_r1 > inclinacao_r2;
34 }
35
36 void ordena(std::vector<Reta> &retas) {
37   std::sort(retas.begin(),retas.end(), inclinacao_maior);
38 }
39
40 int main () {
41   std::vector<Reta> retas;
42   // codigo para inicializar os dados omitido
43   try {
44     ordena(retas);
45     // codigo para imprimir os resultados omitido
46   } catch (int) {
47     std::cout << "erro: inclinacao infinita" << std::endl;
48   }
49 }
```

**6.** [**2 pts**]

Utilizando a metodologia TDD, escreva testes de unidade para um método que determina se um triângulo é Equilátero, Isósceles, Escaleno. O método pertence a uma classe `Triangulo` e possui a seguinte assinatura: `int determinar_tipo()`. O retorno do método é um inteiro representando os casos: Equilátero (0), Isósceles (1), Escaleno (2). A classe deve possuir um construtor que recebe 3 parâmetros do tipo double (representando as dimensões dos lados). O construtor deve tratar casos excepcionais (i.e., triângulos inválidos), por exemplo, lados negativos ou zero e desigualdade triangular não atendida. Em todos esses possíveis casos de excepcionais, deve ser lançada uma exceção chamada `TrianguloInvalidoError`, que deve ser subclasse de `std::exception`. Escreva um programa principal com pelo menos 4 casos de testes (TEST_CASES). Lembre-se que o ciclo TDD consiste de:

(1) Escrever teste

(2) Ver o teste falhar

(3) Fazer o teste passar

(4) O código tem odores (code smells)? Se sim, passe para o passo (5). Se não, break.

(5) Refatorar

(6) Volte para o passo (1)

**7.** [**3 pts**] Considere um tabuleiro de xadrez com as seguintes peças: `Peao, Cavalo, Bispo, Torre, Rainha` e `Rei`. O seu tabuleiro de xadrez será composto de 8 peões, 2 cavalos; 2 bispos; 2 torres, 1 rainha e 1 rei. O movimento das peças de xadrez devem seguir as seguintes restrições: `https://www.baquara.com/xadrez/`

`movimentos.htm`. Queremos realizar a implementação desse tabuleiro em C++ em que todas as classes de peças tenham:

- Um método `get_posX()` que retorne a posição da peça no eixo x (inteiro entre [0, 7]) do tabuleiro
- Um método `get_posY()` que retorne a posição da peça no eixo y (inteiro entre [0, 7]) do tabuleiro
- Um método `get_nome()` que retorne o nome da peça (string)
- Um método `bool pode_mover(int x, int y)` que indica se um movimento do local atual para x, y é válido.
- Um método `void move(int x, int y)` que realiza o movimento. O mesmo deve invocar `pode_mover` e lançar uma exceção quando o movimento for inválido.

**(a)** Implemente uma hierarquia de classes em que cada uma das classes de peças (`Peao, Cavalo, Bispo, Torre, Rainha`, `Rei`) sejam derivadas de uma classe abstrata `Peca`. A classe `Peca` tem atributos `int posicaoX, int posicaoY` e `string _nomePeca` e os métodos virtuais puros `boolean podeMover(int x, int y)` e `void move(int x, int y)`. Os métodos `get_posX()`, `get_posY()` e `get_nome()` são implementados na classe `Peca`

**(b)** Implemente a classe `MovimentoInvalidoException` que seja derivada da classe `std::exception`. O método `what()` dessa classe deve mostrar a seguinte mensagem: `"A peca <_nomePeca> nao pode efetuar esse movimento"` em que ¡_nomePeca¿ seja a string do atributo `_nomePeca`.

**(c)** Use a metodologia TDD para criar 6 arquivos testes de unidade usando a biblioteca `doctest.h`. Cada arquivo deve cobrir uma das peças do jogo de forma que sejam respeitadas as restrições de movimento de cada peça.

**(d)** Crie uma classe `Tabuleiro` que guarda para cada peça um `std::map<Posicao, Peca *>`. O construtor da mesma deve iniciar as peças na posição correta. Por fim, o `Tabuleiro` deve conter um método: `move(Posicao, Posicao)` que muda uma peça de uma dada posição para outra. Tal método pode lançar dois tipos de exceções: `MovimentoInvalidoException` ou `PosicaoSemPecaException`. Implemente a classe `PosicaoSemPecaException` que seja derivada da classe `std::exception`. O método `what()` dessa classe deve mostrar a seguinte mensagem: `"Esta posicao do tabuleiro nao tem peca para ser movimentada"`.