Exceções II

BCC 221 - Programação Orientada a Objectos(POO)

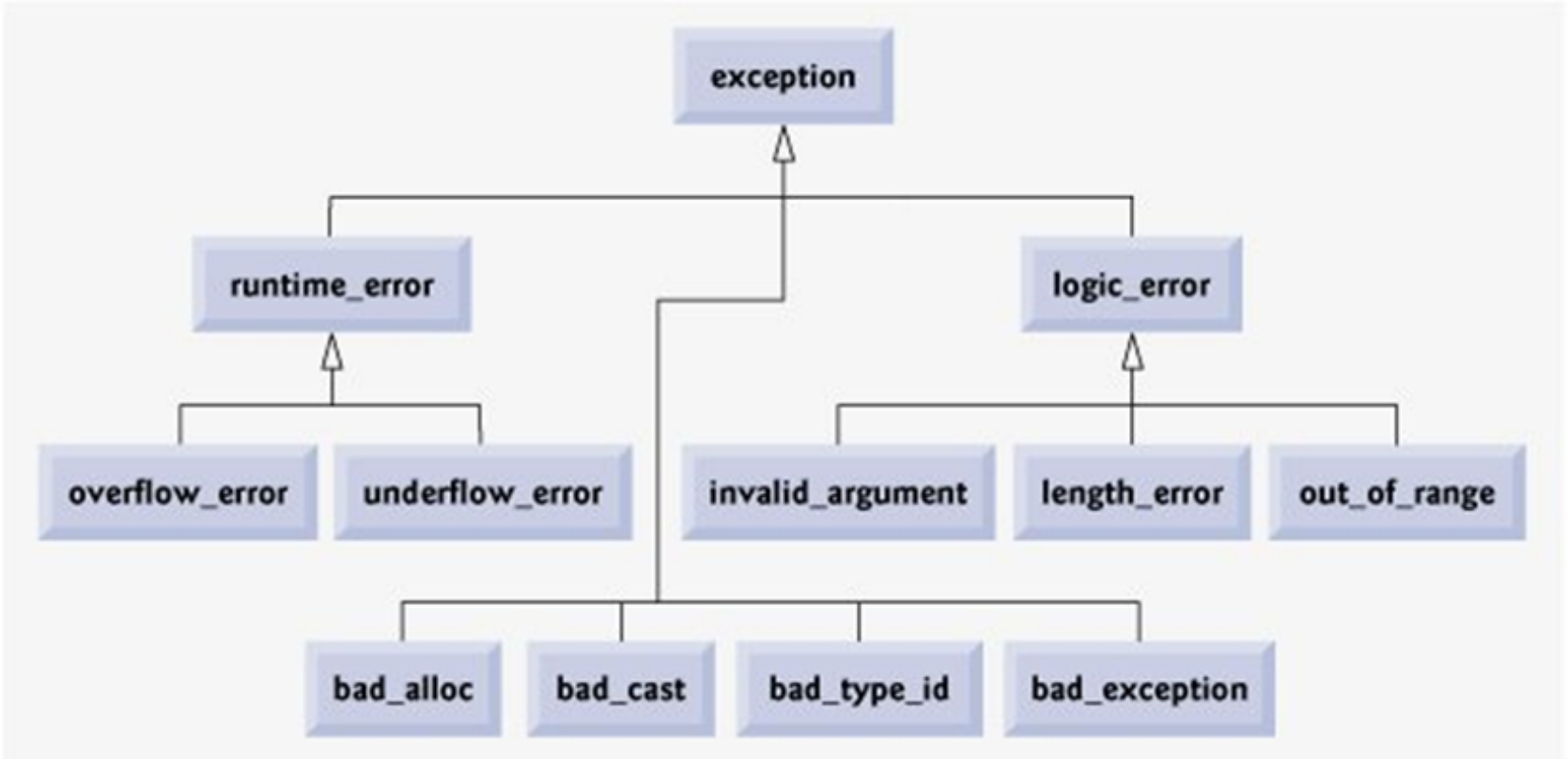
Guillermo Cámara-Chávez

Departamento de Computação - UFOP
Baseado nos slides do Prof. Marco Antônio Carvalho

Lembrando ...

- Uma **exceção** é uma indicação de um **problema que ocorre** durante a **execução de um programa**
- O próprio nome indica que o **problema é infrequente**;
- O **tratamento de exceções** permite que os **programas sejam mais robustos** e também tolerantes a falhas

Classes de Exceções da Biblioteca padrão



Classes de Exceções da Biblioteca padrão (cont.)

- Exceções
  - *bad_alloc:* falha de alocação
  - *bad_cast:* falha de conversão
  - *bad_typeid:* falha de verificação de tipo
  - *bad_exception:* falha de exceção

Classes de Exceções da Biblioteca padrão (cont.)

- Erros lógicos
  - *invalid_argument:* argumento inválido
  - *length_error:* dimensão errada
  - *out_of_range:* fora do intervalo

Classes de Exceções da Biblioteca padrão (cont.)

- Erros de *runtime*
  - *overflow_error:* número muito grande
  - *underflow_error:* número muito pequeno

Exemplo

Modificar o código de divisão por zero, usando uma excecão do sistema.

Exemplo (cont.)

```
#include <iostream>
#include <stdexcept>
class DivideByZeroException : public runtime_error{
public:
    DivideByZeroException(const string msg) :
            runtime_error(msg){}
};

double divisao(int num, int den){
    if (den == 0)
        throw DivideByZeroException("Divisao por zero!
            \n");
    return static_cast<double>(num) / den;
}
```

Exemplo (cont.)

```
int main(){
    int num, den;
    double result;
    cout << "Digite dois numeros inteiros (ctrl+z para
        finalizar)\n";
    while (cin >> num >> den){
        try{
            result = divisao(num, den);
            cout << "A resposta: " << divisao(num, den)
                << endl;
        }
        catch (DivideByZeroException& e){
            cout << "Error: " << e.what();
        }
        cout << "Digite dois numeros inteiros (ctrl+z
            para finalizar)\n";
        }
        return 0;
}
```

Observações

- Nossa classe que representa o erro de divisão por zero é derivada da classe *runtime_error*

Observações

- Nossa classe que representa o erro de divisão por zero é derivada da classe *runtime_error*
- Derivada da classe *exception* da biblioteca padrão C++

Observações

- Nossa classe que representa o erro de divisão por zero é derivada da classe *runtime_error*
- Derivada da classe *exception* da biblioteca padrão C++
- Representa os erros de execução.

Observações

- Nossa classe que representa o erro de divisão por zero é derivada da classe *runtime_error*
- Derivada da classe *exception* da biblioteca padrão C++
- Representa os erros de execução.
- Nossa classe simplesmente passa uma *string* de erro ao construtor da classe *runtime_error*

Observações

- Nossa classe que representa o erro de divisão por zero é derivada da classe *runtime_error*
- Derivada da classe *exception* da biblioteca padrão C++
- Representa os erros de execução.
- Nossa classe simplesmente passa uma *string* de erro ao construtor da classe *runtime_error*
- Não é obrigatório derivar classes específicas para erro, como fizemos.

Observações

- Se optarmos por **derivar** a classe *runtime_error*, teremos um **recurso extra**

Observações

- Se optarmos por **derivar** a classe *runtime_error*, teremos um **recurso extra**
- O método virtual **what**, nos permite obter uma mensagem de erro apropriada;

Observações

- Se optarmos por **derivar** a classe *runtime_error*, teremos um **recurso extra**
- O método virtual **what**, nos permite obter uma mensagem de erro apropriada;
- No nosso exemplo, usamos um objeto da classe *DivideByZeroException* para indicar uma tentativa de divisão por zero

Observações

- Se optarmos por **derivar** a classe *runtime_error*, teremos um **recurso extra**
- O método virtual **what**, nos permite obter uma mensagem de erro apropriada;
- No nosso exemplo, usamos um objeto da classe *DivideByZeroException* para indicar uma tentativa de divisão por zero
- Ao **criar o objeto**, o **construtor** será **chamado** e a mensagem enviada.

Erros comuns

- Não pode haver código entre um bloco *try* e um *catch*
- Cada *catch* só pode ter um único parâmetro
- É um erro de lógica capturar o mesmo tipo de exceção em dois *catch* diferentes;
- Após o tratamento da exceção, **achar que o fluxo de execução volta** para o ponto em que a exceção foi lançada.

Exemplo 2

Criar um vetor usando a classe *vector* e evitar o acesso a uma posição inválida

Exemplo 2 (cont.)

```
#include <iostream>
#include <stdexcept>
#include <vector>

int main(){
    vector<int> v(4);
    int i, valor;

    try{
        cin >> i >> valor;
        v.at(i) = valor;
    }
    catch (out_of_range &e){
        cout << "Error: " << e.what();
    }
    catch (...){
        cout << "Erro inesperado";
    }
        return 0;
}
```

Processando falhas *new*

Criar um vetor de ponteiros de dimensão 50 e alocar dinâmicamente para cada ponteiro um vetor de 50 milhões de posições

Processando falhas *new* (cont.)

```
int main(){
    double *ptr[50];

    for (int i = 0; i < 50; i++){
        ptr[i] = new (nothrow) double[50000000];
        if (ptr[i] == nullptr){
            cerr << "Memory allocation fail! \n";
            break;
        }
        else
            cout << "Allocated 50 millon doubles\n";
    }
```

Processando falhas *new* (cont.)

```
        cout << "\ndeleting...";
        for (int j = 0; j < i; j++){
            if (ptr[i] != nullptr)
                delete[] ptr[j];
        }
        cout << "\ndeleted";
        return 0;
}
```

Processando falhas *new* (cont.)

```
Allocated 50 millon doubles
Allocated 50 millon doubles
Allocated 50 millon doubles
Memory allocation fail!

deleting...
deleted
```

Processando falhas *new* (cont.)

```
#include<stdexcep>
int main(){
    double *ptr[50];

    try{
        for (int i = 0; i < 50; i++){
            ptr[i] = new double[50000000];
            cout << "Allocated 50000000 doubles\n";
        }
    }
    catch(bad_alloc& e){
        cerr << "Error: " << e.what();
    }
    for (int j = i-1; j >= 0; j--){
        if (ptr[i] != nullptr)
            delete[] ptr[i];
    }
    return 0;
}
```

Processando falhas *new* (cont.)

```
Allocated 50 millons doubles
Allocated 50 millons doubles
Allocated 50 millons doubles
Allocated 50 millons doubles
Error: std::bad_alloc
deleting...
deleted
```

Exemplo

Inserir vários pares de números e dividir os mesmos, lançar uma exceção se o divisor é zero ou os valores digitados são inválidos

Exemplo (cont.)

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

int main()
{
    int num, den;
    double res;
    cout << "\nDigite dois numeros: ";
    while (true){
        try{
            if (  !(cin >> num >> den) ){
                throw invalid_argument("\nInserir
                    somente inteiros");
            }
            if ( den == 0 ){
                throw runtime_error("\ndivisao por zero
                    ");
            }
```

Exemplo (cont.)

```
            res = static_cast<double>(num) / den;
            cout << "\nresposta: " << res;
        }
        catch(invalid_argument& e){
            cerr << "\nErro: " << e.what();
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max
                (), '\n');
        }
        catch(runtime_error& e){
            cerr << "\nErro: " << e.what();
        }
        cout << "\nDigite dois numeros (ctrl+c para
            finalizar): " ;
    }
    return 0;
}
```

Exemplo 2

Alterar o tamanho de um vetor definido usando a classe *vector*

```
#include <iostream>          // std::cerr
#include <stdexcept>         // std::length_error
#include <vector>            // std::vector
using namespace std;
int main (void) {
  try {
    // vector throws a length_error if resized above max_size
    vector<int> myvector;
    myvector.resize(myvector.max_size()+1);
  }
  catch (const length_error& le) {
          std::cerr << "Length error: " << le.what() <<
               '\n';
  }
  return 0;
}
```

FIM