UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

UFC CAMPUS QUIXADÁ

Programação Orientada a Objetos

Prof. Thiago Werlley

2021.2

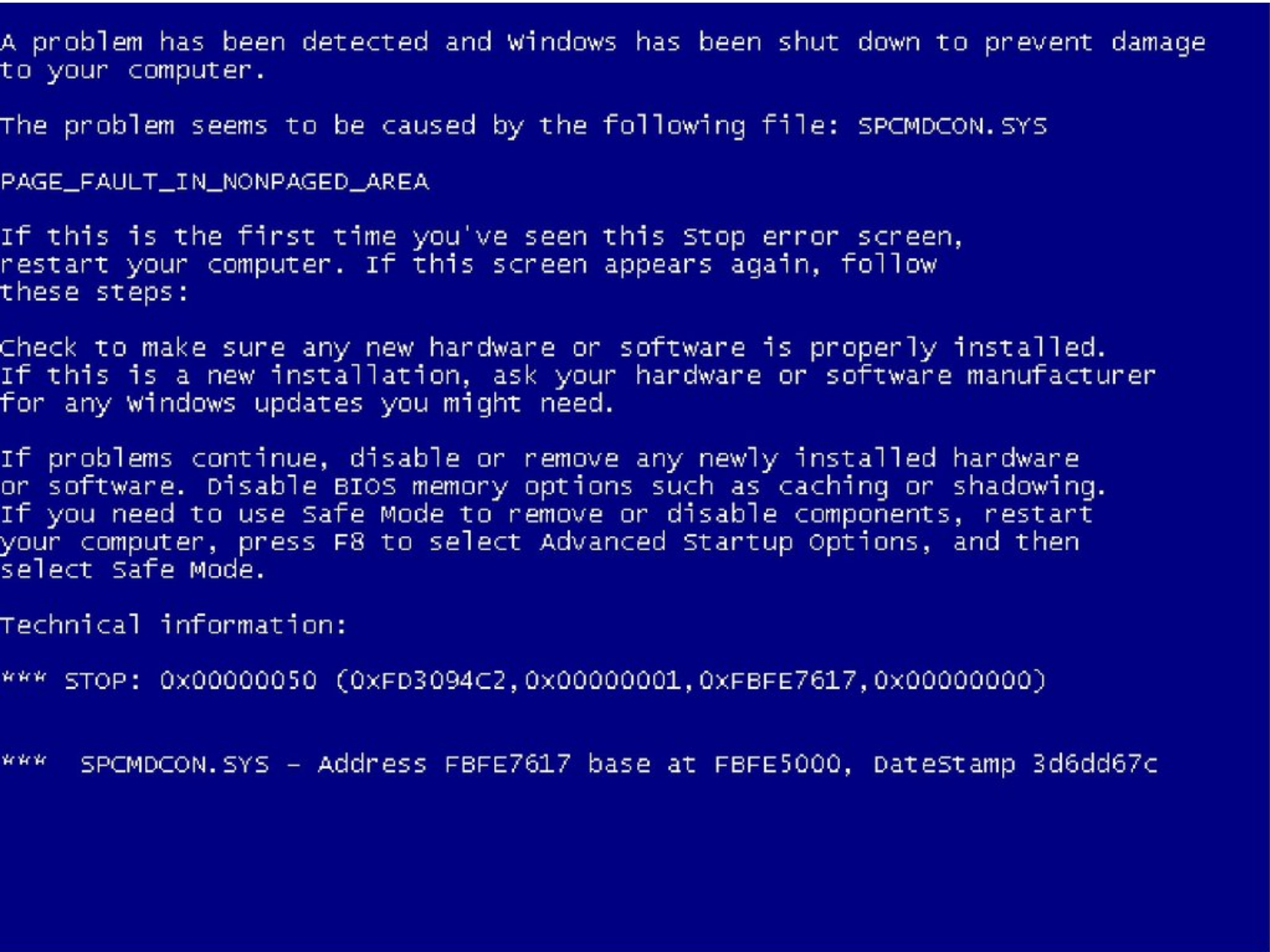
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS DE QUIXADÁ

Na aula passada

- Polimorfismo
  - Funções Virtuais
  - Resolução Dinâmica
  - Classes Abstratas
    - Funções Virtuais Puras
  - Conversão de Tipos
  - Destrutores Virtuais

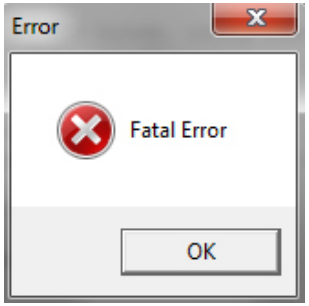
Na aula de hoje

- Exceções
  - *try*, *throw* e *catch*
  - Modelo de Terminação
  - Erros comuns
  - Quando Utilizar Exceções?
  - Classes de Exceções da Biblioteca Padrão


A problem has been detected and Windows has been shut down to prevent damage
to your computer.
The problem seems to be caused by the following file: SPCMDCON.SYS
PAGE_FAULT_IN_NONPAGED_AREA
If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:
Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.
If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.
Technical information:
*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)
*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

Exceções

- 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;
  - A **regra** é que o programa execute corretamente.
- O **tratamento de exceções** permite que os programas sejam mais robustos e também tolerantes a falhas
  - Os erros de execução são processados;
  - O programa trata o problema e continua executando como se nada tivesse acontecido;
  - Ou pelo menos, termina sua execução elegantemente.



Exceções

- Em programas com muitas classes criadas por diferentes programadores, se ocorrer um erro de execução, qual classe o tratará?
  - Como estabelecer a comunicação entre os métodos das diferentes classes?
- Se uma função precisa enviar uma mensagem de erro, ela "**lança**" um objeto representando o erro para fora dela;
- Se a função que a chamou não "**capturar**" este objeto, ele será enviado para quem a chamou
  - E assim por diante, até que alguma função o **capture**.

Exemplo 1

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

int main()
{
	int n=(int)90000000000, *ptr;

	ptr = new int [n];
	ptr[0] = 5;

	cout<<"Endereco: "<<(void*) ptr <<endl;

	delete [] ptr;

	return 0;
}
```

Exemplo 2

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

int main()
{
      int n=(int)90000000000, *ptr;

      ptr = new int [n];

      if(ptr)
      {
            ptr[0] = 5;
            cout<<"Endereco: "<<(void*) ptr <<endl;
            delete [] ptr;
      }
      else
            cout<<"Memoria insuficiente!" <<endl;

      return 0;
}
```

Exceções

- Nenhum destes dois programas evitam o erro causado quando a linha do operador ***new*** for executada
  - Compilação ok;
  - Não haverá tempo nem para executar o ***if*** que vem depois, no segundo código.
- O ideal seria "***tentar***" executar a instrução ***new***
  - Se der certo, ok;
  - Caso contrário, "***lançar***" um objeto com a exceção para que alguma função o "***capture***" e o processe.

try, throw e catch

try, throw e catch

- Estas ações podem ser realizadas com o uso de exceções;
- Temos 3 palavras chave:
  - *try* (*tentar*);
  - *throw* (*lançar*);
  - *catch* (*capturar*).

try

- A instrução ***try*** é utilizada para definir blocos de código em que exceções possam ocorrer
  - O bloco de código é precedido pela palavra ***try*** e é delimitado por { e };
  - As instruções que podem implicar em exceções e todas as instruções que não podem ser executadas em caso de exceção fazem parte do bloco de código.

throw

- A instrução ***throw*** lança uma exceção
  - Indica que houve um erro;
  - É criado um objeto, que contém a informação sobre o erro
    - Logo, podemos criar uma classe que defina o erro
    - Ou utilizar uma existente.
  - Posteriormente, outro trecho de código capturará este objeto e tomará a atitude adequada.

catch

- Exceções são tratadas por manipuladores *catch*
  - Capturam e processam as exceções.
- Pelo menos um *catch* deve seguir um bloco *try*
  - O bloco de código é precedido pela palavra ***catch*** seguido de parênteses e é delimitado por { e };
  - Dentro dos parênteses há os parâmetros da exceção
    - Representa o tipo de exceção que o *catch* pode processar.
  - O *catch* cujos parâmetros sejam de tipos iguais ao da exceção será executado;
  - O valor default para um catch é **...**
- Normalmente um *catch* imprime mensagens de erro, termina o programa elegantemente ou tenta refazer a operação que lançou a exceção.

try, *throw* e *catch*

- Um engano comum é achar que em todo tratamento de exceções é obrigatório usar pelo menos um *try*, um *throw* e um *catch* juntos
  - Recomendável é utilizar juntos o *try* e o *catch* para manipular as exceções;
  - O *throw* é utilizado somente para lançar exceções, não para manipulá-las.

Modelo de Terminação

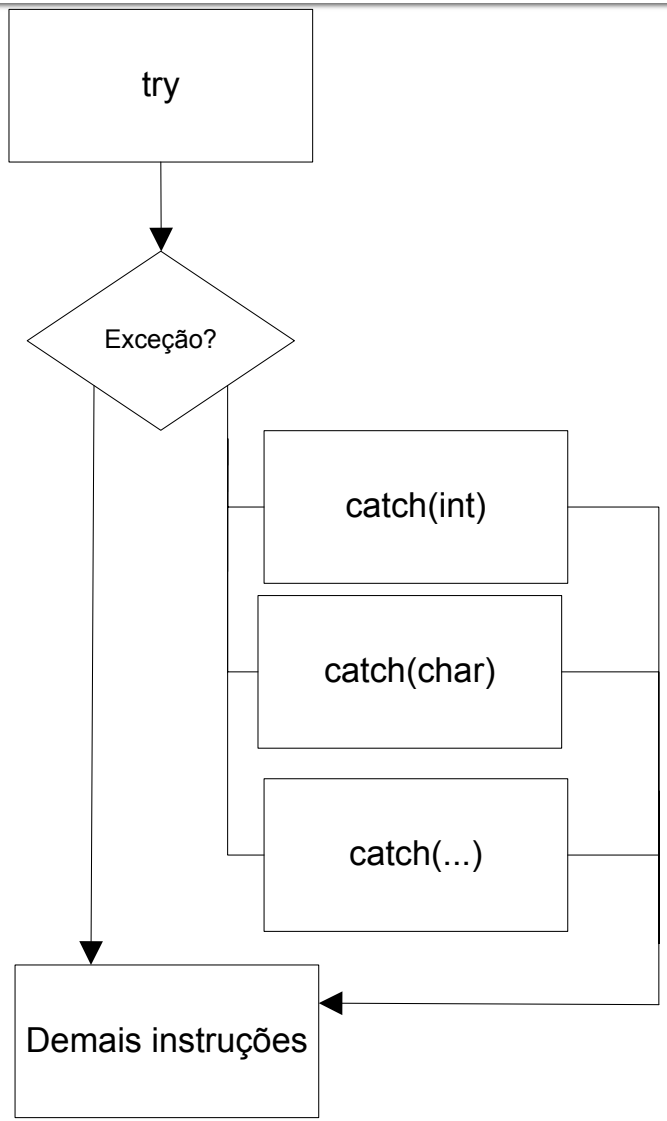
Modelo de Terminação

- Se em um bloco *try*, uma exceção ocorre:
  - O bloco termina imediatamente;
  - O primeiro *catch* cujo tipo seja igual (ou uma classe derivada) do tipo da exceção é executado;
  - Quando o *catch* termina, suas variáveis são destruídas e o fluxo de execução passa para a primeira instrução depois do último *catch* relacionado ao *try* que lançou a execução.
    - O que é chamado de **modelo de terminação**.
  - Há outro modelo em que o fluxo passa para o ponto logo após ao lançamento da exceção
    - **Modelo de resumo**.

Modelo de Terminação

- Se ocorrer uma exceção em um *try* e não houver *catch* correspondente, ou se não houver *try*
  - A função que contém a instrução com erro termina;
  - O programa tenta encontrar um bloco *try* aberto com a chamada da função;
  - Caso a exceção seja realmente inesperada, é executada uma chamada ***abort***
    - Não há nenhum bloco *try* que a contenha;
    - O programa termina sem chamar destrutores.
- Se não ocorrer uma exceção, o bloco *try* termina e todos os *catch* são ignorados.

Modelo de Terminação



Continua na próxima aula...

Criando, Lançando e Capturando Exceções

DivideByZeroException.h

```
#include <stdexcept> // arquivo de cabeçalho stdexcept contém runtime_error
using namespace std;

// objetos DivideByZeroException devem ser lançados por funções
// ao detectar exceções de divisão por zero
class DivideByZeroException : public runtime_error
{
public:
   // construtor especifica a mensagem de erro padrão
   DivideByZeroException(): runtime_error( "tentou dividir por zero" )
   {}
};
```

Criando, Lançando e Capturando Exceções

- Nossa classe que representa o erro de divisão por zero é derivada da classe ***runtime_error***
  - Definida na ***stdexcept***;
  - Derivada da classe ***exception*** da biblioteca padrão C++
    - Classe base de todas as exceções.
  - 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.

Criando, Lançando e Capturando Exceções

- Se optarmos por derivar a classe ***runtime_error***, teremos um recurso extra
  - O método virtual ***what***, que nos permite obter uma mensagem de erro apropriada;
  - No nosso exemplo, utilizaremos 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.

driverDivideByZeroException.cpp

```
#include <iostream>
using namespace std;
#include "DivideByZeroException.h" // Classe DivideByZeroException

// realiza a divisão e lança o objeto DivideByZeroException se
// a exceção de divisão por zero ocorrer
double quotient( int numerator, int denominator )
{
   // lança DivideByZeroException se tentar dividir por zero
   if ( denominator == 0 )
      throw DivideByZeroException(); // termina a função

   // retorna resultado da divisão
   return static_cast< double >( numerator ) / denominator;
}
```

driverDivideByZeroException.cpp

```
int main()
{
   int number1; // numerador especificado pelo usuário
   int number2; // denominador especificado pelo usuário
   double result; // resultado da divisão

   cout << "Enter two integers (end-of-file to end): ";

   // permite ao usuário inserir dois inteiros para dividir
   while ( cin >> number1 >> number2 )
   {
      // bloco try contém código que poderia lançar exceção
      // e código que não deve executar se uma exceção ocorrer
```

driverDivideByZeroException.cpp

```
      try
      {
         result = quotient( number1, number2 );
         cout << "The quotient is: " << result << endl;
      } // fim do try

      // handler de exceção trata uma exceção de divisão por zero
      catch ( DivideByZeroException &divideByZeroException )
      {
         cout << "Exception occurred: "
            << divideByZeroException.what() << endl;
      } // fim do catch

      cout << "\nEnter two integers (end-of-file to end): ";
   } // fim do while

   cout << endl;
   return 0; // termina normalmente
}
```

Erros Comuns

Erros Comuns

- Não pode haver código entre um bloco ***try*** e um ***catch***
  - É um erro de sintaxe.
- Cada ***catch*** só pode ter um único parâmetro
  - Uma lista de parâmetros é um erro de sintaxe.
- 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.

Quando Utilizar Exceções?

Quando Utilizar Exceções?

- Erros síncronos (na execução de uma instrução)
  - Índice de vetor fora dos limites;
  - *Overflow* aritmético (valor fora dos limites do tipo);
  - Divisão por zero;
  - Parâmetros inválidos;
  - Alocação de memória.
- Exceções não devem ser utilizadas para erros assíncronos (pararelos à execução do programa)
  - Erros de I/O;
  - Cliques de mouse e pressionamento de teclas.

Classes de Exceções da Biblioteca Padrão

Classes de Exceções da Biblioteca Padrão