



Programação e Sistemas de Informação

CURSO PROFISSIONAL TÉCNICO DE
GESTÃO E PROGRAMAÇÃO DE SISTEMAS INFORMÁTICOS

Programação Orientada a Objetos Avançada

MÓDULO 11

Professor: João Martiniano

Conteúdos abordados neste módulo

- Introdução às exceções
- O bloco `catch`
- A classe `Exception`
- Lançar exceções
- Criar exceções próprias

INTRODUÇÃO ÀS EXCEÇÕES

Introdução

- As exceções (ou *exception handling*) são um mecanismo através do qual um programa pode responder a eventos inesperados (habitualmente erros)
- Um programa que não utilize exceções corre o risco de *crashar* (ou seja, terminar abruptamente) caso ocorra um erro ou outro tipo de evento inesperado
- Ao invés, utilizando exceções, o programador pode precaver-se de modo a capturar e tratar os erros que ocorram, permitindo que a sua aplicação recupere ou permaneça em funcionamento
- É um mecanismo que não deve ser utilizado de forma excessiva
- Ou seja, devem-se identificar as zonas no código onde é maior a probabilidade de ocorrência de erros e aplicar apenas aí as exceções
- Um exemplo de código que potencialmente gera erros são as operações de I/O com ficheiros

Introdução

- É importante saber que as exceções gastam recursos (podendo tornar as aplicações mais pesadas e lentas) pelo que, sempre que possível, deve-se evitar o seu uso ou pelo menos tentar utilizá-las com alguma parcimónia
- Um erro comum consiste em, por exemplo, não efetuar verificações a dados ou não efetuar verificações que permitem antecipar os erros
- O programador que confie apenas nas exceções, denota alguma preguiça no seu estilo de programação
- Para mais informações:
<https://docs.microsoft.com/en-us/dotnet/standard/exceptions/best-practices-for-exceptions>

Como funcionam as exceções

- Resumidamente, as exceções funcionam da seguinte forma:
 - é feita uma tentativa para executar um bloco de código
 - se ocorrer um erro, é gerada uma exceção
 - se o programa tiver tratamento de exceções o programador pode incluir código para lidar com a exceção gerada

Como funcionam as exceções

- As exceções são implementadas utilizando as *keywords* `try...catch...finally`:

```
try
{
    ...
}
catch
{
    ...
}
finally
{
    ...
}
```

O código dentro do bloco `try` poderá dar origem uma exceção

A execução passa para um de vários blocos `catch`, os quais irão executar instruções quando ocorrem erros

Por fim, caso exista um bloco `finally`, este contém código para libertar recursos ou efetuar operações de finalização, após o tratamento do erro

- Caso exista o bloco `finally` este é sempre executado quer tenha sido levantada uma exceção ou não

Como funcionam as exceções

- Um bloco `try` necessita obrigatoriamente de:
 - um ou mais blocos `catch`:

```
try
{
    ...
}
catch
{
    ...
}
```

- ou um bloco `finally`:

```
try
{
    ...
}
finally
{
    ...
}
```

- ou, um ou mais blocos `catch` e um bloco `finally`

Como funcionam as exceções

- No seguinte exemplo é demonstrada a utilização do mecanismo de exceções
- O código dentro do bloco `try` irá tentar criar um ficheiro de texto e escrever conteúdo para o mesmo
- Caso ocorra um erro, o código dentro do bloco `catch` mostra uma mensagem ao utilizador

```
try
{
    File.WriteAllText(@"C:\texto1.txt", "Algum texto...");
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar criar o ficheiro  
\"texto1.txt\".");
}
```

O BLOCO CATCH

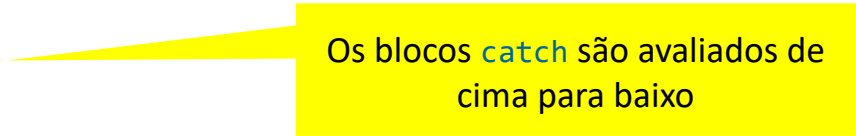
O bloco `catch`

- O bloco `catch` é responsável pelo tratamento das exceções
- É possível "apanhar" qualquer exceção e efetuar o respetivo tratamento, utilizando apenas um bloco `catch` genérico
- No entanto, esta não é a abordagem recomendada
- É também possível encadear vários blocos `catch`, sendo cada bloco responsável por tratar um tipo específico de exceção
- Os vários blocos são avaliados de cima para baixo
- É importante saber isto porque devem-se colocar as exceções da mais específica para a mais geral

O bloco `catch`

- Sintaxe:

```
try
{
    ...
}
[catch [(exceção)]
{
    ...
}]
[catch [(exceção)]
{
    ...
}]
...
```



Os blocos `catch` são avaliados de cima para baixo

O bloco `catch`

- No seguinte exemplo é efetuada a operação de divisão entre duas variáveis
- Caso a variável `y` contenha o valor zero, é lançada a exceção `DivideByZeroException`

```
try
{
    Console.WriteLine(x / y);
}
catch (System.DivideByZeroException)
{
    Console.WriteLine("Erro: tentou efetuar divisão por 0");
}
```

- Neste caso foi tratada apenas um tipo muito específico de exceção

O bloco `catch`: Tratar várias exceções

- É importante saber quais as exceções que um bloco de código pode originar
- Por exemplo, se analisarmos a documentação do método `File.WriteAllText()` veremos que este método pode levantar 8 exceções diferentes:

Exceptions

`ArgumentException`

`path` is a zero-length string, contains only white space, or contains one or more invalid characters as defined by `InvalidPathChars`.

`ArgumentNullException`

`path` is `null`.

`PathTooLongException`

The specified path, file name, or both exceed the system-defined maximum length.

`DirectoryNotFoundException`

The specified path is invalid (for example, it is on an unmapped drive).

`IOException`

An I/O error occurred while opening the file.

`UnauthorizedAccessException`

`path` specified a file that is read-only.

<https://docs.microsoft.com/en-us/dotnet/api/system.io.file.writealltext?view=net-5.0>

O bloco `catch`: Tratar várias exceções

- Assim, o exemplo analisado anteriormente pode ser modificado de modo a tratar de forma diferenciada 3 exceções específicas e no final qualquer outra:

```
try
{
    File.WriteAllText(@"C:\texto1.txt", "Algum texto...");
}
catch (PathTooLongException)
{
    Console.WriteLine("Erro: nome de pasta e/ou ficheiro muito longo");
}
catch (DirectoryNotFoundException)
{
    Console.WriteLine("Erro: localização de ficheiro inválida");
}
catch (IOException)
{
    Console.WriteLine("Erro: ocorreu um erro no dispositivo");
}
catch
{
    Console.WriteLine("Ocorreu um erro ao tentar criar o ficheiro");
}
```

Exceção mais geral colocada no fim

A CLASSE EXCEPTION

A classe `Exception`

- A classe `Exception` é a classe base de todas as exceções
- Contém propriedades que fornecem dados acerca da natureza e possíveis causas de uma exceção
- Utilizando estas propriedades é possível:
 - apresentar ao utilizador informações adicionais sobre a exceção
 - é também possível ao programador conseguir resolver os erros de execução do seu programa

A classe `Exception`

- Resumo breve das propriedades da classe `Exception`:

Propriedade	Descrição
<code>Data</code>	Dados relacionados com a exceção
<code>HelpLink</code>	Link para ficheiro com informação mais extensa sobre a exceção
<code>InnerException</code>	Informações adicionais
<code>Message</code>	Mensagem com dados detalhados sobre a exceção
<code>Source</code>	Nome da aplicação ou objeto que originou a exceção
<code>StackTrace</code>	Uma <i>stack trace</i> utilizada para determinar onde ocorreu a exceção

A classe `Exception`

- No seguinte exemplo são mostrados ao utilizador dados detalhados sobre uma exceção:

```
try
{
    File.WriteAllText(@"C:\texto1.txt", "Algum texto...");
}
catch (DirectoryNotFoundException e)
{
    Console.WriteLine("Erro: localização de ficheiro inválida");
    Console.WriteLine("\nDados detalhados da exceção: {0}", e.Message);
}
```

Importante: instância da classe
`DirectoryNotFoundException`

Detalhes da exceção

LANÇAR EXCEÇÕES

Lançar exceções

- O programador pode lançar exceções utilizando para isso a instrução `throw`
- Sintaxe:
`throw [e]`
- Em que:
`e` Instância de uma classe derivada da classe `System.Exception`

Lançar exceções

- O seguinte exemplo demonstra o lançamento de uma exceção
- Imaginemos uma classe que contém um método para efetuar a divisão entre dois números inteiros:

```
public static class Calcular
{
    public static double Divisao(int x, int y)
    {
        if (y == 0)
        {
            throw new System.DivideByZeroException();
        }

        return x / y;
    }
}
```

Lançar exceções

- Fora da classe, o código que chama o método trata a exceção utilizando as instruções `try...catch`:

```
try
{
    Calcular.Divisao(2, 0);
}
catch (DivideByZeroException)
{
    Console.WriteLine("Erro: divisão por zero");
}
```

| CRIAR EXCEÇÕES
PRÓPRIAS

Criar exceções próprias

- Apesar do grande número de exceções presentes na .NET Framework poderá surgir a necessidade de criar novas exceções
- Neste caso deve ser criada uma classe derivada da classe `Exception`
- O nome da nova exceção deverá terminar com a palavra `Exception`

Criar exceções próprias

- Suponhamos que queremos criar uma exceção que sinaliza que um código postal é inválido
- Para tal criamos a classe `CodigoPostalInvalidoException`, derivada da classe `Exception`:

```
public class CodigoPostalInvalidoException : Exception
{
    public string CodigoPostal { get; }

    public CodigoPostalInvalidoException()
    { }

    public CodigoPostalInvalidoException(string message) : base(message)
    { }

    public CodigoPostalInvalidoException(string message, Exception inner)
        : base(message, inner)
    { }

    public CodigoPostalInvalidoException(string message, string codigoPostal)
        : this(message)
    {
        CodigoPostal = codigoPostal;
    }
}
```

Criar exceções próprias

- Depois, ao validar um determinado código postal, é possível lançar a nova exceção caso seja detetado um código postal inválido:

```
string codigoPostal1 = "3000-330";  
int componente1 = int.Parse(codigoPostal1.Substring(0, 4));  
int componente2 = int.Parse(codigoPostal1.Substring(5, 3));  
  
if ((componente1 < 1000) || (componente1) >= 10000)  
{  
    throw new CodigoPostalInvalidoException("Código postal inválido",  
codigoPostal1);  
}
```