MICROSOFT C#

EXCEPTIONS e EXCEPTION HANDLING EM C#

IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

Célio Carvalho

cdcarvalho@ipca.pt

TRY{…} CATCH {…} FINALLY {…}

- As exceções permitem lidar com situações de erro ou imprevistas de código (e.g. erros) em *runtime*;

```
static void Main(string[] args)
{
    int a = 10, b = 0;

    System.Console.WriteLine($" 2 / 0 = { a / b }");

    System.Console.ReadKey();
}
```

```
Excepção não processada: System.DivideByZeroException: Tentativa de dividir por zero.
   em ConsoleApp1.Program.Main(String[] args) em D:\...\ConsoleApp1\ConsoleApp1\Progr
```

TRY{…} CATCH {…} FINALLY {…}

- Em C# é utilizado o `try {…}`, `catch {…}` e `finally {…}` para lidar com os erros ou situações inesperadas;

- O bloco `finally` é opcional. Se existir, é executado sempre, mesmo que um bloco `catch` tenha sido executado. O `finally` é habitualmente utilizado para libertar recursos (e.g. fechar ficheiros ou conexões de dados efetuadas no bloco `try`);

```
static void Main(string[] args)
{
    int a = 10, b = 0;

    try
    {
        System.Console.WriteLine($" 2 / 0 = { a / b }");
    }
    catch (Exception e)
    {
        Console.WriteLine("ERRO: {0}", e.Message);
    }
    finally
    {
        Console.WriteLine("Fim...");
    }

    System.Console.ReadKey();
}
```

```
ERRO: Tentativa de dividir por zero.
Fim...
```

TRY{…} CATCH {…} FINALLY {…}

- Podem existir vários blocos **catch**, sendo que apenas será executado apenas um.
- É uma técnica utilizada para "especializar" o tratamento de erros em função do tipo de exceção ocorrida. Se não for encontrado nenhum tipo indicado, é levantada uma **Exception** genérica.

```
static void Main(string[] args)
{
    int a = 10, b = 0;

    try
    {
        System.Console.WriteLine($" 2 / 0 = { a / b }");
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine("ERRO: hmmm não é possível dividir por zero.");
    }
    catch (Exception e)
    {
        Console.WriteLine("ERRO: {0}", e.Message);
    }
    finally
    {
        Console.WriteLine("Fim...");
    }

    System.Console.ReadKey();
}
```

```
ERRO: hmmm não é possível dividir por zero.
Fim...
```

TRY{…} CATCH {…} FINALLY {…}

- As `Exception` podem ser lançadas pelo *Common Language Runtime* (CLR), pelo .Net Framework, por código de terceiros e pelo nosso próprio código através da instrução `throw`;

- O seguinte exemplo apresenta uma utilização do `throw`;

```
static void Main(string[] args)
{
    int a = 10, b = 0;

    try
    {
        if (b == 0)
            throw new DivideByZeroException();

        System.Console.WriteLine($" 2 / 0 = { a / b }");
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine("ERRO: hmmm não é possível dividir por zero.");
    }
    catch (Exception e)
    {
        Console.WriteLine("ERRO: {0}", e.Message);
    }
    finally
    {
        Console.WriteLine("Fim...");
    }

    System.Console.ReadKey();
}
```

EXCEPTIONS

- Todas as `Exception` derivam de `System.Exception`;
- Deve existir pelo menos um bloco `catch {…}`;
- Uma exceção pode voltar a ser atirada depois de tratada (através do `throw`);

```
static void Main(string[] args)
{
    int a = 10, b = 0;

    try
    {
        if (b == 0)
            throw new DivideByZeroException();

        System.Console.WriteLine($" 2 / 0 = { a / b }");
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine("ERRO: hmmm não é possível dividir por zero.");
    }
    catch (Exception e)
    {
        Console.WriteLine("ERRO: {0}", e.Message);
        throw e;
    }
    finally
    {
        Console.WriteLine("Fim...");
    }

    System.Console.ReadKey();
}
```

EXCEPTIONS

- Podem ser criadas classes `Exception` especificas derivando da super-classe `System.Exception`;

```
public class OperacaoMatematicaInvalida :Exception
{
    // variables
    private string authenticatedUser;

    0 references
    public OperacaoMatematicaInvalida(string authenticatedUser) : base()
    {
        this.authenticatedUser = authenticatedUser;
    }

    0 references
    public OperacaoMatematicaInvalida(string authenticatedUser, string mensagem) : base(mensagem)
    {
        this.authenticatedUser = authenticatedUser;
    }

    0 references
    public string AuthenticatedUser
    {
        get { return this.authenticatedUser; }
    }

    0 references
    public void NotifyAdministrator(string administratorEmail)
    {
        // aqui codigo para enviar email com base.Message
    }

}
```

```
throw new OperacaoMatematicaInvalida("manel", "não podes dividir por zero!!!");
```

BIBLIOGRAFIA

- This.GetStarted(), IPCA, Luís Ferreira, João C. Silva, Patrícia Leite, Marta Martinho e Célio Carvalho.
- Beginning C# 7 Programming with Visual Studio 2017, Wrox, Benjamim Perkins, Jacob Vibe Hammer e Jon D. Reid.
- https://docs.microsoft.com/en-us/dotnet/csharp/
- https://dotnettutorials.net/course/csharp-dot-net-tutorials/