



Microsoft
.NET



Residência
em Software

Debugging e Exceções

Prof. Hélder Almeida



INSTITUIÇÃO EXECUTORA



UESC

COORDENADORA



APOIO

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO



O que é Debugging

Quando se fala em realizar “debugging” ou “depuração”, se quer dizer que a aplicação está sendo executada com um “debugger” anexado a ela.

Através do debugger é possível ver o que seu código está fazendo enquanto está sendo executado.

Por exemplo, dá para verificar (e alterar) valores de variáveis em tempo de execução e acompanhar a transformação pelo código.

Como “Debugar”

Precisamos definir um ponto de interrupção (breakpoint) no código.

Breakpoint é um ponto no código onde o VS Code vai suspender (pausar) a execução da aplicação, e permitir que você investigue o estado e comportamento da aplicação

É muito importante para realizar a investigação de bugs em sistemas.

Debug e Release

Dois modos para realizar o Build de uma aplicação.

No modo Debug não há otimização para execução de código, permitindo a inserção de pontos de interrupção e outros elementos que permitem executar o código passo-a-passo e inspecionar os valores de variáveis.

Já no modo Release o foco é a otimização do Build, normalmente indicado quando há intenção de inserir o sistema no ambiente de produção.



Residência
em Software

Praticando



O que são Exceções

Quando programamos, devemos tentar antecipar erros e deixar nossa aplicação protegida deles.

Em alguns casos não temos como garantir que um erro não ocorrerá.

Cada vez que ocorre um erro o framework .NET lança uma Exception, que é um objeto com dados do erro.

Alguns tipos de Exceções em C#

- `FormatException`
- `ArgumentNullException`
- `OverflowException`
- `IndexOutOfRangeException`
- `IOException`
- `NullReferenceException`

Mais informações: <https://learn.microsoft.com/en-us/dotnet/api/system.exception?view=net-7.0#choosing-standard-exceptions>

Como tratar Exceções

É possível tratar exceções através de um bloco

```
try{...}-catch(Exception e){...}-finally{...}
```

Através desse bloco é possível tratar tanto exceções de maneira genérica, quanto diferenciando o tratamento por cada tipo.

A parte finally é opcional no tratamento de exceções, e é utilizado quando sempre se deseja executar um certo código.

Criar e lançar exceções

As exceções são usadas para indicar que ocorreu um erro durante a execução do programa. Objetos de exceção que descrevem um erro são criados e, em seguida, lançados com a instrução ou expressão **throw**. Então, o runtime procura o manipulador de exceção mais compatível.

Segundo a Microsoft, deve-se lançar exceções quando:

- O método não pode concluir sua funcionalidade definida. Por exemplo, se um parâmetro para um método tem um valor inválido.
- É feita uma chamada inadequada a um objeto, com base no estado do objeto. Um exemplo pode ser a tentativa de gravar em um arquivo somente leitura.
- Quando um argumento para um método causa uma exceção. Nesse caso, a exceção original deve ser capturada e uma instância de `ArgumentException` deve ser criada.

Exemplo

C# EXCEPTION

System.Object



System.Exception



SystemException

ApplicationException

```
#region Exceptions Examples

try{
    // Code that may throw an exception
    int result = Divide(10, 0);
    Console.WriteLine($"Result: {result}");
}
catch (DivideByZeroException ex){
    // Handle the specific exception
    Console.WriteLine("Error: Cannot divide by zero");
    Console.WriteLine(ex.Message);
}
catch (Exception ex){
    // Handle any other exceptions
    Console.WriteLine("An error occurred");
    Console.WriteLine(ex.Message);
}
finally{
    // Code that will always execute, regardless of whether an exception occurred or not
    Console.WriteLine("Finally block executed");
}

int Divide(int a, int b){
    if (b == 0)
    {
        // Throw a custom exception
        throw new DivideByZeroException("Cannot divide by zero");
    }
    return a / b;
}

#endregion
```

Prática

1. Crie um programa que solicita ao usuário que insira um número. Utilize um bloco try-catch para lidar com a possível exceção gerada se o usuário inserir algo que não seja um número. No bloco catch, exiba uma mensagem amigável informando ao usuário que um número válido deve ser inserido.
2. Escreva uma função que recebe dois números como parâmetros e realiza uma divisão. No entanto, utilize um bloco try-catch para lidar com a exceção de divisão por zero. Se uma divisão por zero for detectada, exiba uma mensagem indicando que a divisão por zero não é permitida.
3. Modifique o exercício 2 para incluir um bloco finally. No bloco finally, exiba uma mensagem indicando que o bloco finally foi alcançado, independentemente de ocorrer uma exceção ou não.
4. Crie uma função que simule uma operação que requer um número positivo como entrada. Se a função receber um número negativo, lance uma exceção personalizada indicando que números negativos não são permitidos. Utilize um bloco try-catch para lidar com essa exceção e exiba uma mensagem apropriada.
5. Escreva um programa que lê um arquivo de texto a partir de um caminho fornecido pelo usuário. Utilize um bloco try-catch para lidar com exceções que possam ocorrer, como FileNotFoundException ou IOException. Exiba mensagens específicas para cada tipo de exceção capturada.