Fundamentos de Desenvolvimento com C#

Aula 03: Tratamento de Exceções

Professor: Rinaldo Ferreira Junior

E-mail: rinaldo.fjunior@prof.infnet.edu.br

- **Professor:** Rinaldo Ferreira Junior
- **Graduação:** Pós-graduado em Arquitetura de Softwares
- **Atuação:** .Net | C# | SQL | NoSQL | Engenheiro de Software
- **E-mail:** rinaldo.fjunior@prof.infnet.edu.br
- **Linkedin:** https://www.linkedin.com/in/rinaldo-ferreira-junior-787326a

Aula 03: Roteiro

- Funções e Subrotinas
- A Classe System.Exception
- Tratamento de Exceções

Aula 03: Funções e Subrotinas

- Funções são métodos que executam blocos de código e retornam um valor após a execução.
- Subrotinas são métodos que executam blocos de código, mas não retornam um valor após a sua execução.
- Tanto funções quanto subrotinas, podem receber dados de entrada (parâmetros) que serão usados no seu processamento.
- Um grande benefício da criação de funções e subrotinas, é a reutilização de código, uma vez que elas podem ser chamadas diversas vezes e em diversos pontos do programa.

Aula 03: Subrotinas

- Subrotinas não devolve nenhum valor, apenas executam o código desejado, diretamente. A instrução void define uma subrotina.

```
public void ProcessTime()
{
    DateTime current = DateTime.Now;
    Console.WriteLine($"Estamos em {current}");
}
```

- Não há armazenamento, apenas a execução

```
ProcessTime();
```

Aula 03: Funções

- Funções devem definir que tipo de dado irão retornar, ao serem chamadas.

```
public int AddNumbers(int number1, int number2)
{
    int result = number1 + number2;
    return result;
}
```

- O armazenamento deve ser compatível com o retorno

```
int result = AddNumbers(10, 5);
Console.WriteLine(result);
```

Aula 03: Funções de Múltiplos Retornos

- Value Tuples são estruturas que podem armazenar diversos valores, que podem estar relacionados ou não.
- Os valores armazenados ganham nomes, conforme o nome de cada parâmetro.
- Value Tuples apresentam uma maneira conveniente de retornar múltiplos valores, sem a necessidade de criar classes ou parâmetros out.
- Introduzidos no C# 7.0

Aula 03: Funções de Múltiplos Retornos

- Defina os diferentes retornos na assinatura da função

```
public (int Sum, int Product) CalculateSumAndProduct(int a, int b)
{
    int sum = a + b;
    int product = a * b;

    return (sum, product);
}
```

- Na chamada, desconstrua o resultado

```
var (sum, product) = CalculateSumAndProduct(5, 10);

Console.WriteLine($"Sum: {sum}");
Console.WriteLine($"Product: {product}");
```

```
var result = CalculateSumAndProduct(5, 10);
int sum = result.Sum;
int product = result.Product;

Console.WriteLine($"Sum: {sum}");
Console.WriteLine($"Product: {product}");
```

Aula 03: A Classe System.Exception

Instituto Infnet

- Exceções são quaisquer atividades que ocorram fora do esperado em um programa.
- O .Net possui uma vasta série de Exceptions, todas derivadas de System.Exception, que permitem capturar informações sobre essas ações inesperadas.
- Exceções podem ser geradas pelo CLR (Common Language Runtime), components de terceiros, ou pelo próprio programa.
- O .Net fornece uma estrutura para capturar e tratar exceções ocorridas durante a execução de um programa.
- O .net também permite disparar uma exceção propositalmente, assim como permite a criação de classes Exception personalizadas.

Aula 03: A Classe System.Exception

- Possui propriedades/métodos que carregam informações para tratamento

Propriedade/Método	Informações
Message (propriedade)	A mensagem gerada pela exceção
Source (propriedade)	O nome da aplicação ou objeto que gerou a exceção
StackTrace (propriedade)	O histórico da exceção
InnerException (propriedade)	A origem da exceção, em caso de exceções aninhadas
HelpLink (propriedade)	Link para texto de ajuda, quando houver
ToString (método)	O nome da exceção, a mensagem, o nome da exceção de origem e o histórico

Aula 03: Tratamento de Exceções

- Tratamento de erros desestruturado
  - Código de difícil leitura, manutenção e depuração
  - É fácil sobrepassar erros
- Tratamento de erros estruturado
  - É suportado por múltiplas linguagens
  - Permite a criação de blocos de código, protegidos
  - Permite filtros
  - Permite manipulação aninhada
  - Código mais fácil de ler, manter e depurar

Aula 03: Tratamento de Exceções

- Um bloco try é usado no C# para dividir código que pode ser afetado por alguma Exception.
- Blocos catch adicionais são usados para capturar e tratar exceções geradas no bloco try.
- Um bloco finally opcional, contém código que é executado ocorrendo ou não, alguma exceção no bloco try, como por exemplo, liberar recursos que estão alocados no bloco try.
- Um bloco try requer um ou mais blocos catch, ou um bloco finally, ou ambos.

- Trate as exceções partindo das mais específicas para as menos específicas.

```
try
{
    // Código protegido entra aqui
}
catch (DivideByZeroException Dex)
{
    // Código do tratamento da exceção entra aqui.
    // Procure capturar apenas exceções que você sabe como tratar.
}
catch (Exception ex)
{
    // Se for tratar diretamente a classe System.Exception, redispare-a no final do catch.
    throw;
}
```

- A instrução throw redispara a exceção corrente ou uma nova exceção

- Exemplo

```
static void Main(string[] args)
{
    int number1 = 3000;
    int number2 = 0;

    try
    {
        Console.WriteLine(number1 / number2);
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine("Divisão de {0} por zero.", number1);
    }

    Console.Read();
}
```

Aula 03: Filtrando Exceções

- Recurso introduzido no C# 6.0.
- Permite definir uma condição para o bloco catch.
- Se a condição for satisfeita, o bloco catch é executado. Caso contrário, os blocos seguintes são avaliados.
- O filtro pode conter qualquer expressão que retorne um bool.

```
static void Main(string[] args)
{
    try
    {
        ConnectDatabase(true);
    }
    catch (CustomException ex) when (ex.Code == 42)
    {
        Console.WriteLine("Ocorreu erro de conexão. Código 42.");
    }
    Console.Read();
}
```