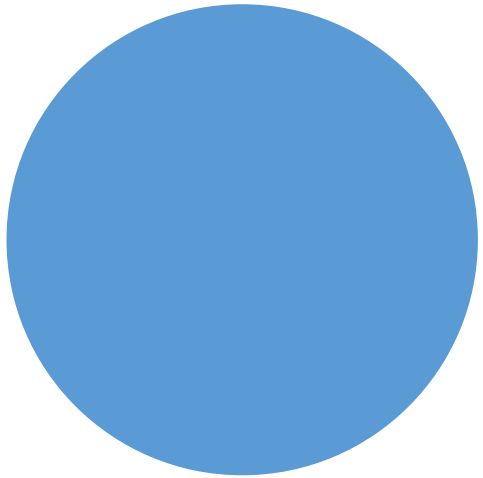
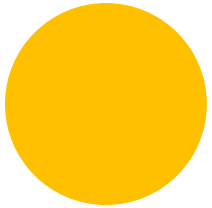

EXCEÇÕES

Prof. Hélio Esperidião

Exceção

Os recursos de manipulação de exceção da linguagem C# ajudam você a lidar com quaisquer situações excepcionais ou inesperadas que ocorram quando um programa for executado.

Uma **exceção** é um sinal que indica que algum tipo de condição excepcional ocorreu durante a execução do programa.

Exceções estão associadas a condições de erro que não tinham como ser verificadas durante a compilação do programa.

Exceções

As exceções em se referem aos erros que podem ser gerados durante a execução de um programa.

Como o próprio nome sugere, trata-se de algo que interrompe a execução normal do programa. É um problema que não ocorre frequentemente.

Exceptions

Exceptions são mecanismos primários para comunicar condições de erros.

Dessa forma não devemos abusar deste recurso mas saber usá-lo com bom senso.

Quando você precisar indicar que ocorreu um erro em uma determinada situação que não pode ser tratado no nível de código atual você pode lançar a exceção para ser tratada em outro nível usando a declaração **throw**.

Que métodos geram exceção?

- Conversões
- Acesso a arquivos não existentes
- Divisão por zero

Lançando uma Exception

Para indicar que ocorreu um erro em uma determinada situação você pode lançar a exceção usando a declaração **throw**.

A declaração **throw** é usada para sinalizar a ocorrência de uma situação anormal (*exceção*) durante a execução do programa.

A exceção gerada é um objeto cuja classe é derivada de **System.Exception**.

Exemplo

```
1 reference
private void Button1_Click(object sender, EventArgs e)
{
    Teste(null);
    Teste("");

}

2 references
private void Teste(string valor)
{
    if (valor == "")
    {
        throw new ArgumentNullException("valor", "o valor do parâmetro não pode ser vazio");
    }
    if (string.IsNullOrEmpty(valor))
    {
        throw new ArgumentNullException("valor", "o valor do parâmetro não pode ser null");
    }

}
```

try/catch/finally

Geralmente a instrução **throw** é usada com um bloco **try/catch/finally.**

Try significa tentar.

Se ele não conseguir executar na tentativa o catch trata o erro.

Evitando que a aplicação simplesmente feche.

```
private void Button1_Click(object sender, EventArgs e)
{
    try
    {
        Teste(null);
    }
    catch (Exception ex)
    {
        MessageBox.Show("O valor não pode ser vazio");
    }

}
1 reference
private void Teste(string valor)
{
    if (valor == "")
    {
        throw new ArgumentNullException("valor", "o valor do parâmetro não pode ser vazio");
    }
    if (string.IsNullOrEmpty(valor))
    {
        throw new ArgumentNullException("valor", "o valor do parâmetro não pode ser null");
    }

}
```

Garantindo a execução com Finally

Garante que os recursos usados sejam liberados, mesmo quando ocorrerem exceções.

Usado quando você usar objetos que encapsulam recursos externos (como conexões de banco de dados ou arquivos).

Você quer garantir que os recursos usados sejam liberados, mesmo quando ocorrerem exceções.

Use o bloco finally garante que o código definido no bloco sempre será executado mesmo ocorrendo uma exceção.

O bloco finally é útil para limpar todos os recursos que foram alocados em um bloco try. O controle sempre será passado para o bloco finally e o código definido no bloco sempre será executado.

Finally

- É opcional.
- Utilizado para desalocar recursos
- Sempre é executado

```
public string LerArquivo(String enderecoArquivo)
{
    String dado = "";
    FileStream fs = null;
    try
    {
        //Abre o arquivo texto
        fs = new FileStream(enderecoArquivo, FileMode.Open);
        StreamReader sr = new StreamReader(fs);
        string line;

        //Um valor é lido do arquivo e exibido no console
        line = sr.ReadLine();
        dado += line;
    }
    catch (FileNotFoundException e)
    {
        MessageBox.Show("Arquivo de dados não existe");
        MessageBox.Show(e.ToString());
    }
    finally
    {
        if (fs != null)
            fs.Close();
    }
    return dado;
}
```

Capturando Exceções

```
try
{
    FazAlgumaCoisa(null);
}
catch (ArgumentNullException ex)
{
    Console.WriteLine(“Exception: “ + ex.Message);
}
```

Múltiplas exceções

```
try
{
  //throw new ArgumentNullException();
}
catch (ArgumentException ex)
{
  //será alcançada aqui
}
catch (ArgumentNullException ex)
{
  //não será alcançada
}
```

Métodos implementados na classe Exception

Método/propriedade	Descrição
ToString()	Imprime o tipo de exceção, seguido da mensagem e **StackTrace**.
Message	Uma breve descrição do erro.
Source	A aplicação onde a exceção ocorreu.
StackTrace	A lista dos métodos na pilha atual. Útil para rastrear o caminho que causou a exceção.
TargetSite	O método que lançou a exceção
InnerException	A exceção que causou a exceção atual. Muitas vezes, as exceções são envolvidas no interior de outras exceções de nível superior.
HelpLink	Um link para um arquivo de ajuda, muitas vezes na forma de uma URL.
Data	Pares chave-valor de exceções especificas que fornecem mais informações.

Exemplo

```
1 reference
public void TesteDivisaoZero()
{
    try
    {
        int divisor = 0;
        float x = 5 / divisor;

    }
    catch (Exception ex)
    {
        MessageBox.Show("ToString()     : " + ex.ToString());
        MessageBox.Show("Message        : " + ex.Message);
        MessageBox.Show("Source         : " + ex.Source);
        MessageBox.Show("HelpLink       : " + ex.HelpLink);
        MessageBox.Show("TargetSite     : " + ex.TargetSite);
        MessageBox.Show("Inner Exception: " + ex.InnerException);
        MessageBox.Show("Stack Trace    : " + ex.StackTrace);
        MessageBox.Show("Data           : ");

    }
}
```