

Fundamentos de C#

Tratamento de Exceções



Tópicos Abordados



- Técnicas para tratamento de erros
- Exceções
 - System exceptions
 - Application exceptions
- Lançando e tratando exceções
 - Blocos *try*, *catch* e *finally*
- Detalhes da classe *Exception*
- Transformando exceções
- Exceções customizadas
- Dicas importantes

Falhas de Execução



- Uma aplicação pode apresentar erros durante a sua execução
 - Erros de programação (*bugs*)
 - Ações indevidas do usuário
 - Erros relacionados ao ambiente
 - Sistema de arquivos
 - Rede
 - etc.
- Como gerenciar essas falhas?
 - Booleanos
 - Magic numbers
 - Mecanismo estruturado de tratamento de exceções

Técnica #1: Booleanos



- Consiste em retornar um booleano indicando que houve um erro

```
bool sucesso = o.Processar();  
if (sucesso)  
{  
    //código em caso de sucesso  
}  
else  
{  
    //código em caso de falha  
}
```

E se o retorno não for capturado?

O que falhou?

Técnica #2: Magic Numbers



- Consiste em criar números que representam códigos de erro

```
int resultado = o.Processar();  
if (resultado == 100)  
{  
    //sucesso  
}  
else if (resultado == 110)  
{  
    //falha na validação  
}  
else if (resultado == 120)  
{  
    //falha de gravação no arquivo  
}
```

E se o retorno não for capturado?

Como entender sem uma tabela de códigos de erro?

Técnica #3: Tratamento de Exceções



- Permite saber exatamente qual erro aconteceu e onde aconteceu
- O fluxo normal do código e o tratamento do erro ficam separados
- É um mecanismo suportado por linguagens de programação mais modernas, como C#, Java, C++

Exceções



- Uma exceção é um objeto da classe **System.Exception** ou de uma de suas subclasses
- Dizemos que uma exceção é lançada (*throw*)
- Existem várias classes que representam exceções, mas podemos também criar nossas próprias

Exemplos de Exceções



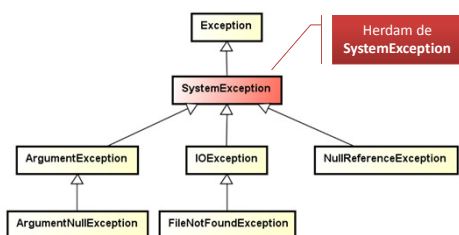
Classe da Exceção	Lançada quando...
ArgumentException	um parâmetro inválido é fornecido a um método
ArgumentNullException	um parâmetro com valor <i>null</i> é fornecido a um método
NullReferenceException	o código tenta acessar um elemento de um objeto cujo valor é <i>null</i>
DivideByZeroException	ocorre uma divisão por 0
IOException	ocorre um erro relacionado à I/O
FileNotFoundException	um arquivo não existe
...	

É uma boa prática que a classe tenha o nome terminado com **Exception**

System Exceptions



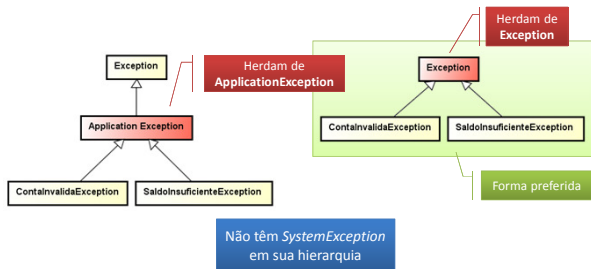
- São exceções genéricas, que podem existir em vários tipos de aplicações
- Lançadas pela CLR
 - Você pode lançá-las na sua aplicação também



Application Exceptions



- São exceções específicas de uma aplicação
- Lançadas por aplicações, e não pela CLR
- São as exceções que você vai criar



Lançando Exceções



- Quando determinado código detecta um problema, ele lança uma exceção
- Lançar a exceção significa avisar quem chamou o método que algo está errado

```
public void Sacar(double valor)
{
    if (valor <= 0)
    {
        ArgumentOutOfRangeException e = new ArgumentOutOfRangeException();
        throw e;
    }
    saldo = saldo - valor;
}
```

Cria a exceção

Lança a exceção

Quando a exceção é lançada (throw), o fluxo de execução volta para quem chamou o método

Tratando Exceções



- Quando uma exceção é lançada, o método que fez a chamada é que pode tratá-la
- O tratamento é feito através dos blocos *try* e *catch*

```
ContaBancaria c = new ContaBancaria();
try
{
    c.Sacar(-100);
}
catch (ArgumentOutOfRangeException e)
{
    Console.WriteLine("Erro ao fazer o saque");
}
// continuação...
```

Dentro do bloco *try* está a chamada que pode gerar uma exceção

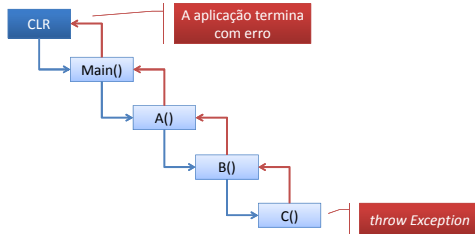
O bloco *catch* é executado apenas se a exceção ocorrer

Depois de executar os blocos *try* ou *catch*, o fluxo continua aqui

Exceções Não Tratadas



- Quando um código recebe o aviso da ocorrência de uma exceção, ele pode decidir não tratá-la
- Se o tratamento não ocorrer, a exceção é lançada para o método anterior na cadeia



O Bloco *finally*



- Executa sempre, independentemente de ter ocorrido uma exceção ou não
- É opcional

```
ContaBancaria c = new ContaBancaria();
try
{
    c.Sacar(-100);
}
catch (ArgumentOutOfRangeException e)
{
    Console.WriteLine("Erro ao fazer o saque");
}
finally
{
    c.FinalizarTransacao();
}
```

Executa sempre

O Bloco *finally*




- O bloco *finally* pode existir também sem o bloco *catch*

```
ContaBancaria c = new ContaBancaria();
try
{
    c.Sacar(-100);
}
finally
{
    c.FinalizarTransacao();
}
```

Se ocorrer uma exceção, o bloco *finally* é executado e a exceção é lançada

O Bloco *catch*



- O bloco *catch* é capaz de tratar a exceção que ele define ou subclasses dela

```
try
{
    AbrirArquivo("arq.txt");
}
catch (FileNotFoundException e)
{
    //...
}
```

Pode lançar *FileNotFoundException*

O *catch* captura a exceção quando ela ocorrer

```
try
{
    AbrirArquivo("arq.txt");
}
catch (IOException e)
{
    //...
}
```

Se *FileNotFoundException* for lançada, o *catch* vai capturar a exceção

IOException é a superclasse de *FileNotFoundException*

O Bloco *catch*



- Como fazer para capturar toda e qualquer exceção que ocorrer?

```
try
{
    //...
}
catch (Exception e)
{
    //...
}
```

Todas as exceções são do tipo *Exception* ou subclasses

O Bloco *catch*




- O bloco *catch* também pode ser definido sem o tipo da exceção

```
catch
{
    Console.WriteLine("Um erro ocorreu");
}
```

Captura exceções de todos os tipos, mas não permite saber o que aconteceu

6

Múltiplos Blocos *catch*


- Um bloco *try* pode ter vários blocos *catch* associados a ele
- Cada *catch* captura um tipo de exceção


```

try
{
    //...
}
catch (ArgumentNullException e)
{
    //...
}
catch (FileNotFoundException e)
{
    //...
}
catch (FileLoadException e)
{
    //...
}

```

O primeiro *catch* compatível com o tipo da exceção será executado

Os demais blocos *catch* não são executados

Múltiplos Blocos *catch*


- Cuidado com a ordem de declaração dos blocos *catch*

```


try
{
    c.Sacar(-100);
}
catch (ArgumentOutOfRangeException e)
{
    //...
}
catch (SystemException e)
{
    //...
}

```

Este código não compila

Forma correta

Este *catch* nunca será executado

Properties da Classe *Exception*


- A classe *Exception* possui diversas properties

	Property	Tipo	Significado
Read-only	Message	string	Mensagem do erro
	InnerException	Exception	Exceção aninhada
	Data	IDictionary	Dados customizados (em forma de pares de chave e valor)
	HelpLink	string	Link para algum local com mais informações sobre o erro
CLR	Source	string	Nome do assembly
	StackTrace	string	Detalhes sobre a pilha de chamadas de métodos
	TargetSite	MethodBase	Objeto que representa o método que lançou a exceção

Properties da Classe *Exception*



- Exemplos de uso das properties
 - Lançamento da exceção

```
Exception e = new Exception("Um erro ocorreu");  
e.HelpLink = "http://infodoerro.com";  
e.Data["valor"] = 10;  
throw e;
```

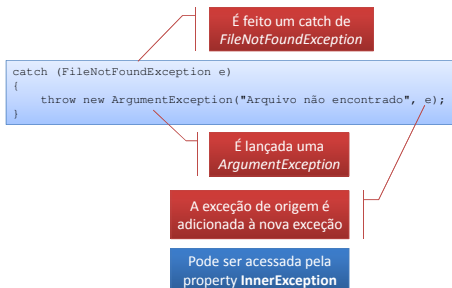
- Tratamento da exceção

```
catch (Exception e)  
{  
    Console.WriteLine(e.Message);  
    Console.WriteLine(e.HelpLink);  
    Console.WriteLine(e.Data["valor"]);  
    Console.WriteLine(e.Source);  
    Console.WriteLine(e.TargetSite);  
    Console.WriteLine(e.StackTrace);  
}
```

Transformando o Tipo da Exceção



- Em algumas situações pode ser interessante tratar uma exceção e lançar outra, de outro tipo



Criando Exceções Customizadas



- Você pode criar exceções para usar nas suas aplicações
 - Application exceptions
- As exceções devem herdar de **Exception**
 - Nunca de *SystemException*
- Exemplo de implementação

```
public class MyException : Exception  
{  
    public MyException() { }  
    public MyException(string message) : base(message) { }  
    public MyException(string message, Exception inner) : base(message, inner) { }  
    protected MyException(System.Runtime.Serialization.SerializationInfo info,  
        System.Runtime.Serialization.StreamingContext context) : base(info, context) { }  
}
```

Dicas Importantes sobre Exceções



- Crie uma hierarquia de classes de exceção para representar as exceções da sua aplicação
 - Muitas vezes estas classes terão apenas construtores
- Use as *system exceptions* já existentes quando elas forem aplicáveis
- Evite lançar exceções genéricas, como *Exception* ou *SystemException*
- Não use o tratamento de exceções como parte da lógica de execução da sua aplicação
- Um método deve lançar uma exceção, e não retorná-la