IFRN

PROGRAMAÇÃO ORIENTADA A OBJETOS EM C#

Exceções

Prof. Gilbert Azevedo

Objetivos

- Conceituar erros e exceções no contexto da POO
- Realizar o tratamento de exceções na escrita de programas
  - Instruções *try*, *catch* e *finally*
  - Exceções comuns do .NET
- Programar o lançamento de exceções
  - Instrução *throw*
- Validar o estado de objetos utilizando exceções

Erros

- Erros em sistema computacionais podem ocorrer por vários motivos
  - Comportamento imprevisível de usuários
  - Falhas de hardware, conexão ou comunicação
  - Falta de direito de acesso a recursos
  - Erros de memória e alocação
  - Estouro de capacidade no armazenamento de variáveis

Tratamento de Erros

- Abordagem Clássica
  - Utiliza variáveis globais para sinalizar erros
  - A lógica do programa fica junto com o tratamento do erro
  - Utiliza códigos de erro numéricos pouco explicativos
  - Pouca documentação
- Abordagem de POO
  - Utiliza objetos de erro chamados de Exceções
  - Permite separar a lógica do programa do tratamento do erro
  - Utiliza o conceito de classe para modelar os diferentes tipos de erros
  - Documentação mais detalhada facilita a manutenção do código

Tratando Exceções com Try-Catch

- O tratamento de erros é feito com as instruções Try-Catch
  - O bloco *try* (execução protegida): fluxo principal do programa
  - O bloco *catch* (captura da exceção): lógica de tratamento do erro

```
try {
// Código protegido – fluxo normal da aplicação
}
catch {
// Código do tratamento de erro – fluxo de exceção
}
```

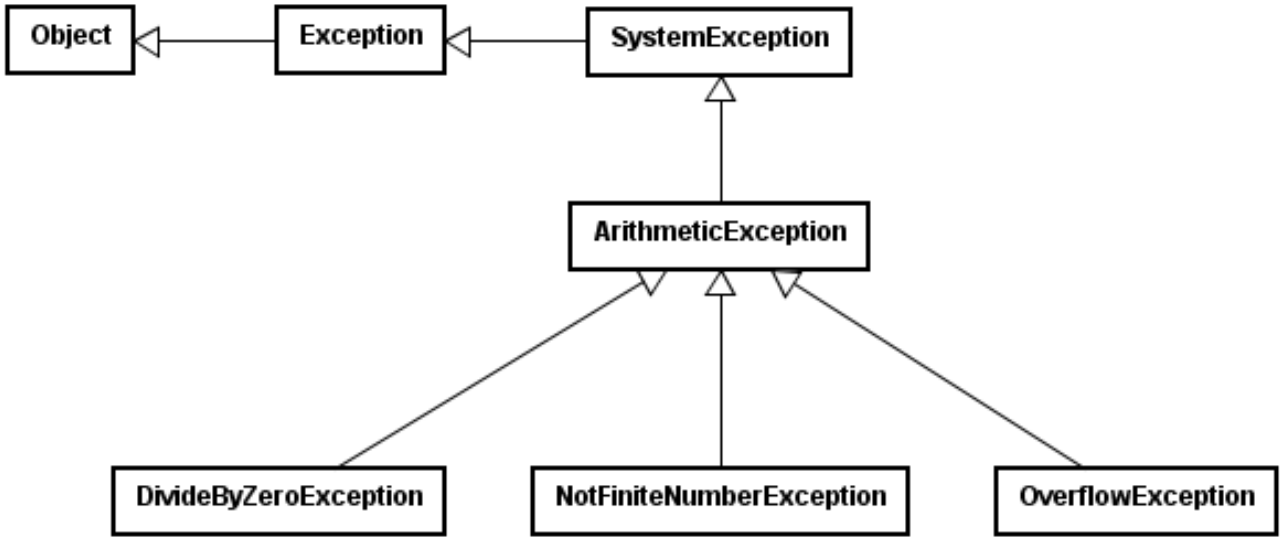
Exemplo

- Verificando se o valor informado é um inteiro válido
  - Uma exceção é lançada se o valor informado não puder ser convertido

```
int n;
try {
  n = int.Parse(Console.ReadLine());
}
catch {
  Console.WriteLine("Valor informado é inválido");
}
```

Classes de Exceções

- As exceções são organizadas em uma hierarquia de classes e são especializadas para notificar situações específicas de erro.
- Exemplo: Hierarquia para erros em operações aritméticas



Classes de Exceções

- A documentação do Framework informa as exceções que podem ser lançadas por um método e quando elas ocorrem
- Método Parse da struct int
  - ArgumentNullException – O argumento informado é nulo
  - FormatException – O valor informado não é um inteiro
  - OverflowException – O valor informado é menor ou maior que os limites do tipo
- https://docs.microsoft.com/en-us/dotnet/api/system.int32.parse?view=netcore-3.1

Capturando o Objeto da Exceção

- Quando uma exceção é lançada no programa, o objeto de erro pode ser capturado em um bloco Catch
- Todos os erros serão capturados com o uso da classe Exception

```
try {
  int a = 1, b = 0;
  Console.WriteLine(a / b);
}
catch (Exception obj) {
  Console.WriteLine(obj.Message);
}
```

Capturando Erros Específicos

- É possível capturar exceções de classes específicas, se necessário, para tratar diferentemente as situações de erro

```
try {
  int x = int.Parse(Console.ReadLine());
  int y = int.Parse(Console.ReadLine());
  int z = x/y;
}
catch (FormatException obj) {
  Console.WriteLine("Valor informado não é um número");
}
catch (DivideByZeroException obj) {
  Console.WriteLine("Divisão por zero");
}
```

Exceções Comuns do .NET

- Exception
- IndexOutOfRangeException
- NullReferenceException
- InvalidOperationException
- ArgumentNullException
- ArgumentOutOfRangeException

- https://docs.microsoft.com/pt-br/dotnet/standard/exceptions/

Tratamento de Exceções – Caso Geral

```
try {
  // Instruções a serem executadas
}
catch (TipoExceção erro) {
  // Tratamento de exceções de um tipo específico
}
catch (Exception erro) {
  // Tratamento genérico de exceções
}
finally {
  // Instruções sempre executadas, ocorram ou não erros
}
```

Bloco Try

- O Try é obrigatório
  - Bloco de execução protegida: implementa o fluxo principal do programa
  - O *runtime* tentará executar todas as instruções no bloco
- Se não houver erro
  - todas as instruções são executadas
- Se houver erro
  - Instruções restantes no try são ignoradas
  - A execução vai para os blocos catch ou finally.
  - Se nenhum catch capturar a exceção, a execução do aplicativo é encerrada

Bloco Catch

- O Catch é opcional
  - Bloco de manipulação da exceção: lógica os tratamentos de erro
- Um Try pode possuir vários blocos Catch
  - Os manipuladores devem ser escritos do mais específico para o mais genérico
- A exceção é consumida no bloco Catch
  - O objeto de erro é destruído
  - A exceção pode ser relançada utilizando a instrução throw

Bloco Finally

- O Finally é opcional
  - Instruções que devem sempre ser executadas, ocorram ou não erros
- Um Try possui apenas um bloco Finally
- Blocos Finally são utilizados para liberação de recursos
  - Manipuladores de arquivos
  - Conexões com bancos de dados

Instruções Executadas

- Execução sem erros
  - Try – todas as instruções
  - Finally – todas as instruções
- Execução com erros
  - Try – executado parcialmente
  - Catch – se a exceção for da classe informada no bloco
  - Finally – todas as instruções

Lançando Exceções

- Try – catch – finally: Testar uma condição de erro
  - Tratando exceções lançadas por métodos decorrentes de algum erro
  - Utilizando classes do Framework, de terceiros, entidades
- Throw: Informar uma condição de erro
  - Lançando exceções em consequência de uma condição de erro
  - Escrevendo suas próprias classes
- Para sinalizar a condição de erro
  - Instanciar um objeto de erro (classe descendente de Exception): new
  - Lançar o objeto de erro: throw

Testando um Argumento

- Sinalizando um erro de argumento inválido
  - new ArgumentOutOfRangeException: Instancia o objeto de erro
  - throw: lança o objeto de erro no aplicativo
  - *try-catch* deve ser usado quando o método for chamado

```
private string nomeMes(int mes) {
  switch (mes) {
    case 1:  return "Janeiro";
    ...
    case 12: return "Dezembro";
    default: throw new ArgumentOutOfRangeException("Mês inválido");
  }
}
```

Validando o Estado do Objeto

- Exceções devem ser usadas para validar o estado do objeto
  - Informar a condição de erro quando um valor inválido for recebido em construtores ou métodos de acesso (set)

```
class Triangulo {
  private double b, h;
  public void SetBase(double b) {
    if (b < 0) throw new ArgumentOutOfRangeException("Base inválida");
    else this.b = b;
  }
}
```

Validando o Estado do Objeto

- E os métodos devem ser testados quando forem chamados na interface com o usuário

```
Triangulo t = new Triangulo();
try {
  t.SetBase(double.Parse(txtBase.Text));
}
catch (ArgumentOutOfRangeException erro) {
  MessageBox.Show(erro.Message);
}
catch (Exception erro) {
  MessageBox.Show("Valor informado não é um número");
}
```

Referências

- Microsoft Visual C# 2010 – Passo a passo, John Sharp, Bookman, 2010
- Exceções e Manipulação de Exceções
  - https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/exceptions/index
- Práticas Recomendadas para Exceções
  - https://docs.microsoft.com/pt-br/dotnet/standard/exceptions/best-practices-for-exceptions