Curso C# Conceitos Básicos

Diogo Cezar Teixeira Batista

xgordo@gmail.com

Universidade Tecnológica Federal do Paraná
Campus Cornélio Procópio
UTFPR-CP

Cornélio Procópio - 2008

Agenda I

1. Programação Baseada em Objetos
   - Convenções e Padrões de Nomenclatura
   - Classes
   - Propriedades
   - Modificadores de visibilidade
   - Herança
   - Declaração e Chamada de Métodos e Objetos
   - Métodos e Atributos Static
   - Const e ReadOnly
   - Classes e Métodos Abstratos
   - Interfaces
   - Métodos Virtuais
   - Classes e Métodos Sealed - Finais
2. Tratamento De Erros e Exceções

Agenda II

- Comando Throw
- Bloco Try - Catch
- Bloco Try - Catch - Finally

Recomendação da Microsoft para nomeação de variáveis

- Evite usar underline "_";
- Não crie variáveis com o mesmo nome mudando somente entre maiúsculas e minúsculas;
- Utilize nomes de variáveis com minúsculas;
- Evite utilizar todas as letras maiúsculas;
- Notação *camelCasing*: Primeira letra de cada palavra em caixa alta, menos da primeira;
- Recomendações para nomeação de classes, métodos: Notação *PascalCasing*: primeiro caractere de cada palavra em caixa alta.

Recomendação da Microsoft para nomeação de variáveis

- Evite usar underline "_";
- Não crie variáveis com o mesmo nome mudando somente entre maiúsculas e minúsculas;
- Utilize nomes de variáveis com minúsculas;
- Evite utilizar todas as letras maiúsculas;
- Notação *camelCasing*: Primeira letra de cada palavra em caixa alta, menos da primeira;
- Recomendações para nomeação de classes, métodos: Notação *PascalCasing*: primeiro caractere de cada palavra em caixa alta.

Recomendação da Microsoft para nomeação de variáveis

- Evite usar underline "_";
- Não crie variáveis com o mesmo nome mudando somente entre maiúsculas e minúsculas;
- Utilize nomes de variáveis com minúsculas;
- Evite utilizar todas as letras maiúsculas;
- Notação *camelCasing*: Primeira letra de cada palavra em caixa alta, menos da primeira;
- Recomendações para nomeação de classes, métodos: Notação *PascalCasing*: primeiro caractere de cada palavra em caixa alta.

Recomendação da Microsoft para nomeação de variáveis

- Evite usar underline "_";
- Não crie variáveis com o mesmo nome mudando somente entre maiúsculas e minúsculas;
- Utilize nomes de variáveis com minúsculas;
- Evite utilizar todas as letras maiúsculas;
- Notação *camelCasing*: Primeira letra de cada palavra em caixa alta, menos da primeira;
- Recomendações para nomeação de classes, métodos: Notação *PascalCasing*: primeiro caractere de cada palavra em caixa alta.

Recomendação da Microsoft para nomeação de variáveis

- Evite usar underline "_";
- Não crie variáveis com o mesmo nome mudando somente entre maiúsculas e minúsculas;
- Utilize nomes de variáveis com minúsculas;
- Evite utilizar todas as letras maiúsculas;
- Notação *camelCasing*: Primeira letra de cada palavra em caixa alta, menos da primeira;
- Recomendações para nomeação de classes, métodos: Notação *PascalCasing*: primeiro caractere de cada palavra em caixa alta.

Recomendação da Microsoft para nomeação de variáveis

- Evite usar underline "_";
- Não crie variáveis com o mesmo nome mudando somente entre maiúsculas e minúsculas;
- Utilize nomes de variáveis com minúsculas;
- Evite utilizar todas as letras maiúsculas;
- Notação *camelCasing*: Primeira letra de cada palavra em caixa alta, menos da primeira;
- Recomendações para nomeação de classes, métodos: Notação *PascalCasing*: primeiro caractere de cada palavra em caixa alta.

Classes

Definição

Uma classe é um poderoso tipo de dado em C#. Como estrutura, uma classe define os dados e o comportamento dos tipos de dados.

Código 1: Exemplo de Classe em C#

```
1  class NomeDaClasse {
2          // Definição dos atributos
3          private int atrib1;
4          private string atrib2;
5          // Método construtor
6          public NomeDaClasse(int param1, string param2){
7          }
8          // Definição dos métodos
9          public tipoRetorno MetodoUm([lista de parâmetros]){
10         return [valor];
11         }
12 }
```

Propriedades

Definição

As propriedades são recursos fornecidos pelas classes para que seja possível alterar seus valores.

Código 2: Exemplo de Propriedades em C#

```
1  public tipodedado NomeDaPropriedade {
2          get{
3                  return nomeAtributo;
4          }
5          set{
6                  nomeAtributo = value;
7          }
8  }
9  // Utiliza-se da seguinte maneira
10 this.NomeDaPropriedade = valor;
11 valor = this.NomeDaPropriedade;
```

Modificadores de visibilidade

C# apresenta os seguintes modificadores de visibilidades:

- *private*;
- *public*;
- *protected*;
- *internal*;

Código 3: Exemplo de utilização dos modificadores de visibilidade C#

```
1  class NomeDaClasse {
2          private   int atrib1;
3          public    int atrib2;
4          protected int atrib3;
5          internal  int atrib4;
6          ...
7  }
```

Herança

Definição

A herança é um recurso utilizado para derivar classes que têm métodos ou atributos em comum. Sua principal vantagem é o reaproveitamento de código.

Código 4: Exemplo de declaração de herança em C#

```
1  class NomeDaClasse : ClasseBase {
2  ...
3  }
```

This e *Base*

Definição

As cláusulas *this* e *base* são referências que indicam a propria classe e a classe base, respectivamente.
Classe base: classe cuja a classe atual herda as propriedades e atributos.

Código 5: Exemplo de this e base em C#

```
1  this.nomeAtributo = valor;
2  valor = this.nomeAtributo;
3  this.NomeMetodo();
4
5  base.nomeAtributoClasseBase = valor;
6  valor = base.nomeAtributoClasseBase;
7  base.NomeMetodoClasseBase();
```

Declaração e Chamada de Métodos e Objetos

- Instanciar um objeto: operador *new*;
- Acessar seus atributos e métodos: instrução ".";

Código 6: Exemplo instanciação de objeto em C#

```
1  MinhaClasse obj = new MinhaClasse();
```

Código 7: Exemplo acesso a atributos e métodos em C#

```
1  obj.nomeMetodo();
2  obj.nomeAtributo = 23;
3  obj.NomePropriedade = "Apenas um teste";
```

Métodos e Atributos *Static* I

O que é *static*?

Static define um método ou atributo como pertencentes à classe em questão e não aos objetos.

Como se declara?

Sua declaração é feita com a palavra *static* depois do modificador de acesso (*public*, *private*) e antes do tipo de dado (*int*, *string*).

Como se acessa?

O seu acesso é feito pelo nome da classe e não mais pela referência da classe ou pelo nome do objeto.

Métodos e Atributos *Static* II

Código 8: Exemplo acesso a atributos e métodos estáticos em C#

```
1  NomeDaClasse.atributoEstatico = valor;
2  valor = NomeDaClasse.atributoEstatico;
3  NomeDaClasse.MetodoEstatico();
```

Const e ReadOnly I

Definição

São operadores utilizados para a criação de constantes, cujos os valores não poderão ser alterados durante a execução do programa.

Algumas diferenças entre os operadores:

- *const*:
  - Não pode ser estático (*static*);
  - O valor é setado em tempo de compilação;
  - É inicializado somente na compilação.
- *readonly*:
  - Pode ser estático (*static*);
  - O valor é setado em tempo de execução;
  - Pode ser inicializado na declaração ou na codificação do construtor.

Classes e Métodos Abstratos I

Definição

A classe abstrata é um tipo de classe que somente pode ser herdada e não instanciada.

Para que é utilizada?

É utilizada para definir as funcionalidades que serão implementas em suas subclasses.

Classes e Métodos Abstratos II

Código 9: Exemplo de implementação de uma classe abstrata em C#

```
1  abstract class formaClasse
2  {
3      abstract public int Area();
4  }
5  class quadrado : formaClasse
6  {
7      int x, y;
8      // Se não for implementado o método Area()
9      // será gerado um compile-time error.
10     // Utiliza-se o operador override para indicar a
           sobrescrita.
11
12     public override int Area(){ return x * y; }
13 }
```

Interfaces I

Definição

Uma interface define as operações que um objeto será obrigado a implementar.

- Nunca contém implementação;
- Não permite construtores;
- Deve-se criar uma classe ou estrutura e herdar da interface;
- Deve-se implementar todos os métodos da interface.

Interfaces II

Código 10: Exemplo de implementação de uma interface em C#

```
1  interface IExemploInterface {
2      void ExemploMetodo();
3  }
4  class ImplementacaoClasse : IExemploInterface {
5      void IExemploInterface.ExemploMetodo() { }
6      static void Main() {
7          IExemploInterface obj = new ImplementacaoClasse();
8          obj.ExemploMetodo();
9      }
10 }
```

Interfaces III

Note que, para se sobrescrever um método da interface utilizamos <*Interface*>.<*Metodo*>. Código 10, Linha 5. A declaração de uma instância de uma interface é feita de forma diferente da declaração de um objeto normal, aqui temos: Interface <var> = new <ClasseQueImplementaAInterface>(); Código 10, Linha 15.

Métodos Virtuais

- Usado para permitir que um método seja sobrescrito;
- Podem possuir corpo;
- Caso um método não seja declarado como *virtual* ou *abstract*, não poderá ser sobrescrito;

Classes e Métodos Sealed - Finais

Definição

Uma classe selada é utilizada para restringir características da herança do objeto.

- Os métodos declarados como *sealed* também não poderão ser sobrescritos;

Código 11: Exemplo de implementação de uma classe sealed em C#

```
1  sealed class ClasseSelada {
2      public int x, y;
3  }
4  class MainClass {
5      static void Main() {
6          ClasseSelada sc = new ClasseSelada();
7          sc.x = 110;
8          sc.y = 150;
9          Console.WriteLine("x = {0}, y = {1}", sc.x, sc.y);
10     }
11 }
```

Tratamento De Erros E Exceções

Definição

Ações que causam anômalias nas aplicações.

- Várias formas de tratamento;
- Plataforma .NET: Tratamento de excessões estruturadas;
- Objeto herdado de *System.Exception* deve ser criado para representar as excessões;
- O tratamento pode acontecer com alguma excessão pré-definida.

Excessões Pré-definas Mais Comuns

Excessão	Descrição (disparado quando)
System.OutOfMemoryException	alocação de memória, através de new, falha.
System.StackOverflowException	quando a pilha(stack) está cheia e sobrecarregada.
System.NullReferenceException	uma referência nula(null) é utilizada indevidamente.
System.TypeInitializationException	um construtor estático dispara uma excessão.
System.InvalidCastException	uma conversão explícita falha em tempo de execução.
System.ArrayTypeMismatchException	o armazenamento dentro de um array falha.
System.IndexOutOfRangeException	o índice do array é menor que zero ou fora do limite.
System.MulticastNotSupportedException	a combinação de dois delegates não nulo falham.
System.ArithmeticException	DivideByZeroException e OverflowException. Base aritmética.
System.DivideByZeroException	ocorre uma divisão por zero.
System.OverflowException	ocorre um overflow numa operação aritmética. Checked.

Comando Throw I

Definição

É utilizado para disparar ou sinalizar a ocorrência de uma situação inesperada durante a execução do programa, ou seja uma excessão.

- O parâmetro seguido deve ser da classe *System.Exception* ou derivada.

Comando Throw II

Código 12: Exemplo de utilização do comando throw

```
1  using System;
2  class Throws{
3    public static void Main(string[] args){
4      if(args.Length==1)
5        System.Console.WriteLine(args[0]);
6      else{
7        ArgumentOutOfRangeException ex;
8        ex = new ArgumentOutOfRangeException("Utilize uma
             string somente");
9        throw(ex); //Dispara a excessão
10     }
11   }
12 }
```

Bloco Try - Catch I

Definição

Uma ou mais instruções *catch* são colocadas logo abaixo do bloco *try* para interceptar uma excessão.

- Dentro do bloco catch é encontrado o tratamento da excessão;
- Encontrado na forma hierárquica (cada *catch* é verificado de acordo com a exceção);
- Um *catch* pode estar isolado, tratando qualquer exceção.

Bloco Try - Catch II

Código 13: Exemplo de utilização do bloco try - catch

```
1  using System;
2  class Catchs{
3    public static void Main(){
4      int iMax=0;
5      Console.Write("Entre um inteiro para valor máximo, entre
           0 e o máximo será sorteado:");
6      try{
7        iMax = Console.ReadLine().ToInt32();
8        Random r = new Random();
9        int iRand = r.Next(1,iMax);
10       Console.Write("O valor sorteado entre 1 e {1} é {0}",
             iRand, iMax);
11     }
12     catch(ArgumentException){
13         Console.WriteLine("O não é um valor válido");
14     }
```

Bloco Try - Catch III

```
15        catch(Exception e){
16          Console.WriteLine(e);
17        }
18      }
19    }
```

Bloco Try - Catch - Finally I

Finally

A instrução *finally* garante a execução de seu bloco, independente da excessão ocorrer no bloco *try*.

- *Finally* é utilizado para liberação de recursos consumidos;

Bloco Try - Catch - Finally II

Código 14: Exemplo de utilização do bloco Try - Catch - Finally

```
1  using System;
2  using System.Xml;
3  class TryCatchFinally{
4  public static void Main(){
5    XmlDocument doc = null;
6    try{
7      doc = new XmlDocument();
8      doc.LoadXml("<Exception>The Exception</Exception>"); //
           Carrega o conteúdo
9      throw new Exception(doc.InnerText); //Dispara a excessão
10   }
11   catch(OutOfMemoryException){
12     //Tratamento aqui
13   }
14   catch(NullReferenceException){
15     //Tratamento aqui
```

Bloco Try - Catch - Finally III

```
16      }
17      catch(Exception e){
18        //Tratamento aqui
19        Console.WriteLine("Excessão ocorrida no programa {0}", e
              );
20      }
21      finally{
22        Console.WriteLine(@"Gravando o Documento no C:\..."); //
              Uso do verbatim (@)
23        doc.Save(@"c:\exception.xml"); //Grava o conteúdo
24      }
25      Console.WriteLine("Esta linha não será executada...");
26  }
27  }
```

Informativos I

- Salas para realização da prova:
  - Tarde (15:50 as 17:30) Sala A135;
  - Noite (21:30 as 23:00) Sala A024;
- Cola autorizada: Papel A4, com o nome escrito em vermelho dos 2 lados. Preenchimento da folha à caneta;
- Prova será preenchida a caneta;
- Lista de exercícios deve ser entregue até amanhã antes da prova;