Curso de Extensão

Tecnologias Microsoft

UNICAMP

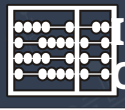
INF-0990

Programação em C#
Aula 4

Prof. Dr. Ricardo Ribeiro Gudwin

gudwin@unicamp.br

10 de Setembro de 2022



INSTITUTO DE COMPUTAÇÃO

Seleção de Classes da BCL

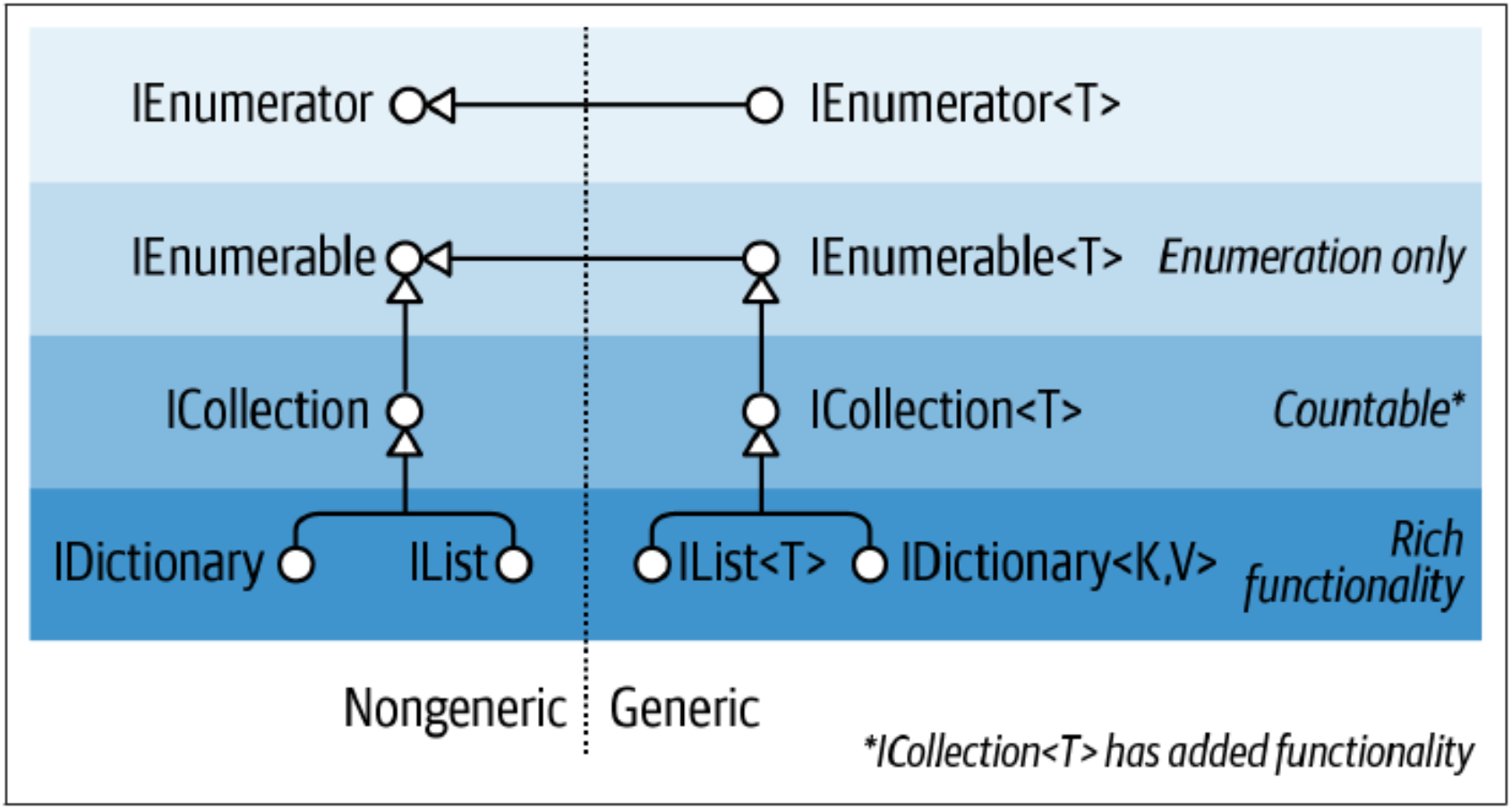
BCL: Base Class Library

- BCL: Base Class Library
  - Muitas das funcionalidades necessárias quando desenvolvemos um programa não estão na linguagem em si, mas nas funcionalidades providas pela BCL
  - Dentre as funcionalidades essenciais:
    - Manipulação de Strings
      - → Classes string, StringBuilder, CultureInfo
    - Manipulação de Datas
      - → Classes DateTime, DateTimeOffset, TimeSpan, DateOnly, and TimeOnly
    - Conversão string → numérica e numérica → string
    - Geração de Números Aleatórios
      - → Classe Random
    - GUIDs: Globally Unique Identifiers
      - → Classe Guid
    - Classes Utilitárias
      - → Classes Console, Environment, Process, AppContext

Coleções

- Criando Coleções
  - Além do uso de arrays, pela linguagem C#, existem diversas classes que podem ser úteis para o gerenciamento de diferentes tipos de coleções
    - Interfaces que definem protocolos padronizados para coleções de diferentes tipos
    - Classes que implementam diferentes tipos de coleções (listas, dicionários, etc.)
    - Classes-base para coleções que sejam específicas para uma aplicação

Coleções (Interfaces)



Coleções (Classes)

- Processando Listas
  - Classes ArrayList e List<T>
  - Classe LinkedList<T>
  - Classes Queue e Queue<T>
  - Classes Stack e Stack<T>
  - Outras
    - Classes BitArray, HashSet<T> e SortedSet<T>
- Dicionários
  - Dicionários são coleções do tipo chave/valor
  - Classes Dictionary<Tkey,Tvalue> e Hashtable
  - Classe OrderedDictionary
    - Um OrderedDictionary é um dicionário não-genérico que mantém os elementos na mesma ordem em que eles foram inseridos. O acesso tanto pode ser pelo índice como pela chave
  - Classes ListDictionary e HybridDictionary
    - Um ListDictionary usa uma lista-ligada para armazenar os dados do dicionário. Não provê a possibilidade de ordenamento, mas preserva a ordem original de entrada dos itens. Eficiente com listas pequenas (menos de 10 itens), mas lenta para um número muito grande de itens.
    - HybridDictionary começa usando uma lista-ligada, como um ListDictionary, mas muda para um hashtable quando o dicionário começa a crescer, resolvendo o problema de ineficiência de ListDictionary
  - Classes SortedDictionary<TKey,TValue> e SortedList<TKey,TValue>
    - Ambos são dicionários que permitem ordenação
    - SortedList é implementada internamente com pares ordenados, permitindo um bom desempenho para a busca, mas um baixo desempenho na inserção de novos elementos
    - SortedDictionary é bem mais rápida que SortedList ao inserir elementos em ordem aleatória. Entretanto, SortedList permite que os itens sejam acessíveis por índices (permitindo iteração), o que é bem mais complexo com um SortedDictionary

Coleções Especializadas

- Classes para Coleções
  - São muito convenientes para um uso mais comum, mas não permitem o controle do que acontece com os itens que são adicionados ou removidos da coleção
  - Em coleções com tipos (classes) mais sofisticadas, algumas vezes pode ser desejável esse controle
    - para disparar eventos quando um item é adicionado ou removido da coleção
    - para fazer o update de propriedades que dependem de itens adicionados ou removidos
    - Para detectar inserções ou remoções que sejam ilegais, gerando uma exceção
  - Classes Collection<T> e CollectionBase
    - Collection<T> é uma classe base para a construção de listas customizadas
    - Prevê métodos virtuais para inserção, remoção, limpeza e definição de índices, que devem ser implementados em suas subclasses
    - CollectionBase é uma versão não-genérica de Collection<T>
  - Classes KeyedCollection<TKey,TItem> e DictionaryBase
    - São o equivalente para dicionários
  - Classe ReadOnlyCollection<T>
    - É um wrapper para uma versão Read Only de uma coleção
    - Pode ser útil quando se deseja iterar sobre a coleção

Assemblies

- Assembly
  - É a unidade básica de deployment no .NET, funcionando como um container para todos os tipos (classes) gerados no código
  - Contém os tipos compilados com seu código em Intermediate Language (IL), recursos necessários para sua execução e informações para auxiliar no versionamento do código e referenciar outros assemblies dos quais o código depende
  - No .NET, um assembly é um único arquivo com a extensão .dll
  - Quando uma aplicação executável é gerada, além do arquivo com o assembly (.dll), é gerado um arquivo executável (.exe) apropriado à plataforma definida
  - Um assembly possui 4 diferentes tipos de coisas:
    - Um manifesto do assembly
      - → Provê informações para o CLR, como no nome, versão e outros assemblies que são referenciados
    - Um manifesto de aplicação, quando o assembly encapsular uma aplicação
      - → Provê informações ao sistema operacional, tais como o assembly pode ser executado, e eventuais privilégios administrativos necessários para sua execução
    - Código dos Tipos Compilados
      - → Código em IL, e metadados dos tipos definidos no assembly
    - Recursos
      - → Outros dados inseridos no assembly, tais como imagens, e textos em diferentes linguagens

Utilizando Assemblies Externos

- Usando o Nuget
  - Deve-se localizar o nome de referência do assembly externo no Nuget
  - Utiliza-se o .NET CLI para baixar o pacote e referenciar o assembly externo
    - `dotnet add package <NOME_ASSEMBLY> --version <VERSION_#>`
  - No código em C#, utiliza-se a diretiva `using` e o nome do assembly para usar suas classes no seu programa
- Usando Assemblies em Arquivos
  - Deve-se disponibilizar os arquivos .DLL dos assemblies em algum lugar conhecido (e.g.: diretório `lib`)
  - Acrescenta-se as seguintes diretivas no arquivo .csproj

```
<ItemGroup>
    <Reference Include="ExternalLibrary">
      <HintPath>..\lib\ExternalLibrary.dll</HintPath>
    </Reference>
</ItemGroup>
```

  - Acrescenta-se as diretivas `using` necessárias no código em C#

Diretivas de Pré-processamento

- Diretivas de Pré-processamento
  - Podem ser definidas

Preprocessor directive	Action
`#define symbol`	Defines *symbol*
`#undef symbol`	Undefines *symbol*
`#if symbol [operator symbol2]...`	*symbol* to test *operators* are ==, !=, &&, and ||, followed by `#else`, `#elif`, and `#endif`
`#else`	Executes code to subsequent `#endif`
`#elif symbol [operator symbol2]`	Combines `#else` branch and `#if` test
`#endif`	Ends conditional directives
`#warning text`	*text* of the warning to appear in compiler output
`#error text`	*text* of the error to appear in compiler output
`#error version`	Reports the compiler version and exits
`#pragma warning [disable | restore]`	Disables/restores compiler warning(s)
`#line [ number ["file"] | hidden]`	*number* specifies the line in source code (a column can also be specified from C# 10); *file* is the filename to appear in computer output; `hidden` instructs debuggers to skip over code from this point until the next `#line` directive
`#region name`	Marks the beginning of an outline
`#endregion`	Ends an outline region
`#nullable option`	See "Nullable reference types" on page 16

Documentação XML

- Comentários com Meta-informação XML
  - Podem ser inseridos no código, para auxiliar no processo de geração automática de documentação de código
  - Ferramentas como o **doxygen** podem ser utilizadas para gerar automaticamente páginas HTML, PDF e outros com documentação de apoio sobre o funcionamento das classes geradas no código, a partir desses comentários
  - O C# pré-define um conjunto de tags XML
    - `<summary>`, `<param>`, `<returns>`, etc.
    - Podem ser utilizados com meta-informação que é coletada pelas ferramentas de documentação, para gerar

Documentação XML

- Exemplos de meta-informação

```
/// <summary>Cancels a running query.</summary>
public void Cancel() { ... }

/// <summary>
/// Cancels a running query
/// </summary>
public void Cancel() { ... }

/**
<summary> Cancels a running query. </summary>
*/
public void Cancel() { ... }
```

- Existem plugins para o Visual Studio Code que auxiliam no processo de inserir essa meta-informação
  - E.g. **C# XML Documentation Comments**

Exemplo de Documentação Gerada

My Project

Main Page | Classes | Search

Public Member Functions | Public Attributes | List of all members

Octopus Class Reference

A classe **Octopus** é utilizada para representar um polvo More...

Public Member Functions

Octopus (string oname)

Construtor da classe **Octopus** More...

Public Attributes

int **Age** = 10

int **result**

Detailed Description

A classe **Octopus** é utilizada para representar um polvo

Constructor & Destructor Documentation

◆ Octopus()

Octopus.Octopus (string oname) inline

Construtor da classe **Octopus**

Parameters