UNIDADE 10 – NAMESPACE

Nessa unidade veremos a funcionadade e importantes do namespace.

Os namespaces são fortemente usados na programação em C# de duas formas. Primeiro, o .NET framework usa os namespaces para organizar suas muitas classes.

SUMÁRIO

1 NAMESPACE

1.1 ORGANIZAÇÃO

O código fonte de uma aplicação é definido em diversos arquivos. Conforme a quantidade de arquivos cresce surge a necessidade de algum tipo de organização para poder encontrar os arquivos rapidamente quando for necessário modificá-los.

A ideia para organizar logicamente os arquivos de uma aplicação é bem simples e as pessoas que utilizam computadores já devem estar familiarizadas. Os arquivos são separados em pastas ou diretórios.

1.2 O COMANDO NAMESPACE

Na terminologia do C#, as pastas nas quais são organizadas as classes e interfaces de uma aplicação são chamadas de **namespaces**. Devemos utilizar o comando namespace para separar as classes e interfaces de uma aplicação.

```
1  namespace Sistema.Contas
2  {
3      class Conta
4      {
5          // corpo da classe
6      }
7  }
```

Tabela 1: Conta.cs

É comum, para cada namespace, criar uma pasta com o mesmo nome do namespace e salvar todos os arquivos fonte que possuem classes ou interfaces desse namespace nessa pasta.

1.3 NAMESPACES ENCADEADOS

Assim como as pastas de um sistema operacional, os namespaces podem ser colocados dentro de outros namespaces.

```
1  namespace Sistema
2  {
3      namespace Contas
4      {
5          class Conta
6          {
7              // corpo da classe
8          }
9      }
10 }
```

Tabela 2: Conta.cs

Outra maneira de encadear namespaces é utilizar o símbolo “.”.

```
1  namespace Sistema.Contas
2  {
3      class Conta
4      {
5          // corpo da classe
6      }
7  }
```

Tabela 3: Conta.cs

1.4 NAMESPACE GLOBAL

Todas as classes, interfaces ou namespaces que não forem explicitamente colocadas em um namespace são automaticamente colocados no namespace global.

1.5 UNQUALIFIED NAME VS FULLY QUALIFIED NAME

Com a utilização de namespaces é apropriado definir o que é o nome simples (Unqualified Name) e que é o nome completo (fully qualified name) de uma classe ou interface.

O nome simples é o identificador declarado à direita do comando class ou interface. O nome completo é formado pela concatenação dos nomes dos namespaces com o nome simples através do caractere “.”.

Por exemplo, considere a seguinte código:

```
1  namespace Sistema.Contas
2  {
3      class Conta
4      {
5          // corpo da classe
6      }
7  }
```

Tabela 4: Conta.cs

O nome simples da classe acima é: Conta e o nome completo é: Sistema.Contas.Conta.

Duas classes de um mesmo namespace podem "conversar" entre si através do nome simples de cada uma delas. O mesmo vale para interfaces. Por exemplo, considere as seguintes classes:

```
1  //Arquivo:Sistema\Contas\Conta.cs
2  namespace Sistema.Contas
3  {
4      class Conta
5      {
6          // corpo da classe
7      }
8  }
```

Tabela 5: Conta.cs

```
1  // Arquivo: Sistema\Contas\ContaPoupanca.cs
2  namespace Sistema.Contas
3  {
4      class ContaPoupanca : Conta
5      {
6          // corpo da classe
7      }
8  }
```

Tabela 6: ContaPoupanca.cs

A classe ContaPoupanca declara que herda da classe Conta apenas utilizando o nome simples.

Por outro lado, duas classes de namespaces diferentes precisam utilizar o nome completo de cada uma delas para "conversar" entre si. O mesmo vale para interfaces. Como exemplo, considere as seguintes classes:

```
1  //Arquivo:Sistema\Contas\Conta.cs
2  namespace Sistema.Contas
3  {
4      class Conta
5      {
6          // corpo da classe
7      }
8  }
```

Tabela 7: Conta.cs

```
1  // Arquivo: Sistema\Clientes\Cliente .cs
2  namespace Sistema.Clientes
3  {
4      class Cliente
5      {
6          private Sistema.Contas.Conta conta;
7      }
8  }
```

Tabela 8: Cliente.cs

1.6 USING

Para facilitar a escrita do código fonte, podemos utilizar o comando using para não ter que repetir o nome completo de uma classe ou interface várias vezes dentro do mesmo arquivo.

```
1  // Arquivo: Sistema\Clientes\Cliente.cs
2  using Sistema.Contas;
3
4  namespace Sistema.Clientes
5  {
6      class Cliente
7      {
8          private Conta conta;
9      }
10 }
```

Tabela 9: Cliente.cs

Podemos utilizar vários namespaces. As declarações de using aparecem antes da declaração de qualquer namespace.

Atenção

Nunca crie um projeto ou pasta com o mesmo nome de uma classe. Pois o namespace pode se perder mandando mensagens confusas para o compilador.

1.7 NÍVEIS DE VISIBILIDADE

No C#, há cinco níveis de visibilidade: privado, interno, protegido, protegido interno e público. Podemos definir os níveis privado, protegido, público e interno com os modificadores private, protected, public e internal respectivamente.

Privado

O nível privado é aplicado com o modificador private.

O que pode ser privado? Atributos, propriedades, construtores, métodos, classes aninhadas ou iterfaces aninhadas.

Os itens em nível de visibilidade privado só podem ser acessados por código escrito na mesma classe na qual eles foram declarados.

Interno

O nível interno é aplicado com o modificador internal.

O que pode ser interno? Atributos, propriedades, construtores, métodos, classes ou interfaces.

Os itens em nível de visibilidade interno só podem ser acessados por código escrito em classes do mesmo assembly (.exe ou .dll) da classe na qual eles foram declarados.

Protegido

O nível protegido é aplicado como modificador protected.

Os itens em nível de visibilidade protegido só podem ser acessados pela própria classe ou por classes derivadas.

O que pode ser protegido? Atributos, propriedades, construtores, métodos, classes aninhadas ou interfaces aninhadas.

Público

O nível público é aplicado quando omodificador public é utilizado.

Os itens em nível de visibilidade público podem ser acessados de qualquer lugar do código da aplicação.

O que pode ser público? Atributos, propriedades, construtores, métodos, classes ou interfaces.

Protegido Interno

O nível protegido interno é aplicado associando o modificador protected com o modificador internal, resultando em protected internal.

Os itens em nível de visibilidade protegido interno só podem ser acessados por código do mesmo assembly (.exe ou .dll) ou no código das classes derivadas.

O que pode ser protegido interno? Atributos, construtores, métodos, classes aninhadas ou interfaces aninhadas.

1.8 Exercícios de Fixação

1) Crie um projeto no chamado **Unidade 10**.

2) Faça uma classe para modelar as contas dentro de um namespace chamado **Unidade_10.Contas**.

```
1  //Arquivo: Organizacao\Sistema\Contas\Conta.cs
2  namespace Organizacao.Sistema.Contas
3  {
4      class Conta
5      {
6          public double Saldo { get; set; }
7
8          public void Deposita(double valor)
9          {
10             this.Saldo += valor;
11         }
12     }
13 }
14
```

Tabela 10: Conta.cs

3) Teste a classe Unidade_10.Contas.Conta.cs.

```
1  // Arquivo: Organizacao\ Sistema\Aplicacao\TestaConta.cs
2  using Organizacao.Sistema.Contas;
3
4  namespace Organizacao.Sistema.Aplicacao
4  {
5      class TestaConta
6      {
7          private static void Main()
8          {
9              Conta c = new Conta();
10             c.Deposita(100);
11             System.Console.WriteLine(c.Saldo);
12         }
13     }
14 }
```

Tabela 10: Conta.cs

Bons Estudos!