Programação Orientada à Objetos

Aula 01
Introdução & Variáveis

Henrique Poyatos
henrique.poyatos@bandtec.com.br

Agenda

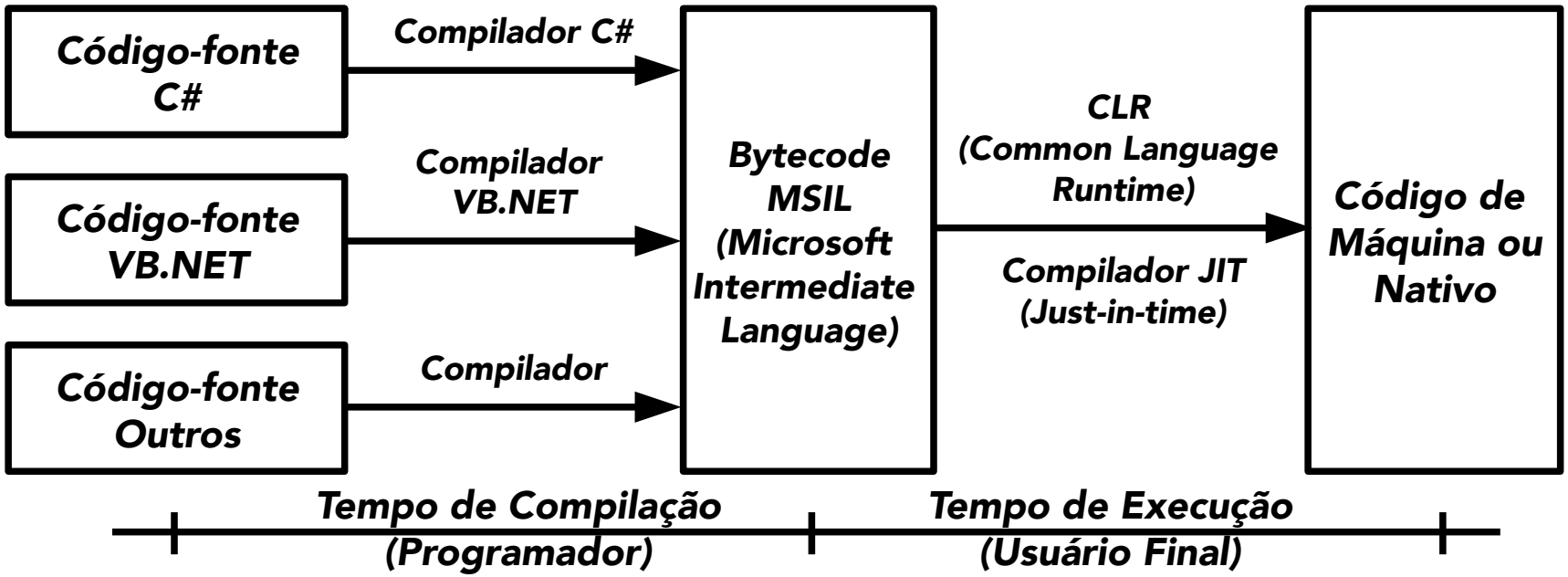
1. Termos
2. Compilado & Interpretado
3. .NET Framework
4. Criando um projeto no Visual Studio .NET
5. Considerações sobre a Linguagem C#
6. Entendendo o Código C#
7. Tipos de Variáveis
8. Declarações e Atribuições
9. Conversões

Termos

- **.NET:** Plataforma única de desenvolvimento Microsoft
- **Visual Studio:** É a IDE (Integrated Development Environment, Ambiente de Desenvolvimento Integrado) mais conhecida da Microsoft. É um "editor" de códigos, com uma série de recursos que iremos explorar. Em nosso treinamento utilizaremos o Visual Studio 2008. Outros: Visual Studio 2003, Visual Studio 2005 e agora o Visual Studio 2010.
- **Framework:** Conjunto de bibliotecas de Classes do qual a plataforma .NET se utiliza para a compilação, e do qual nós utilizaremos por conta de uma variedade de rotinas lá existentes.
- **C#, VB.NET, ASP.NET, python.NET, C++:** Linguagens de programação suportadas pela plataforma .NET;

Compilado & Interpretado

- Criado à imagem e semelhança da Plataforma Java, a Plataforma .NET é compilada & interpretada, ou também pode ser conhecida como duplamente compilada.
- Em tempo de compilação o programador compila gerando um bytecode. Este bytecode é distribuído e o usuário ao executar, utilizar um compilador JIT (presente no .NET Framework). É por esta razão que aplicativos feitos na plataforma .NET precisam ter o .NET Framework instalado na mesma máquina de execução.



.NET Framework (ou Microsoft .NET)

- O Framework, como citado anteriormente, é uma imensa Biblioteca de Classes, que se faz necessária para compilação e execução de nossos programas.
- Adicionalmente nos disponibiliza um conjunto variado de rotinas, desde as mais simples como Conversões, Formatações de Texto, Tratamento de Datas e Valores, Envio e Recebimento de Emails, Conexões de Banco de Dados, até rotinas mais sofisticadas e requintadas como Utilização de Threads, Entradas e Saídas de Arquivos (I/O), Renderizações Gráficas, Rotinas de Segurança, Serialização de Objetos e muito mais.
- O Framework que utilizaremos (4.5) traz uma série de novas possibilidades à linguagem C#.

Visual Studio .NET
Como criar um projeto

- Para criação de nossos projetos iniciais, estaremos utilizando um tipo de projeto do Visual Studio 2012 Express, que nos possibilitará a programação do C#, sem nos preocuparmos como elementos gráficos e visuais.
- O tipo de projeto que trabalharemos é um projeto C# de *Console Application*.

- Para criar um primeiro projeto:
  - Abra o Visual Studio 2012 Express
  - Selecione o Menu File -> New -> Project
  - Ao lado esquerdo, selecione a opção Visual C#
  - Ao lado direito, selecione a opção Console Application.
  - Indique o nome do projeto como “AloMundo”

Considerações sobre a Linguagem C#

- A codificação C# exige que todo a lógica da programação esteja contida em uma definição de tipo, sendo este tipo uma classe, uma estrutura, uma interface ou um enumerador.
- Em C# não é possível criar funções globais ou variáveis globais, como em muitas linguagens de programação.
- C# é uma linguagem de programação *case-sensitive*, ou seja, **Cliente** é diferente de **cliente**, e ***WriteLine*** é diferente de ***Writeline***.
- Palavras-chave em C# são sempre escritas em letras minúsculas, como por exemplo, class, private, public, interface, if, while, etc...
- Tipos e Namespaces são sempre escritas com a primeira em maiúscula, as demais em minúsculas, e palavras significativas começando em maiúscula, como por exemplo, System, Console, SqlConnection, etc...

Arquivo Program.cs

O arquivo Program.cs adicionado automaticamente ao projeto, tem a seguinte estrutura:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AloMundo
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Program.cs - using

Existe um bloco inicial que indica o uso de algumas bibliotecas, como System, System.Linq e System.Text.
O comando using é equivalente ao "import" em Java e em outras linguagens.

O fato delas estarem relacionadas não indica necessariamente que estejam sendo utilizadas, porém o Visual Studio já as adiciona naturalmente, entendendo que provavelmente você precisará delas.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AloMundo
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Program.cs - namespace

A namespace é um container de classes, ou seja, é um depósito lógico por onde poderemos encontrar nossas classes no futuro. A namespace AloMundo foi criada a partir do nome do projeto.

Num projeto, podem existir dezenas de namespaces, de forma a separar logicamente classes de código correlatos.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AloMundo
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Program.cs - class

Como dito anteriormente, todo o código C# estará representado dentro de um tipo. Seja uma classe, interface, enumerador ou estrutura.

Tipicamente nossos códigos serão colocados dentro de classes, e o Visual Studio já adiciona automaticamente a classe Program, para que possamos iniciar a nossa codificação.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AloMundo
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Program.cs – função/método Main

Para a nossa class Program foi adicionado automaticamente um método (rotina) Main. Este método é um ponto de entrada para a execução de nosso programa.

Precisamos deste método para iniciarmos a execução de nosso programa.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AloMundo
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Program.cs – Comando de Saída

O comando/função/método Console.Write() ou Console.WriteLine() será utilizado para exibir informações no console. Trata-se de um comando de saída.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace AloMundo
{
    class Program
    {
        static void Main(string[] args)
        {
                Console.WriteLine("Alô Mundo !!!");
                // O comando abaixo vai ser um macete
                // para "segurar" o console
                Console.ReadKey();
        }
    }
}
```

Tipos de Variáveis Inteiras e Reais

Tipo de Sistema	Abreviação	Descrição	Variação
System.Byte	byte	Números inteiros de 8bits sem sinal	0 a 255
System.Int16	short	Números inteiros de 16bits com sinal	-32.768 a 32.767
System.Int32	int	Números inteiros de 32bits com sinal	-2.147.483.648 a 2.147.483.647
System.Int64	long	Números inteiros de 64bits com sinal	- 9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
System.Sbyte	sbyte	Números inteiros de 8bits com sinal	-128 a 127
System.UInt16	ushort	Números inteiros de 16bits sem sinal	0 a 65.535
Tipo de Sistema	Abreviação	Descrição	Variação
System.Single	float	Números de ponto flutuante de 32bits	+/- 1,5 x 10E-45 a +/- 3,4 x 10E38
System.Double	double	Números de ponto flutuante de 64bits	+/- 5,0 x 10E-324 a +/- 1,7 x 10E308
System.Decimal	decimal	Números de ponto flutuante de 96bits	+/- 1,0 x 10E-28 a +/- 7,9 x 10E28

Tipos de Variáveis - Lógico, Caracter e Data/Hora

Tipo de Sistema	Abreviação	Descrição	Variação
System.Boolean	bool	Tipo booleano (Verdadeiro ou Falso)	true ou false

Tipo de Sistema	Abreviação	Descrição	Variação
System.Char	char	Um único caracter unicode de 16bits	
System.String	string	Cadeia (conjunto) de caracteres unicode	

Tipo de Sistema	Abreviação	Descrição	Variação
System.DateTime	DateTime	Tipo data e hora	01/01/0001 00:00:00 a 31/12/9999 23:59:59

Variáveis
Declaração

Para a declaração normalmente utilizaremos os tipos abreviados.

Sintaxe:

```
<tipo_dados> <nome_variável>;
```

Exemplos:

```
static void Main(string[] args)
{
    int numeroInteiro;
    string nomeCliente;
}
```

Regras para a declaração de variáveis:

- O nome deve iniciar com letra ou "_";
- Não são permitidos espaços no nome;
- Não são permitidos caracteres de pontuação;
- Podem ser utilizados números;
- Não podem ser utilizadas palavras reservadas como nome;
- O nome deve ser único no contexto;
- Evitar nomes como a, b, x, z que não são significativos;

Variáveis
Atribuição

Para a atribuição de valores nas variáveis iremos utilizar a seguinte sintaxe.

Sintaxe:

```
<nome_variável> = valor;
```

Exemplos:

```
static void Main(string[] args)
{
    string nomeCliente;
    nomeCliente = "Pedro Ferreira";
    Console.WriteLine("Valor Variável: " + nomeCliente);
}
```

Declaração e Inicialização:

```
static void Main(string[] args)
{
    string nomeCliente = "Pedro Ferreira";
}
```

Variáveis
Conversão Implícita

Em alguns casos será obrigatório a conversão de tipos para atribuição de valores entre variáveis.

Exemplos:

```
static void Main(string[] args)
 {
     short numeroShort = 10;
     int numeroInt = 90;

     numeroInt = numeroShort;
 }
```

Esta conversão (implícita) ocorrerá sem problemas uma vez que um valor do tipo short "cabe" dentro de uma variável do tipo int.

```
static void Main(string[] args)
{
    short numeroShort = 10;
    int numeroInt = 90;

    numeroShort = numeroInt;
}
```

Esta conversão implica em um erro de compilação uma vez que um valor do tipo int "não cabe" dentro de uma variável do tipo short. Mesmo que o valor 90 caiba num short, o C# irá criticar a atribuição.

Variáveis
Conversão Explícita

Para efetuar conversões entre tipos similares.

Sintaxe:

```
variável = (tipo)valor;
```

Tais conversões somente são possíveis entre tipos da mesma "família". Por exemplo, de short para int, de int para long, de double para decimal, de int para double, mas não de string para int

Exemplos:

```
static void Main(string[] args)
{
    short numeroShort = 10;
    int numeroInt = 90;

    numeroShort = (short)numeroInt;

    Console.WriteLine("Número Short: " + numeroShort);
}
```

Variáveis – Conversão Número para String

Para efetuar conversões de valores numéricos para string .

Sintaxe:

```
variável = valor.ToString();
```

Exemplos:

```
static void Main(string[] args)
{
    short numeroShort = 10;
    string numeroTexto;

    numeroTexto = numeroShort.ToString();

    Console.WriteLine("Número em Texto: " + numeroTexto);
}
```

Variáveis – Conversão Data para String

Para efetuar conversões de valores de data para string .

Sintaxe:

```
variável = valor.ToString(formato);
```

Exemplos:

```
static void Main(string[] args)
{
    DateTime data = DateTime.Now;
    string dataFormatada;
    string dataHoraFormatada;

    dataFormatada = data.ToString("dd/MM/yyyy");
    dataHoraFormatada = data.ToString("dd/MM/yyyy HH:mm");

    Console.WriteLine("Data não formatada: " + data);
    Console.WriteLine("Data formatada: " + dataFormatada);
    Console.WriteLine("Data/Hora formatada: " + dataHoraFormatada);

   Console.ReadKey();
}
```

Variáveis – Conversão String para Números

Para efetuar conversões de valores string para valores numéricos.

Sintaxe:

```
variável = Convert.FORMATO(texto);
```

Exemplos:

```
static void Main(string[] args)
{
    string valorTexto = "20";
    Console.WriteLine("Valor: " + Convert.ToInt32(valorTexto));
    Console.ReadKey();
}
```

As opções mais comuns da classe Convert são.

Convert.ToInt16
Convert.ToInt32
Convert.ToInt64
Convert.ToSingle
Convert.ToDecimal
Convert.ToDouble
Convert.ToDateTime