

**Primeiro programa em C#**

**Anatomia de um programa em C#**

**O ambiente de desenvolvimento**

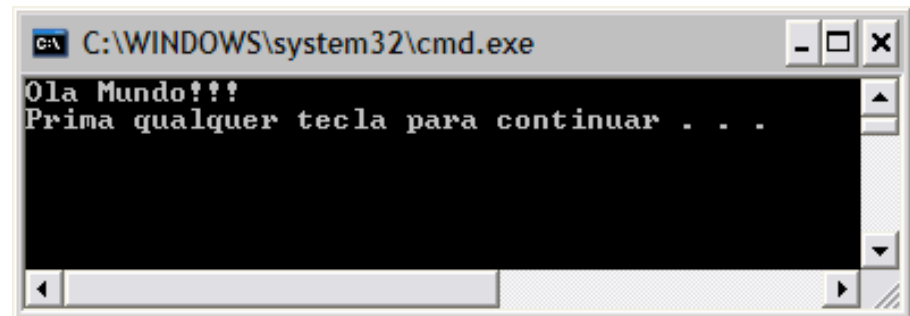
## Versão do clássico programa “helloworld” em C#

- Usa os principais elementos de um programa em C#: declaração de uma classe, uso do espaço de nomes, função Main, escrita pela consola

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

//Primeiro programa em C#

namespace Exercicios_LP_INF
{
    class PrimeiroPrograma
    {
        static void Main(string[] args)
        {
            /* escrever uma mensagem pela consola*/
            Console.WriteLine("Ola Mundo!!!");
        }
    }
}
```



## `using system`

- Indica que o espaço de nomes (**System**) vai ser usado pelo programa, o que implica que todas as definições contidas neste espaço (módulo) vão ser importadas, ou antes, podem ser usadas no programa
- O espaço de nomes System define classes de uso comum (e.g. Console)
- O ambiente C# ao gerar o template de um programa inclui automaticamente alguns espaços de nomes pré-definidos
- Os espaços de nomes podem ser pré-definidos pelo sistema (ambiente) ou definidos pelo programador.
- De momento, este conceito não nos interessa, voltaremos a ele mais tarde

```
namespace Exercicios_INF { ... }
```

- Indica que o espaço de nomes (**Exercicios\_INF**) onde vamos colocar o nosso programa
- Os espaços de nomes podem ser pré-definidos pelo sistema (ambiente) ou definidos pelo programador.
- De momento, este conceito não nos interessa, voltaremos a ele mais tarde

```
class PrimeiroPrograma { ... }
```

- Declara a classe PrimeiroPrograma e dentro dela os seus elementos: variáveis, métodos (funções), constantes, etc.
- Os vários elementos de uma classe serão explicados em detalhe ao longo da disciplina
- Os elementos de uma classe são declarados sempre entre as chavetas { e }, que delimitam a classe
- De momento, este conceito não nos interessa, voltaremos a ele mais tarde e será o conceito base da disciplina de segundo semestre – Programação Orientada por Objectos

```
static void Main(string[] args) { ... }
```

- Declara o método **Main()**, que define o ponto de início de execução do programa.
- Em C#, um 'programa' é uma classe com um ponto de início de execução, definido através do método **Main()**.
- **static** e **void** serão explicados mais adiante.
- Métodos em C# são análogos a funções em C ou Pascal
- Vamos colocar entre chavetas as instruções do nosso programa
- De momento, este conceito não nos interessa (voltaremos a ele mais tarde) mas é importante saber que as nossas instruções serão sempre colocadas dentro das chavetas do **Main**

```
Console.WriteLine("Ola Mundo !!!");
```

- comando usado para escrever um texto (entre aspas) através da consola. A classe Console é definida no módulo System e esta por sua vez define o método WriteLine().
- este é um comando de activação ou chamada de método (ou função)

## comentários

- Anotações no código fonte que não são consideradas pelo compilador
- comentário iniciado por `//`, restrito a uma única linha

```
// Primeiro programa em C#
```

- Comentário delimitado por `/*` e `*/`, podendo ocupar várias linhas

```
/* escrever uma mensagem pela consola */
```



## Utilização do Visual Studio express edition

*Como utilizar na disciplina*

Download em <http://www.microsoft.com/express/>

Ambiente de trabalho muito poderoso e complexo

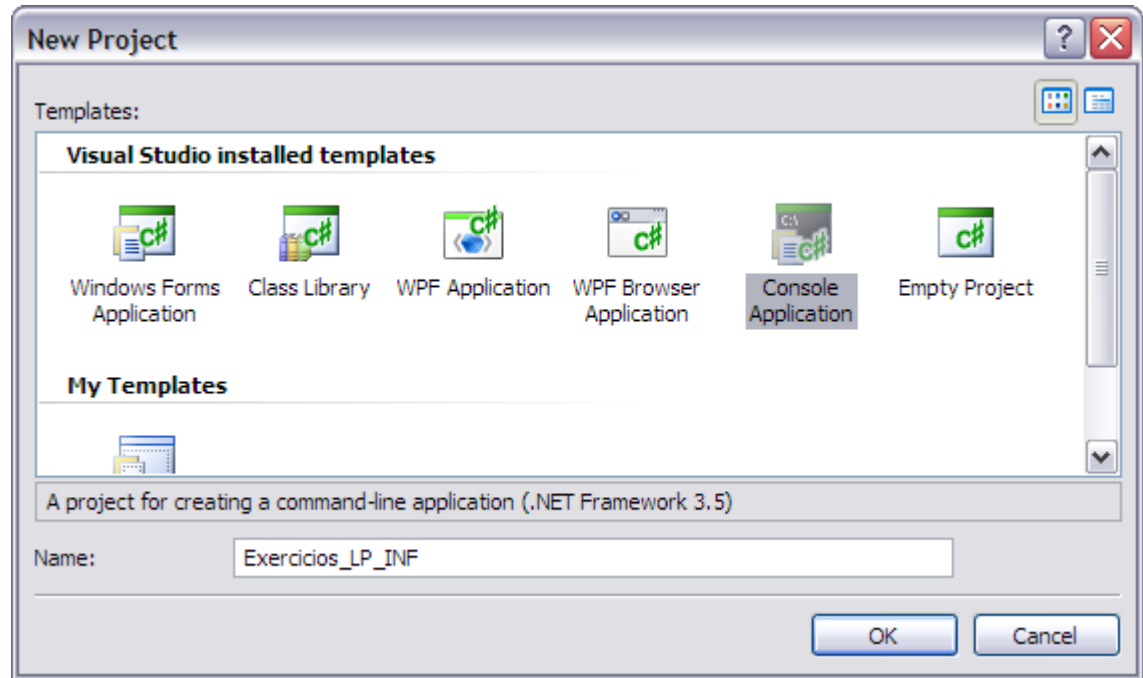
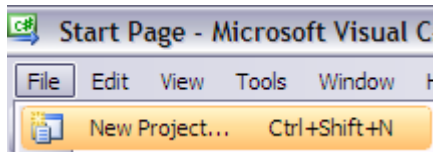
Utilização progressiva das suas potencialidades

Aplicação da máxima “é melhor saber fazer bem de uma maneira de que mal de várias”

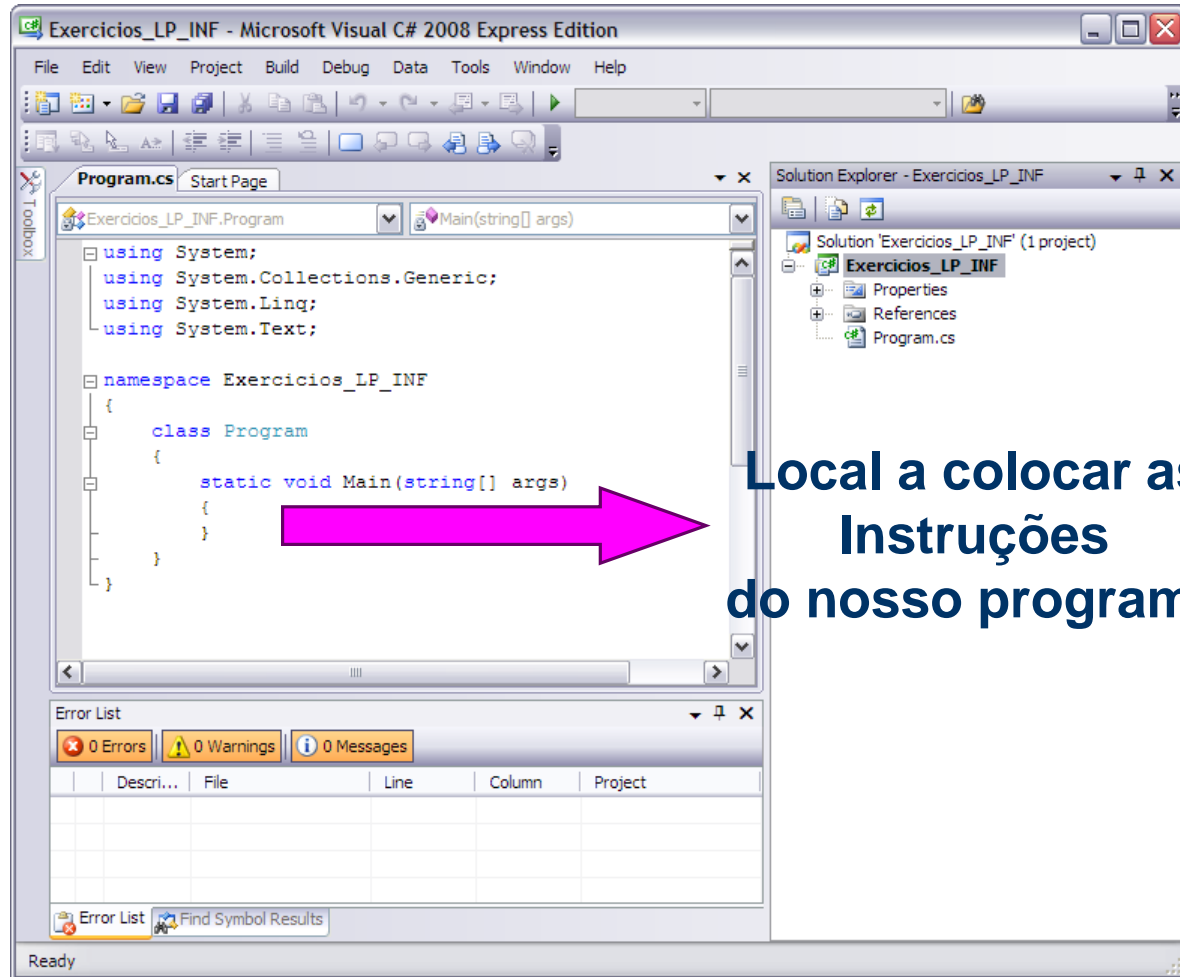
## Importante

Definir uma forma de trabalho que nos permita “andar” com todos os exemplos e exercícios independentemente do local de trabalho (casa, escola, sala de informática, portátil, ...) => controlo apertado do nosso trabalho

## Passo1: Criar projecto que contém o template do meu primeiro programa

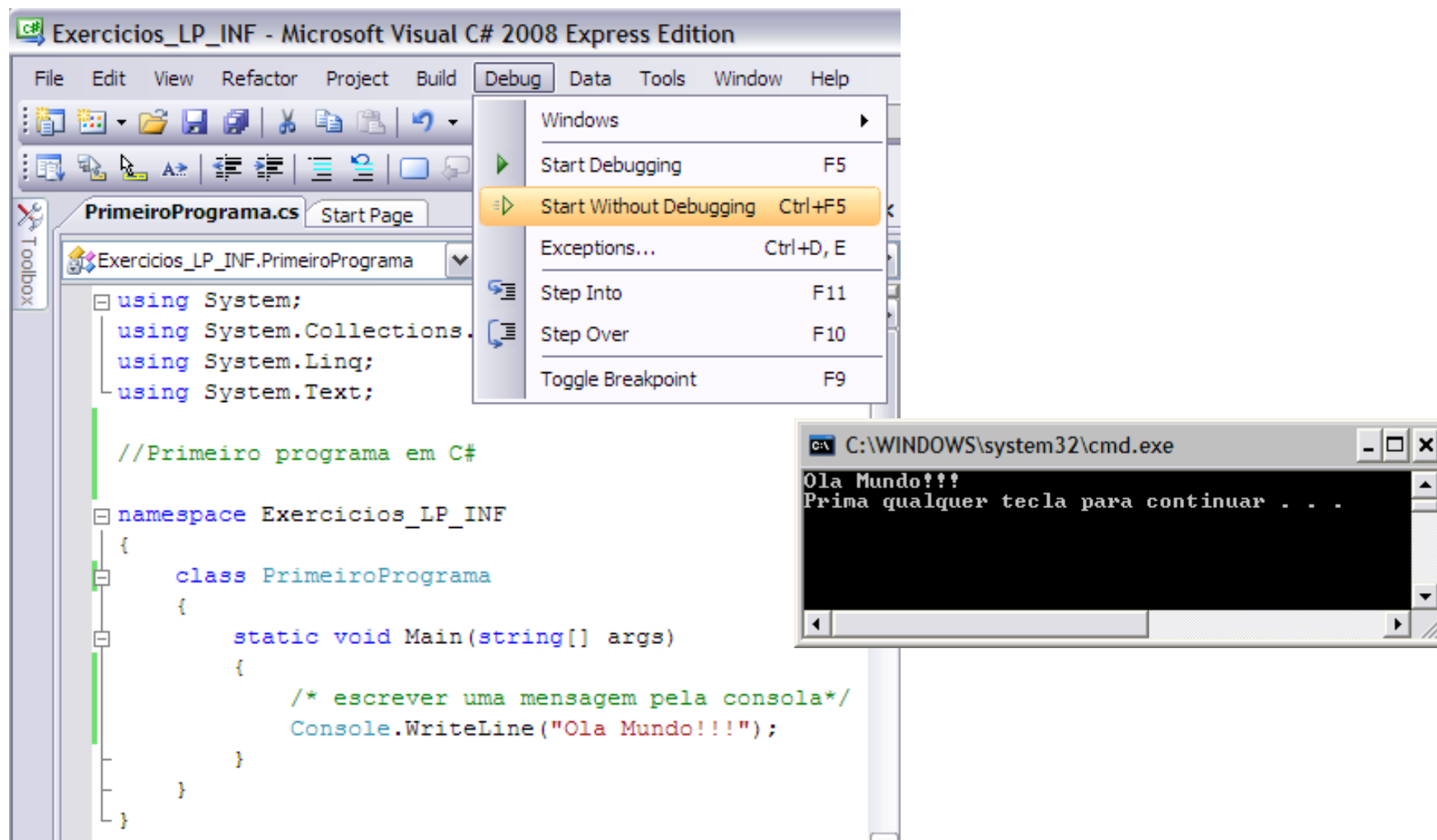


## Passo 1: Criar projecto que contém o template do meu primeiro programa

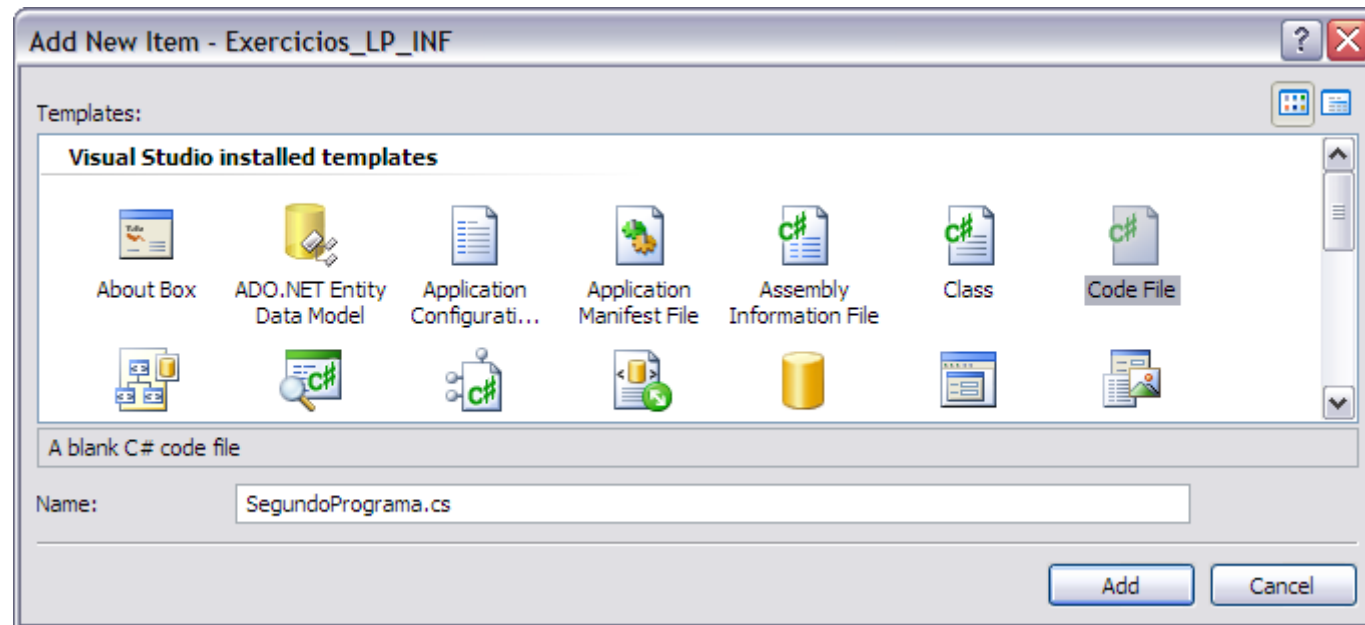
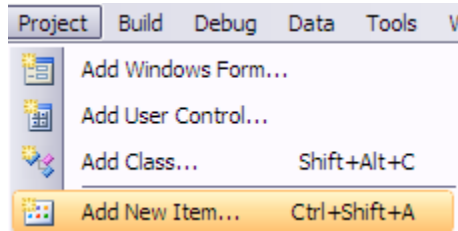


**Local a colocar as  
Instruções  
do nosso programa**

## Passo 3: Criar, compilar e executar o primeiro programa

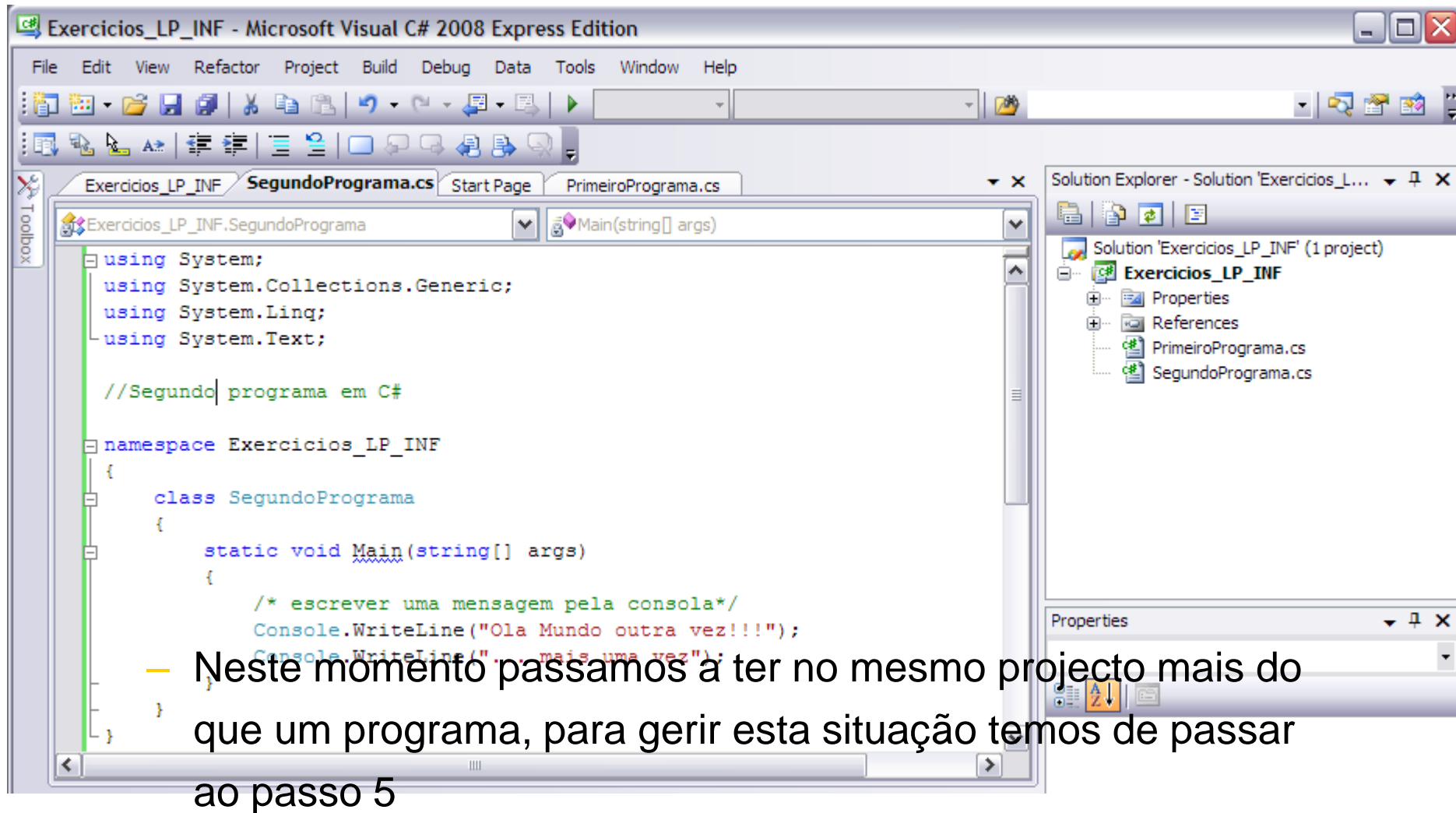


## Passo 4: Criar, compilar e executar o segundo programa



- É criado um ficheiro SegundoPrograma.cs
- Copiar para este ficheiro o template de um programa
- Alterar o nome da classe para SegundoPrograma
- Introduzir as instruções pretendidas

## Passo 4: Criar, compilar e executar o segundo programa



The screenshot displays the Microsoft Visual C# 2008 Express Edition IDE. The main editor window shows the code for 'SegundoPrograma.cs'. The code includes using statements for System, System.Collections.Generic, System.Linq, and System.Text. It defines a namespace 'Exercicios\_LP\_INF' containing a class 'SegundoPrograma' with a static 'Main' method. The 'Main' method contains two console write lines: 'Ola Mundo outra vez!!!' and '... mais uma vez'. The Solution Explorer on the right shows the solution 'Exercicios\_LP\_INF' with two projects: 'PrimeiroPrograma.cs' and 'SegundoPrograma.cs'. The Properties window is also visible at the bottom right.

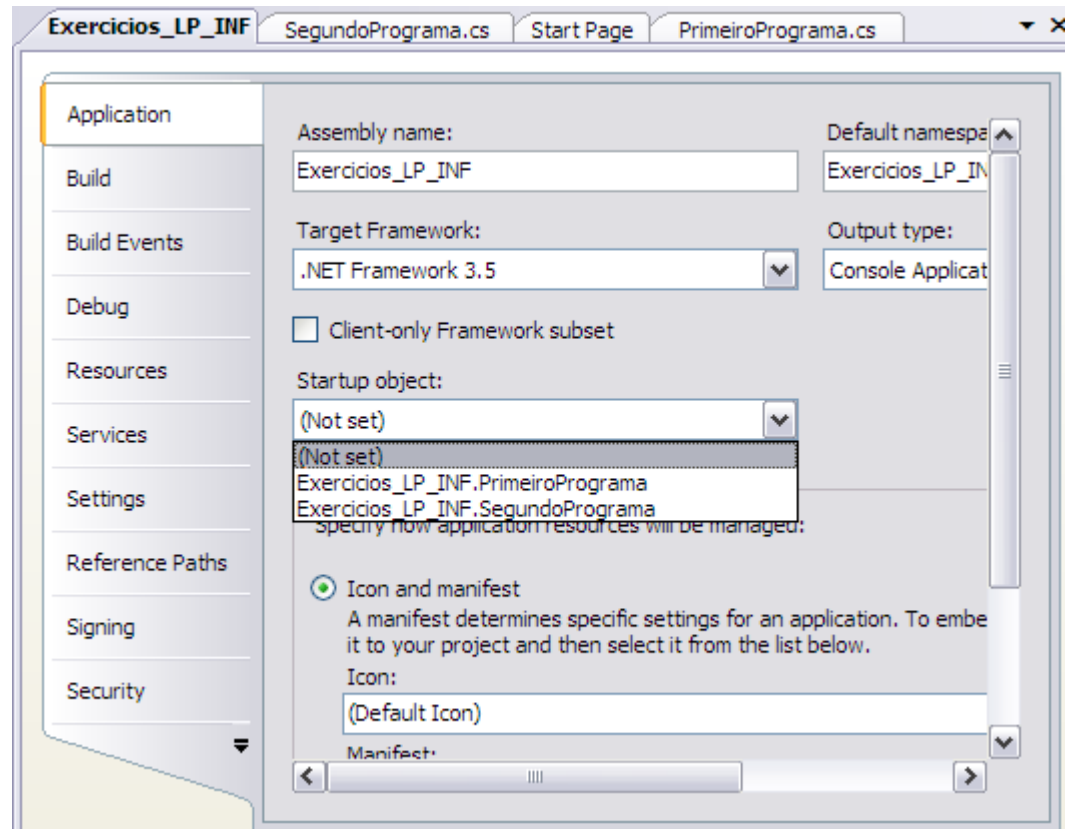
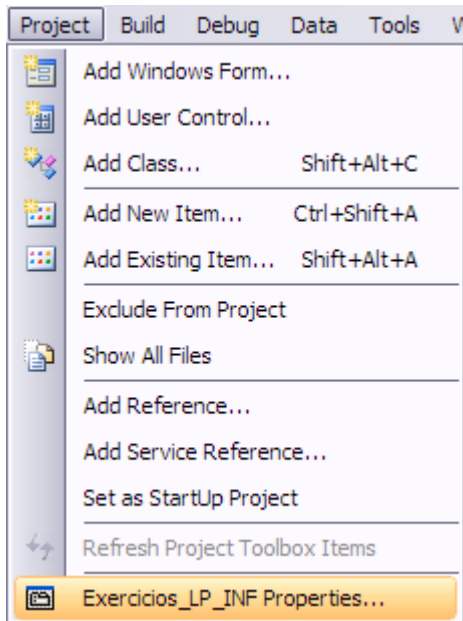
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

//Segundo programa em C#

namespace Exercicios_LP_INF
{
    class SegundoPrograma
    {
        static void Main(string[] args)
        {
            /* escrever uma mensagem pela consola*/
            Console.WriteLine("Ola Mundo outra vez!!!");
            Console.WriteLine("... mais uma vez");
        }
    }
}
```

Neste momento passamos a ter no mesmo projecto mais do que um programa, para gerir esta situação temos de passar ao passo 5

## Passo 5: Gerir vários programas num mesmo projecto



Vantagem: todos os nossos programas vão estar num mesmo projecto o que ajuda à sua gestão

# Variáveis e memória

## Variáveis

Uma variável é uma área de memória, identificada por um nome, que pode armazenar valores de um determinado tipo

Operações que podem ser efectuadas sobre variáveis

Utilização do seu valor (  $x + 1$  )

Atribuição de um valor novo (  $x = 5$  )

O tipo da variável permite determinar os valores que esta pode conter e as operações em que pode participar

C# é uma linguagem fortemente tipada (todas as variáveis têm obrigatoriamente um tipo) e um valor pertencente a um tipo só pode ser usado como argumento em operações que prevêm argumentos desse tipo



# Variáveis e memória

## Variáveis

Os valores (dados) que uma variável contém ao longo da sua existência são de um só tipo

A declaração de uma variável tem de conter a indicação do tipo de dados que vai conter

Se atribuir à variável um valor que não pertence ao tipo especificado é obtido um erro de compilação

## Declaração

```
tipoDaVariavel nomeDaVariavel = valorInicial;  
tipoDaVariavel nomeDaVariavel;
```

# Variáveis e memória

## Declaração

```
tipoDaVariavel nomeDaVariavel = valorInicial;  
tipoDaVariavel nomeDaVariavel;
```

Os valores (dados) que uma variável contém ao longo da sua existência são de um só tipo

A declaração de ma variável tem de conter a indicação do tipo de dados que vai conter

Se atribuir à variável um valor que não pertence ao tipo especificado é obtido um erro de compilação

# Variáveis e memória

## Operação de atribuição ( = )

`variável = expressão`

Num dado instante uma variável apenas tem um valor

`x = 5`

O valor anteriormente contido na variável `x` é perdido e a variável passa a ter o valor 5

`x = x + 55`

O valor anteriormente contido na variável `x` é perdido e a variável passa a ter a soma do valor anterior com 55

# Constantes

## Declaração

```
const tipoDaConst nomeDaConst = valor;
```

Igual a uma variável mas o seu conteúdo não pode ser alterado

# Tipos de dados primitivos

## Tipos de dados numéricos

int e double são os mais importantes

Tipo	Faixa de Valores	Tamanho
sbyte	-128 até 127	8 bits
byte (sem sinal)	0 até 255	8 bits
char (unicode)	U+0000 até U+ffff	16 bits
short	-32.768 até 32.767	16 bits
ushort (sem sinal)	0 até 65.535	16 bits
int	-2.147.483.648 até 2.147.483.647	32 bits
uint (sem sinal)	0 até 4.294.967.295	32 bits
long	-9.223.372.036.854.775.808 até 9.223.372.036.854.775.807	64 bits
ulong (sem sinal)	0 até 18.446.744.073.709.551.615	64 bits

Tipo	Faixa de Valores	Precisão	Tamanho
float	$1.5 \times 10^{-45}$ até $3.4 \times 10^{38}$	7 dígitos	32 bits
double	$5.0 \times 10^{-324}$ até $1.7 \times 10^{308}$	15-16 dígitos	64 bits

Tipo	Faixa de Valores	Precisão	Tamanho
decimal	$1.0 \times 10^{-28}$ até $7.9 \times 10^{28}$	28 dígitos	128 bits

# Tipos de dados primitivos

## Tipo Lógico ou Boolean

O tipo boolean é usado em geral para operações lógicas.

Os valores possíveis para uma variável deste tipo são true e false

- `bool nomevar;`

# Tipos de dados primitivos

## Tipo Caracter

```
char nomeVar;
```

```
nomeVar = 'a';
```

```
nomeVar = '\u0041'; //formato unicode
```

```
nomeVar = '\n'; //caracter nova linha
```

# Outros tipos de dados

## Tipo String (sequência de caracteres)

```
string nomeVar;
```

```
nomeVar = "Isto e uma string";
```

```
nomeVar = "Outra string...?";
```

```
nomeVar = "Inicio\tFim\n Nova linha";
```

```
//experimentar esta
```



# Expressões e Operadores

## Expressões

Uma expressão é uma combinação de operandos e operadores.

Expressões em C# são semelhantes às expressões usadas em outras linguagens como C ou Pascal

No caso de operações encadeadas, como em  $a+b*c$  o cálculo da expressão é feito de acordo com a precedência entre os operadores.

Parênteses são utilizados para alterar a ordem de cálculo das operações -  $(a+b)*c$       $a+(b*c)$

# Expressões e Operadores

## Expressões

As operações são realizadas numa expressão de acordo com a ordem de prioridade. Quando houver empate entre operações, elas serão realizadas da esquerda para a direita.

$6/2/3$  é calculado como  $(6/2)/3$

neste caso a associatividade é à esquerda

Em C#, apenas os operadores de atribuição

$(= \ * \ /= \ \% \ += \ -= \ <<= \ >>= \ \&= \ ^= \ |=)$

e o operador ternário  $(? :)$

têm associatividade à direita.

# Expressões e Operadores

## Principais operadores em ordem de precedência

<b>Categoria</b>	<b>Operadores</b>	<b>Associatividade</b>
Unário	<code>+</code> <code>-</code> <code>!</code>	esquerda
Multiplicativo	<code>*</code> <code>/</code> <code>%</code>	esquerda
Aditivo	<code>+</code> <code>-</code>	esquerda
Relacional	<code>&lt;</code> <code>=</code> <code>&gt;</code> <code>&gt;=</code> <code>&lt;=</code> <code>==</code> <code>is</code>	esquerda
Igualdade	<code>==</code> <code>!=</code>	esquerda
'and' (bool)	<code>&amp;&amp;</code>	esquerda
'ou' (bool)	<code>  </code>	esquerda

# Expressões e Operadores

## Operadores de atribuição combinados

O operador de atribuição pode ser combinado com outros operadores aritméticos

Operador	Significado
=	Atribuição simples
+=	Atribuição com adição
-=	Atribuição com subtração
*=	Atribuição c/ multiplicação
/=	Atribuição c/ divisão
%=	Atribuição do resto

# Expressões e Operadores

## Operadores relacionais

Comparação entre valores do mesmo tipo;

O resultado será sempre do tipo lógico (true,false);

O operador relacional perde em prioridade para os operadores aritméticos

Operador	Significado	Exemplo de uso
>	Maior que	$a > b$ , $4 > 3$ , $c > 3$
<	Menor que	$2 * a > b * 2$
!=	Diferente de	$a + b != c$
==	Igual a	$a == 0$
<=	Menor ou igual a	$b * a <= 0$
>=	Maior ou igual a	$a >= b$

## Operadores lógicos

Utilizados no processo de tomada de decisão e controles de repetição

O resultado será sempre do tipo lógico (true,false);

**&&** - usado na conjunção (E) de proposições

**||** - usado na disjunção (OU) de proposições

**!** - usado na negação (NOT) de proposições

**^** - disjunção exclusiva

Proposição é qualquer elemento que possa produzir um valor lógico (variável lógica, expressão relacional ou expressão lógica);

## **Expressões de controlo de fluxo**

Permitem alterar o fluxo de execução de um programa

O fluxo de execução é por defeito sequencial

## **Expressões condicionais**

## **Expressões de selecção de alternativa**

## **Expressões de repetição com base numa condição**

## **Expressões de repetição com base num contador**

## **Quebra de Ciclos**

## Expressões condicionais (IF-THEN)

o bloco de instruções (ou a instrução) é executada se a condição lógica for verdadeira

```
if (<condição>)  
{  
    <bloco de instruções>  
}
```

```
if (<condição>) <instrução>;
```



## Expressões condicionais (IF-THEN-ELSE)

se condição lógica for verdadeira é executado o primeiro bloco de instruções (ou a instrução) senão é executado o segundo bloco

```
if (<condição>)  
{  
    <primeiro bloco de instruções>  
}  
else  
{  
    <segundo bloco de instruções>  
}
```

## Expressões de selecção de alternativa (switch)

É executado um bloco de instruções de acordo com o valor de uma variável

```
switch (<variável>)  
{  
    case <valor>:  
        <bloco de instruções>  
        break;  
  
    ...  
  
    case <valor>:  
        <bloco de instruções>  
        break;  
  
    ...  
  
    default:  
        <bloco de instruções>  
        break;  
  
}
```

## Expressões de repetição com base numa condição

É executado um bloco de instruções de acordo com o valor de uma variável

Primeiro é testada a condição, só se esta for verdadeira o bloco de instruções é executado

```
while (<condição>)  
{  
    <bloco de instruções>  
}
```

## Expressões de repetição com base numa condição

É executado um bloco de instruções de acordo com o valor de uma variável

Primeiro é executado o bloco de instruções, depois é testada a condição, se esta for verdadeira o bloco volta a ser repetido

```
do
{
    <bloco de instruções>
}
while (<condição>) ;
```

## Expressões de repetição com base num contador

É executado um bloco de instruções de acordo com o incremento de uma variável enquanto uma condição for verdadeira

Primeiro são executadas as operações sobre a variável de controlo, e só depois é executado o bloco de instruções

```
for (<inicialização>; <condição>; <actualização>;)
{
    <bloco de instruções>
}
```

## **Quebra de ciclos - break**

Em qualquer bloco de instruções numa expressão de repetição o comando break termina o ciclo

O seu uso leva a uma Programação pouco estruturada

# Aninhamento de estruturas de Controlo de Fluxo

**Uma expressão de repetição é no seu todo um comando como outro qualquer**

Pode fazer parte de um qualquer bloco de instruções

As combinações podem ser as mais variadas, tendo apenas de respeitar este principio de que esta expressão é um comando como outro qualquer

# Interatividade

## Escrita de valores através da consola

O parâmetro da expressão tem de ser uma string, ou uma lista de parâmetros em que o primeiro é uma string

```
Console.WriteLine("Escreve e muda de linha");  
Console.Write("Escreve mas não muda de linha");
```

```
int x = 2;
```

```
string nome = "abc";
```

```
Console.Write("primeiro={0} segundo={1}", x, nome);
```

```
Console.Write("primeiro=" + x + "segundo=", nome);
```



# Interatividade

## Leitura de valores através da consola

O resultado da operação é sempre uma string que terá de ser convertido para posterior utilização

```
string linha = Console.ReadLine();
```

```
int x = Int32.Parse(linha);
```

```
int y = Int.Parse(linha);
```

```
string nome = "abc" + linha;
```